***workflow process in process step***

***"process.label" + " = Geeks Workflow Process":***

The label that appears in the AEM workflow process step dropdown.

***public class GeeksWorkflowProcess implements WorkflowProcess { }:***

**Workflow Process**

A workflow process in AEM is a fundamental part of creating workflows that automate tasks such as content approval, publishing, asset management, and more. By implementing the WorkflowProcess interface, you can inject custom logic into these workflows, enabling complex automation that fits your business needs.

When a class implements the WorkflowProcess interface, it must provide an implementation for the execute method defined in the interface. This method is where you define the custom logic that should run when the workflow step is executed.

***public void execute(WorkItem workItem, WorkflowSession workflowSession, MetaDataMap processArguments) :***

**WorkItem workItem**

The WorkItem represents the current item being processed in the workflow. It provides access to the workflow instance and the data associated with this workflow step.

Key Methods of WorkItem:

* **getWorkflow**: Returns the workflow instance this item is part of.
* **getWorkflowData**: Returns the data associated with this item, typically the payload (e.g., a JCR path or a resource).

Certainly! Let's dive deeper into the execute method of a workflow process in AEM, focusing on the workItem, workflowSession, and processArguments parameters.

**WorkflowSession workflowSession**

The WorkflowSession provides a session to interact with the workflow engine. It allows you to manage the workflow process, access workflow models, and perform operations related to workflows.

If we need Resourec Resolver we can inject Resource Resolver Factory into the class but in workflows we can not directly inject ,we can get the ResourceReslover by using workflowsession in workflows .

***Session session = workflowSession.adaptTo(Session.class);***

Key Methods of WorkflowSession:

* **adaptTo**: Allows adapting the WorkflowSession to other interfaces, such as Session (JCR session).
* **getSession**: Returns the JCR session associated with the workflow session.
* **terminateWorkflow**: Allows terminating a workflow instance.

**MetaDataMap processArguments**

The MetaDataMap contains the arguments passed to this workflow process step. These arguments can be configured when setting up the workflow model in AEM.

Key Methods of MetaDataMap:

* **get**: Retrieves a value for a given key. You can also specify a default value if the key is not present.
* **containsKey**: Checks if a specific key is present.
* **keySet**: Returns a set of all keys in the map.

WorkflowData workflowData = workItem.getWorkflowData();

 **workItem**: This represents an item in the workflow. It could be a task, a job, or a step in the workflow process.

 **getWorkflowData()**: This method retrieves the workflow data associated with the workItem

Session session = workflowSession.adaptTo(Session.class);

You need a access to work jcr for that we using this line .we can not able use resource resolver in workflows .

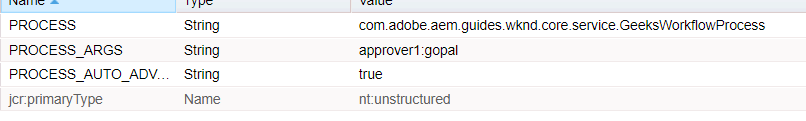
String path = workflowData.getPayload().toString() + "/jcr:content";

***/content/dq-aem/us/en/dummy-page/jcr:content—logs data***

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

It converts the Resource type to Node type because we don’t have Resource Resolver in workflows.

String[] processArgs = processArguments.get("PROCESS\_ARGS", "string").toString().split(",");



MetaDataMap wfd = workItem.getWorkflow().getWorkflowData().getMetaDataMap();

**workItem**: This is likely an object representing a unit of work within a workflow.

**getWorkflow()**: This method retrieves the workflow associated with the workItem.

**getWorkflowData()**: This method retrieves the data associated with the workflow.

**getMetaDataMap()**: This method retrieves a metadata map, which is likely a collection of key-value pairs related to the workflow data.