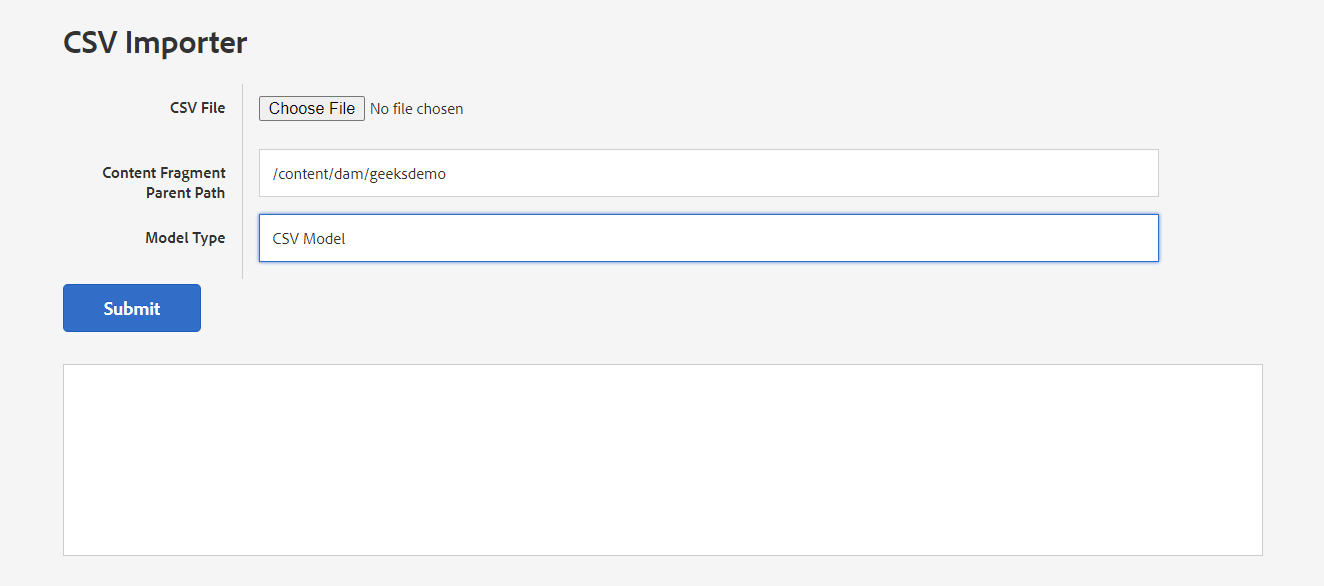
**Write a AEM utility , which can import the excel sheet data and create the content fragments in AEM**

****

**HTML CODE**

**<form id="form">**

**<div class="form-row">**

**<h4 acs-coral-heading class="coral-Heading coral-Heading--4">CSV File</h4>**

**<span>**

**<input accept="\*/\*"**

**type="file"**

**name="csv"**

**ngf-select**

**required**

**placeholder="Select the Categories CSV file"/>**

**</span>**

**</div>**

**<div class="form-row">**

**<h4 acs-coral-heading class="coral-Heading coral-Heading--4">Content Fragment Parent Path</h4>**

**<span>**

**<input type="text"**

**name="parentPath"**

**class="coral-Textfield"**

**required="true"**

**value="/content/dam/geeksdemo"**

**placeholder=""/>**

**</span>**

**</div>**

**<div class="form-row">**

**<h4 acs-coral-heading class="coral-Heading coral-Heading--4">Model Type</h4>**

**<span>**

**<select required="true" name="modelType" class="coral-Textfield">**

**<option value="csv\_model">CSV Model</option>**

**</select>**

**</span>**

**</div>**

**<div class="form-row">**

**<div class="form-left-cell">&nbsp;</div>**

**<button onclick="createContentFragment();return false;" class="coral-Button coral-Button--primary">Submit</button>**

**</div>**

**</form>**

**JAVASCRIPT FOR TRIGGER THE SERVLET**

**function createContentFragment() {**

**var form = $('form')[0]; // You need to use standard javascript object here**

**var formData = new FormData(form);**

**//These lines select the first form element on the page using jQuery ($('form')[0]) and store it in the variable form. Then,**

**//it creates a new FormData object named formData from the selected form. FormData objects are used to send form data to the server via AJAX requests.**

**$(".results").html("");**

**$.ajax({**

**url: '/bin/content/createContentFragment',**

**data: formData,**

**method: 'POST',**

**contentType: false, // NEEDED**

**processData: false, // NEEDED**

**// This block initiates an AJAX request using jQuery's $.ajax() method. It specifies the URL to which the request will be sent (url),**

**// the data to be sent (data), the HTTP method (method), and other settings.**

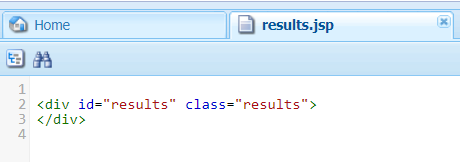
**// contentType: false and processData: false are set to ensure that jQuery does not attempt to process the FormData object in any way.**

**// This is necessary when sending FormData in an AJAX request.**

**success: function(data) {**

**console.log("Success");**

**$(".results").html(data.status);**

****

**//This line clears the content of the element with the class "results" using jQuery. It's clearing any previous results before making a new request.**

**// jsout.name("status").value("Success Created total of " + resourceList.size() + " Content Fragments");(this is java line written in java).name=status,**

**//through this it prints the**

**}**

**//If the AJAX request is successful, the success callback function is executed. It receives the response from the server as the data parameter.**

**// Here, it logs "Success" to the console and updates the content of the element with the class "results" with the value of data.status**

**error: function(data) {data**

**console.log("error", data);**

**$(".results").html(data.responseText);**

**}**

**// If the AJAX request encounters an error, the error callback function is executed. It also receives the response from the server as the data parameter.**

**// Here, it logs the error data to the console and updates the content of the element with the class "results" with the error message (data.responseText).**

**------------------------------------------------------------------------------------------------------------------------------**

public class CreateGenericContentFragment extends SlingAllMethodsServlet {

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

private static final long serialVersionUID = 1L;

protected static final String SERVICE\_DESCRIPTION = "Create Generic Content Fragment from CSV";

protected static final String CF\_TEMPLATE = "/conf/geeksdemo/settings/dam/cfm/models/";

private static final String PARENT\_PATH = "parentPath";

private static final String MODEL\_TYPE = "modelType";

private static final String CSV = "csv";

@Override

For creating any node we are using “dopost” ,For retrieveing we use “doget” method

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

LOGGER.debug("Received POST request to create content fragment.");

request.setCharacterEncoding("UTF-8");

response.setContentType("application/json");

response.setHeader("Cache-Control", "nocache");

response.setCharacterEncoding("utf-8");

Using JSON (JavaScript Object Notation) is common in web development for exchanging data between a

server and a client

JsonWriter jsout = new JsonWriter(response.getWriter());

jsout.beginObject();

🡪USING THIS JSOUT VARIABLE WE CAN PRINT THE DATA IN THE FRONTEND

LOGGER.info("Inside doPost method.");

LOGGER.info(CF\_TEMPLATE);

LOGGER.info(MODEL\_TYPE);

LOGGER.info("parent\_pathg :"+PARENT\_PATH);

//content/dam/geeksdemo

String parentPath = request.getParameter(PARENT\_PATH);

//CSV Model

String modelType = request.getParameter(MODEL\_TYPE);

LOGGER.info("parent\_path :"+parentPath);

LOGGER.info("modelType :"+modelType);

ResourceResolver resolver = request.getResourceResolver();

try {

//import org.apache.commons.lang3.StringUtils;

The **StringUtils** class offers a wide range of methods for common string operations such as trimming, concatenation, padding, searching, and replacing. These methods are often more convenient and efficient than writing the equivalent code from scratch.

For example, you can use **StringUtils.isBlank()** to check if a string is **null**, empty, or contains only whitespace characters. Similarly, **StringUtils.capitalize()** can be used to capitalize the first letter of a string.

if (StringUtils.isNotEmpty(parentPath) && StringUtils.isNotEmpty(modelType)) {

LOGGER.info("Parent path and model type are not empty.");

// StringUtils.i

//Here CSV means , The uploaded file take it as a CSV

Part filePart = request.getPart(CSV);

The “request.getParts()” method returns collections of all Part objects. If you have more than one input of type file, multiple Part objects are returned. Since Part objects are named, the getPart(String name) method can be used to access a particular Part. Alternatively, the getParts() method, which returns an Iterable<Part>, can be used to get an Iterator over all the Part objects.

The javax.servlet.http.Part interface is a simple one, providing methods that allow introspection of each Part. The methods do the following:

* Retrieve the name, size, and content-type of the Part

InputStream fileContent = filePart.getInputStream();

1. **Create a FileInputStream**: A **FileInputStream** is created using the **File** object. This class allows you to open a file in read mode, and it provides methods to read bytes from the file.
2. **Wrap in InputStream**: To make it more flexible and compatible with other methods that accept **InputStream**, the **FileInputStream** is wrapped within an **InputStream** reference. This is done by typecasting or using a constructor that takes an **InputStream** parameter.

LOGGER.info("fileContent :"+fileContent);

In the loggers I am getting this output “fileContent :java.io.ByteArrayInputStream@465d06e6”

CSVReader header = new CSVReaderBuilder(new InputStreamReader(fileContent)).withSkipLines(0).build();

1. **new InputStreamReader(fileContent)**: This creates an **InputStreamReader** object, which converts bytes from the input stream (**fileContent**) into characters. In this case, **fileContent** is assumed to be an **InputStream** obtained from reading a CSV file.
2. **new CSVReaderBuilder(...)**: This creates a builder for constructing instances of **CSVReader**, which is a class provided by OpenCSV for reading CSV files. It takes an **InputStreamReader** as its parameter, so it can read characters from the input stream.
3. **withSkipLines(0)**: This method configures the **CSVReader** to skip a certain number of lines before reading from the CSV file. In this case, **0** is passed, indicating that no lines should be skipped, so the reader starts reading from the beginning of the file.
4. **build()**: This method finalizes the construction of the **CSVReader** instance with the specified configuration.

// Read the header line from the CSV file and store it as a list of strings

List<String> headerList = Arrays.asList(header.readNext());

LOGGER.info("headerList :"+headerList);

In the loggers I am getting this output headerList :[EmployeeID, FirstName, LastName]

JcrNodeResource, type=dam:Asset, superType=null, path=/content/dam/geeksdemo/10

JcrNodeResource, type=dam:Asset, superType=null, path=/content/dam/geeksdemo/2

JcrNodeResource, type=dam:Asset, superType=null, path=/content/dam/geeksdemo/3

1. **header.readNext()**: This method reads the next line from the CSV file being processed by the **CSVReader** object **header**. In this case, since **CSVReader** is configured to start reading from the beginning of the file and no lines are skipped (**withSkipLines(0)**), **readNext()** will read the first line of the CSV file, which is typically the header line containing column names.
2. **Arrays.asList(...)**: This method is used to create a **List** containing the elements of the array returned by **header.readNext()**. It converts the array of column names (obtained from **header.readNext()**) into a **List<String>**.
3. **List<String> headerList = ...**: This line declares a new variable **headerList** of type **List<String>** and assigns it the list of column names obtained from **header.readNext()**.

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

🡪This piece of code is is looking in the place I am replacing another code

// List<String> resourceList = header.readAll().stream()

// .filter(data -> StringUtils.isNotEmpty(data[0]))

// .map(data -> {

// createContentFragment(resolver, parentPath, headerList, data, modelType);

// return "Yes";

// }).collect(Collectors.toList());

**header.readAll()**: This method reads all the remaining lines from the CSV file and returns them as a **List<String[]>**. Each **String[]** represents a row of data in the CSV file.

1. **.stream()**: This converts the list of rows into a stream of rows, which allows us to use the Stream API for further processing.
2. **.filter(data -> StringUtils.isNotEmpty(data[0]))**: This filters the rows based on a condition. It checks if the first column of each row is not empty. If the condition is **true**, the row passes through the filter, otherwise, it's excluded from further processing.
3. **.map(data -> { ... })**: This maps each filtered row to a new value. Inside the **map** function, it invokes the **createContentFragment** method with some parameters, performs some action (in this case, it's creating content fragments), and then returns "Yes". This "Yes" is just a placeholder value as the return type of **map** needs to be consistent.
4. **.collect(Collectors.toList())**: This collects the mapped values into a list.

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

* This line creates a new **ArrayList** named **resourceList** to store the results of processing each row of the CSV file.

List<String[]> allRows = header.readAll(); // Read all rows from the CSV file

* This line reads all rows from the CSV file using the **CSVReader** object **header** and stores them in a list named **allRows**. Each element of **allRows** is an array of strings, where each string represents a cell value in the CSV.

**for** (String[] data : allRows) {

* This line starts a for-each loop that iterates over each row in the **allRows** list. Each iteration assigns the current row to the **data** variable.

**if** (StringUtils.*isNotEmpty*(data[0])) { // Check if the first column of the row is not empty

* This line checks if the first element of the current row (**data[0]**) is not empty. **StringUtils.isNotEmpty()** is a method from the Apache Commons Lang library that checks if a string is not **null** and not empty.

createContentFragment(resolver, parentPath, headerList, data, modelType);

This line calls the createContentFragment method, passing several parameters: resolver, parentPath, headerList, data, and modelType. This method likely processes the current row in some way.

resourceList.add("Yes");

* This line adds the string **"Yes"** to the **resourceList** regardless of whether the condition in the if statement was true or not. This appears to be a placeholder or indicator that a row was processed successfully.

}

}

***LOGGER***.info("resources size :"+resourceList.size());

This code essentially reads each row from the CSV file, checks if the first column of each row is not empty, processes the row with **createContentFragment** if it meets the condition, and adds "Yes" to the **resourceList**.

jsout.name("status").value("Success Created total of " + resourceList.size() + " Content Fragments");

}

} catch (IOException | ServletException e) {

LOGGER.error("Exception occurred while processing POST request.", e);

jsout.name("status").value("Failed Created all Content Fragments");

} finally {

if (resolver != null && resolver.hasChanges()) {

resolver.commit();

}

}

jsout.endObject();

}

//From the above code we are getting the details about cf headers and cf data

public void createContentFragment(ResourceResolver resourceResolver, String parentPath, List<String> headerList, String[] data, String modelType) {

// LOGGER.info("In create content fragment :"+data[0] +" "+data[1]+" "+data[2]);

try {

//modelResource=/content/dam/mysite/data[0] , data[0] means cf name or element name if element name how can create the cf element without creating the cf

Resource modelResource = resourceResolver.getResource(parentPath + "/" + data[0]);

LOGGER.info(String.valueOf(modelResource));

ContentFragment newFragment = null;

if (modelResource == null) {

//From this line we are getting this path "templateOrModelRsc=/conf/mysite/settings/dam/cfm/models/csv-model"

Resource templateOrModelRsc = resourceResolver.getResource(CF\_TEMPLATE + modelType);

LOGGER.info("CF\_TEMPLATE :"+CF\_TEMPLATE);

LOGGER.info("templateOrModelRsc :"+templateOrModelRsc);

Resource parentResource = resourceResolver.getResource(parentPath);

FragmentTemplate tpl = templateOrModelRsc.adaptTo(FragmentTemplate.class);

newFragment = tpl.createFragment(parentResource, data[0], data[0]);

} else {

newFragment = modelResource.adaptTo(ContentFragment.class);

}

if (newFragment != null) {

Iterator<ContentElement> elements = newFragment.getElements();

while (elements.hasNext()) {

ContentElement element = elements.next();

for (int i = 0; i < headerList.size(); i++) {

if (StringUtils.equalsIgnoreCase(element.getName(), headerList.get(i)) || StringUtils.containsAny(element.getName())) {

element.setContent(data[i], element.getContentType());

break;

}

}

}

}

} catch (ContentFragmentException e) {

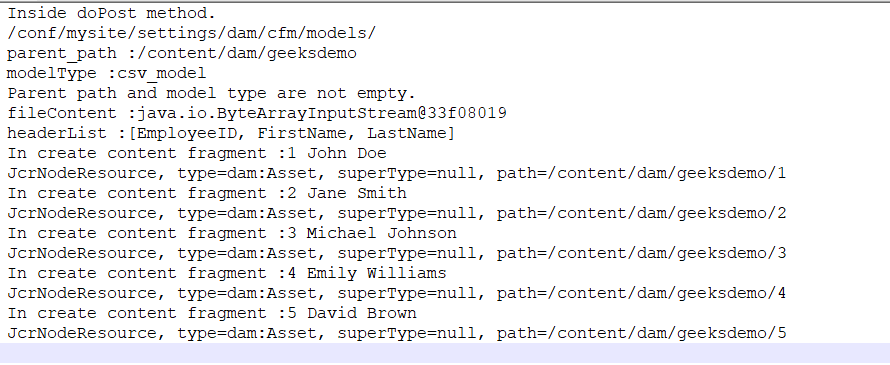
LOGGER.error("Exception occurred while creating content fragment.", e);

}

}

}

**LOGS:**

****

**CREATECONTENT FRAGMENT METHOD**

**This Method is used for create the content fragments**

createContentFragment(ResourceResolver resourceResolver, String parentPath, List<String> headerList, String[] data, String modelType)

**CASE-1:**

**We are getting below parameters from the** createContentFragment method.

**LOGS DETAILS REGARDING PARAMETERS**

**Resource Resolver:** [**org.apache.sling.resourceresolver.impl.ResourceResolverImpl@5307b3c6**](mailto:org.apache.sling.resourceresolver.impl.ResourceResolverImpl@5307b3c6)

**Parent Path: /content/dam/geeksdemo**

**Header List: [EmployeeID, FirstName, LastName]**

**Data: [3, Michael, Johnson]**

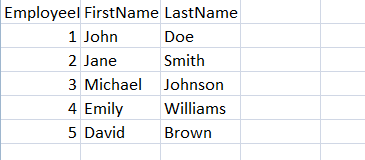
**Model Type: csv\_model**

**In create content fragment :3 Michael Johnson**

**CASE-2:**

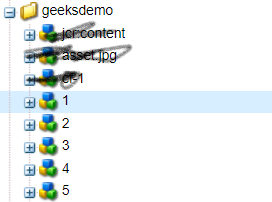
Resource modelResource = resourceResolver.getResource(parentPath + "/" + data[0]);

**THIS IS CSV FILE DATA , WITH THIS CSV FILE WE CAN ABLE CERATE 5 CONTENT FRAGMENTS**



**HERE 1 ST CONTENT FRAGMENT DETAILS IMAGE**





From this line we are getting “modelResource= /content/dam/geeksdemo/1” .

1—CF Name

Based on above csv file we will get the 5 cf’s modelResource’s

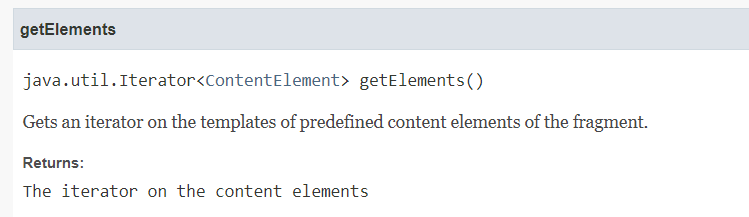
**CASE-3:**

ContentFragment newFragment = modelResource.adaptTo(ContentFragment.**class**);

**It seems like you're referring to a "contentfragment" class. In software development, a class is a blueprint for creating objects**

**CASE-4:**

Iterator<ContentElement> elements = newFragment.getElements();

****

****

**getElements():**

**This method is called on the “newFragment” object. It presumably retrieves a collection of ContentElement**

**An iterator is a tool used to traverse through a collection of elements, allowing you to access each element in sequence.**