Date: 25/03/2025

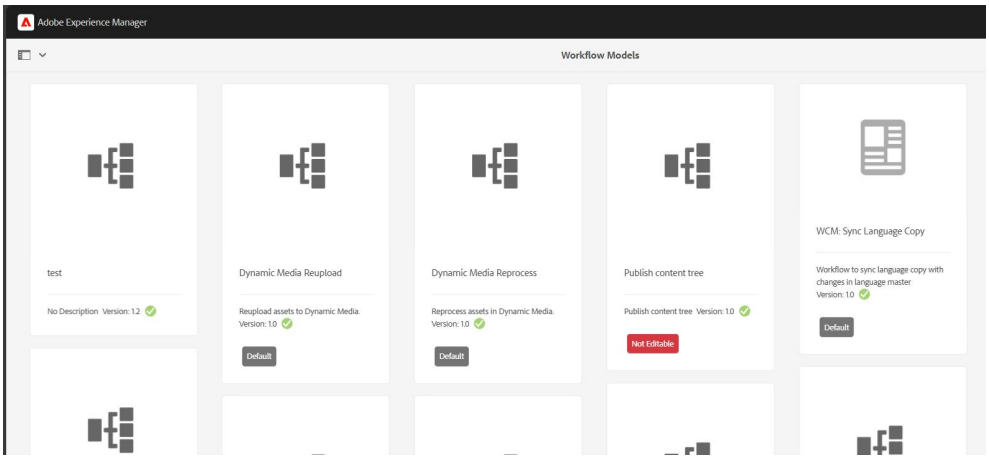
Tasks

1. Create Custom Workflow Model
2. Create Custom Workflow Process
3. Create Event Handler
4. Create Sling Job
5. Create Scheduler
6. Create Users & Group with Permissions
7. Test and Troubleshoot Custom Workflow, Event Handler, Sling Job, and Scheduler
8. Best Practices for Workflow and Event Handling in AEM

1. Create Custom Workflow Model

Steps:

1. **Navigate to AEM Workflow Console:** Go to **Tools > Workflow > Models**.
2. **Create a New Workflow Model:** Click on **Create** and provide a name and description.
3. **Add Workflow Steps:** Drag and drop process steps from the component list.
4. **Configure Each Step:** Define transitions, handlers, and conditions.
5. **Save & Activate the Workflow Model:** Ensure it is properly configured.



2. Create Custom Workflow Process

Steps:

1. **Create a Java Class:** Implement WorkflowProcess interface.
2. **Implement Process Logic:** Add business logic inside execute method.
3. **Register as OSGi Component:** Annotate the class with @Component and @Service.
4. **Deploy & Test Workflow Process:** Deploy the code and test in AEM.

Example Code:

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

public class CustomWorkflowProcess implements WorkflowProcess {

   @Override

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

      String payload = (String) workItem.getWorkflowData().getPayload();

      System.out.println("Processing workflow for payload: " + payload);

   }

}
```

3. Create Event Handler

Steps:

1. **Create an OSGi Event Listener:** Implement EventHandler interface.
2. **Subscribe to AEM Events:** Define event topics in @Component annotation.
3. **Handle Event Logic:** Implement business logic in handleEvent method.
4. **Deploy and Validate Events:** Deploy the listener and verify functionality.

Example Code:

```
package com.myTraining.core;

import org.apache.sling.api.resource.ResourceResolverFactory;
import org.apache.sling.api.resource.observation.ResourceChange;
import org.apache.sling.api.resource.observation.ResourceChangeListener;
import org.osgi.service.component.annotations.Component;
```

```
import org.osgi.service.component.annotations.Reference;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component(
        service = ResourceChangeListener.class,
        property = {
                ResourceChangeListener.PATHS + "=/content/myTraining",
                ResourceChangeListener.CHANGES + "=ADDED",
ResourceChangeListener.CHANGES + "=CHANGED",
ResourceChangeListener.CHANGES + "=REMOVED"                    }
)
public class MyEventHandler implements ResourceChangeListener {
    private static final Logger LOGGER = LoggerFactory.getLogger(MyEventHandler.class);

    @Reference
    private ResourceResolverFactory resourceResolverFactory;

    @Override
    public void onChange(java.util.List<ResourceChange> changes) {
        // Look at each change one by one
        for (ResourceChange change : changes) {
            String path = change.getPath();

            LOGGER.info("Something happened! The path is: {}", path);
        }
    }
}
```

4. Create Sling Job

Steps:

1. **Create an OSGi Component:** Implement JobConsumer interface.
2. **Define Job Topic:** Use @Component annotation to define the job name.
3. **Implement Job Execution Logic:** Write processing logic in process method.
4. **Deploy & Trigger Job:** Deploy and test by scheduling a job manually.

Example Code:

```
@Component(service = JobConsumer.class, property = {JobConsumer.PROPERTY_TOPICS
+ "=custom/job"})

public class CustomSlingJob implements JobConsumer {

    @Override
```

```
    public JobResult process(Job job) {
        System.out.println("Executing custom job: " + job.getTopic());
        return JobResult.OK;
    }
}
```

5. Create Scheduler

Steps:

1. **Create an OSGi Component:** Implement Runnable interface.
2. **Configure Scheduler Properties:** Use @Designate annotation for scheduling intervals.
3. **Implement Execution Logic:** Define the logic inside the run method.
4. **Deploy & Verify Execution:** Deploy and check logs to ensure it runs at expected intervals.

Example Code:

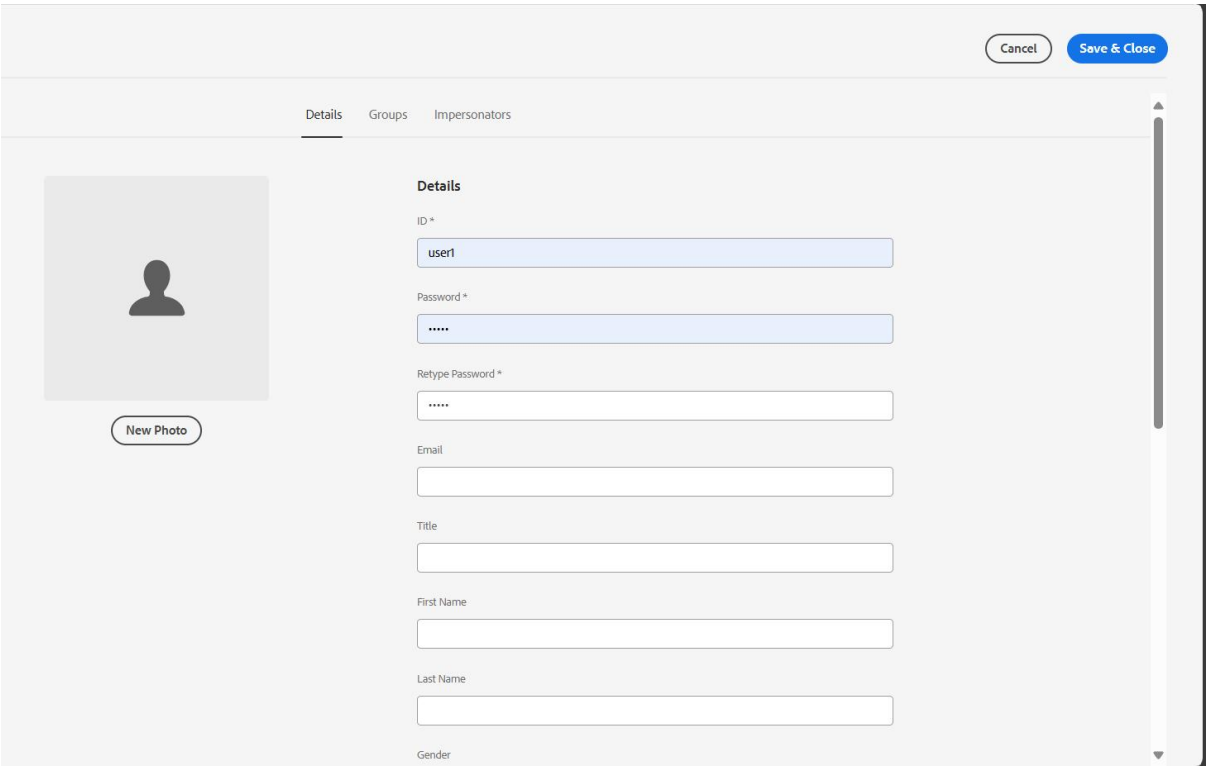
```
package com.myTraining.core.schedulers;

import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.metatype.annotations.AttributeDefinition;
import org.osgi.service.metatype.annotations.Designate;
import org.osgi.service.metatype.annotations.ObjectClassDefinition;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Designate(ocd=SimpleScheduledTask.Config.class)
@Component(service=Runnable.class)
public class   SimpleScheduledTask implements Runnable {

    @ObjectClassDefinition(name="A scheduled task",
                    description = "Simple demo for cron-job like task with properties")
    public static @interface Config {

        @AttributeDefinition(name = "Cron-job expression")
        String scheduler_expression() default "*/15 * * * * ?";
```

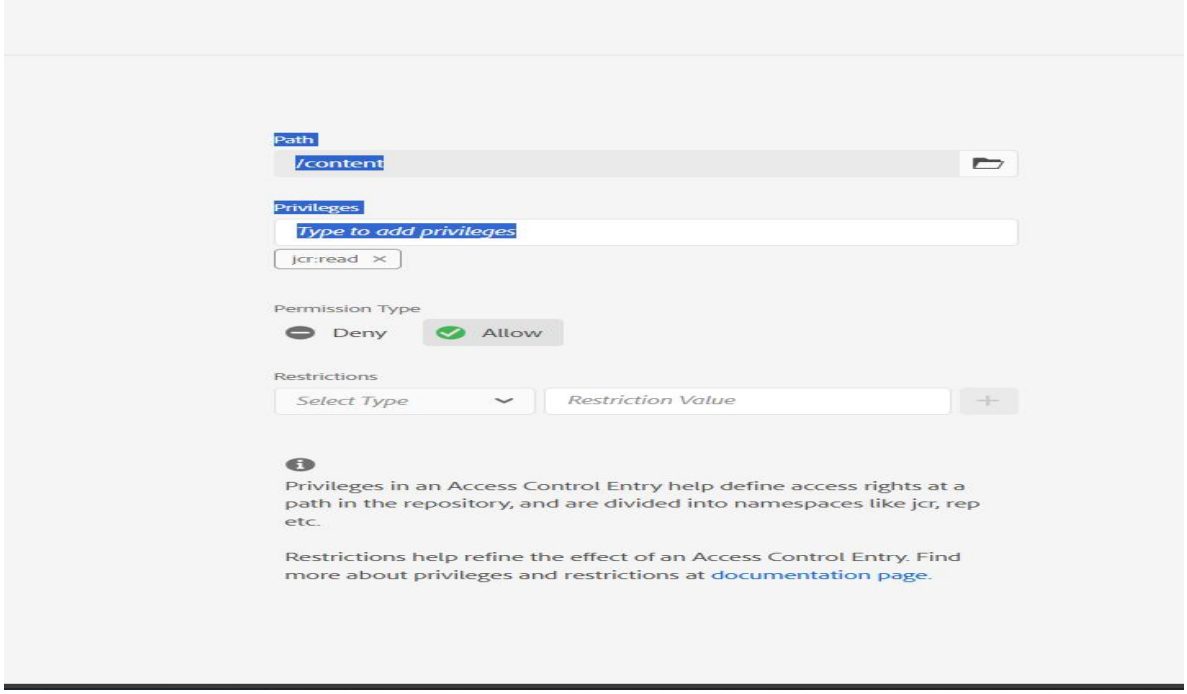
```
    @AttributeDefinition(name = "Concurrent task",
                    description = "Whether or not to schedule this task concurrently")
    boolean scheduler_concurrent() default false;

    @AttributeDefinition(name = "A parameter",
                    description = "Can be configured in /system/console/configMgr")
    String myParameter() default "";
  }

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

  private String myParameter;

  @Override
  public void run() {
    logger.debug("SimpleScheduledTask is now running, myParameter='{}'", myParameter);
  }

  @Activate
  protected void activate(final Config config) {
    myParameter = config.myParameter();
  }

}
```

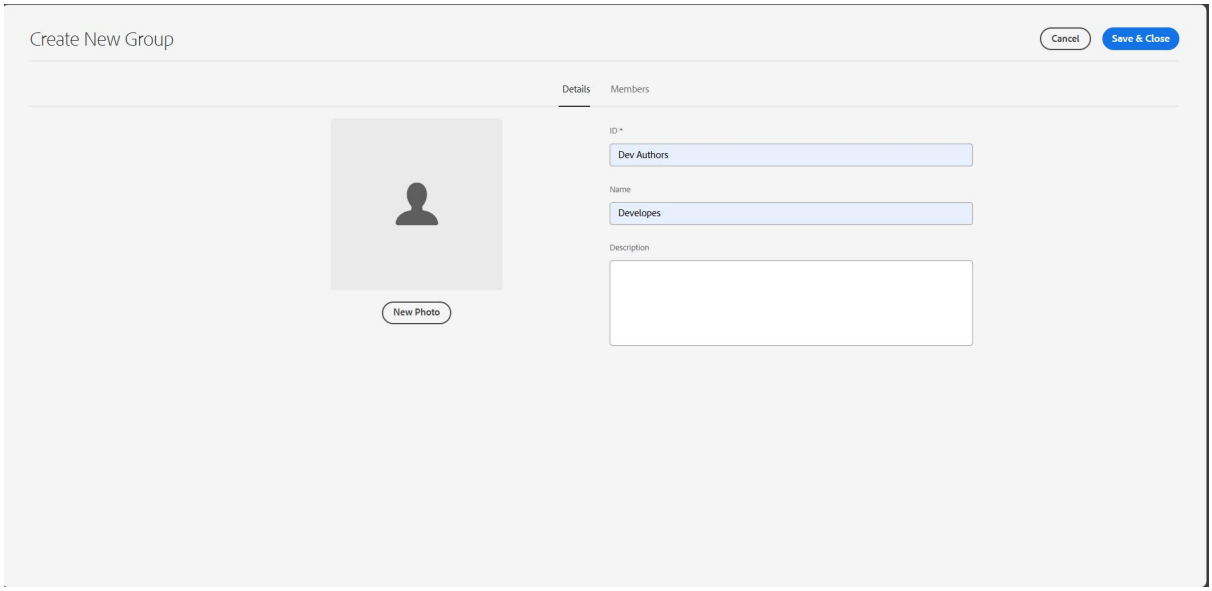
6. Create Users & Group with Permissions

Steps:

1. **Navigate to AEM Users Console: Go to Tools > Security > Users.**
2. **Create a New User: Provide username, password, and required details.**
3. **Create a New Group: Assign appropriate permissions.**
4. **Add Users to Group: Manage user roles efficiently.**
5. **Verify Access Control: Test by logging in with the created user.**


Cancel
Save & Close
Details
Groups
Impersonators
New Photo
Details
ID *
user1
Password *
Retype Password *
Email
Title
First Name
Last Name
Gender


Path
/content
Privileges
Type to add privileges
jcr:read
Permission Type
Deny
Allow
Restrictions
Select Type
Restriction Value
Privileges in an Access Control Entry help define access rights at a path in the repository, and are divided into namespaces like jcr, rep etc.
Restrictions help refine the effect of an Access Control Entry. Find more about privileges and restrictions at documentation page.



7. Test and Troubleshoot Custom Workflow, Event Handler, Sling Job, and Scheduler

Testing Steps:

1. **Trigger Workflow Manually: Use AEM UI or programmatic methods.**
2. **Monitor Logs: Use tail -f error.log to check logs in AEM.**
3. **Debug Errors: Modify and redeploy components as needed.**
4. **Validate Expected Behavior: Ensure the implemented logic is working correctly.**

8. Best Practices for Workflow and Event Handling in AEM

Guidelines:

- **Use Asynchronous Processing: Prefer Sling Jobs for long-running tasks.**
- **Optimize Workflow Execution: Keep workflow steps minimal and efficient.**
- **Secure Event Handlers: Validate and filter events before processing.**
- **Monitor Performance: Use AEM monitoring tools to detect bottlenecks.**
- **Follow AEM Coding Standards: Maintain best practices in Java and OSGi components.**