**1. Create Custom Workflow (my custom workflow).**

Go to Tools > Workflow > Models.  
Click on "Create".  
Enter the title: my custom workflow.  
Click on "Create & Open".  
Add a Process Step to the workflow model.  
Configure the Process Step:

* Title: Custom Process Step
* Process: CustomWorkflowProcess (created in the next step)  
  Save the workflow.

**2. Create a custom workflow process and print the page title in the logs and run this workflow on the page so that it can provide some metadata in the logs.**

Path: /apps/example/core/src/main/java/com/example/core/workflow

**CustomWorkflowProcess.java**

package com.example.core.workflow;

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;

import org.osgi.service.component.annotations.Component;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

public class CustomWorkflowProcess implements WorkflowProcess {

private static final Logger LOG = LoggerFactory.getLogger(CustomWorkflowProcess.class);

@Override

public void execute(WorkItem workItem, WorkflowSession workflowSession, MetaDataMap metaDataMap) {

String pagePath = workItem.getWorkflowData().getPayload().toString();

LOG.info("Executing Custom Workflow on page: {}", pagePath);

MetaDataMap metaData = workItem.getWorkflowData().getMetaDataMap();

metaData.put("processedBy", "Custom Workflow");

LOG.info("Metadata added: processedBy=Custom Workflow");

}

}

**3. Create an Event handler in aem and print the resource path in the logs.**

Path: /apps/example/core/src/main/java/com/example/core/event

**CustomEventHandler.java**

package com.example.core.event;

import org.osgi.service.component.annotations.Component;

import org.osgi.service.event.Event;

import org.osgi.service.event.EventHandler;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Component(service = EventHandler.class, property = {"event.topics=org/apache/sling/api/resource/Resource/ADDED"})

public class CustomEventHandler implements EventHandler {

private static final Logger LOG = LoggerFactory.getLogger(CustomEventHandler.class);

@Override

public void handleEvent(Event event) {

String resourcePath = (String) event.getProperty("path");

LOG.info("Resource added at path: {}", resourcePath);

}

}

**4. Create sling job to print hello world messages in logs**

Path: /apps/example/core/src/main/java/com/example/core/jobs

**HelloWorldJob.java**

package com.example.core.jobs;

import org.apache.sling.event.jobs.consumer.JobConsumer;

import org.osgi.service.component.annotations.Component;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Component(service = JobConsumer.class, property = JobConsumer.PROPERTY\_TOPICS + "=example/helloWorldJob")

public class HelloWorldJob implements JobConsumer {

private static final Logger LOG = LoggerFactory.getLogger(HelloWorldJob.class);

@Override

public JobResult process(Job job) {

LOG.info("Hello World from Sling Job!");

return JobResult.OK;

}

}

**5. Create a schedule to print the yellow world in logs every 5 mins through custom configuration using cron expression.**

Path: /apps/example/core/src/main/java/com/example/core/schedulers

**YellowWorldScheduler.java**

package com.example.core.schedulers;

import org.osgi.service.component.annotations.Component;

import org.osgi.service.component.annotations.Activate;

import org.osgi.service.component.annotations.Modified;

import org.osgi.service.metatype.annotations.Designate;

import org.osgi.service.metatype.annotations.ObjectClassDefinition;

import org.osgi.service.metatype.annotations.AttributeDefinition;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Component(service = Runnable.class, property = {

"scheduler.expression=0 0/5 \* 1/1 \* ? \*", // Every 5 mins

"scheduler.concurrent=false"

})

@Designate(ocd = YellowWorldScheduler.Config.class)

public class YellowWorldScheduler implements Runnable {

private static final Logger LOG = LoggerFactory.getLogger(YellowWorldScheduler.class);

@ObjectClassDefinition(name = "Yellow World Scheduler Config")

public @interface Config {

@AttributeDefinition(name = "Scheduler expression")

String expression() default "0 0/5 \* 1/1 \* ? \*";

}

@Activate

@Modified

protected void activate(Config config) {

LOG.info("Yellow World Scheduler activated.");

}

@Override

public void run() {

LOG.info("Yellow World");

}

}

**6. Create 3 users and add them to a group (Dev author create this new group) and give permission to read only for /content and /dam folder, and they should have replication access as well.**

Go to AEM > Tools > Security > Groups.  
Click on "Create" and name the group "DevAuthor".

Go to AEM > Tools > Security > Users.  
Create 3 users:

* User 1: author1
* User 2: author2
* User 3: author3  
  Add all 3 users to the "DevAuthor" group.

Go to CRXDE:  
Path: /content

* Add "read" permission for "DevAuthor".

Path: /content/dam

* Add "read" permission for "DevAuthor".

Go to AEM > Tools > Deployment > Replication > Agents on Author

* Select the default replication agent.
* Add the "DevAuthor" group under the "Allowed Principals".
* Save the configuration.