**Task-10 CODE**

**OSGI CONFIGURATION**

**package** com.adobe.aem.guides.demo.core.schedulers;

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

**import** org.osgi.service.metatype.annotations.AttributeType;

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

@ObjectClassDefinition(name = "my scheduler",description = "scheduler is ccreated")

**public** **@interface** schedulerpractisemrng {

@AttributeDefinition(

name="service name",

type = AttributeType.***STRING***,

description = "enter the service name"

)

**public** String getservice\_name() **default** "practise";

@AttributeDefinition

(

name="can run concurrently",

type=AttributeType.***BOOLEAN***,

description="can run concurrently"

)

**public** **boolean** canrunconcurrently() **default** **false**;

@AttributeDefinition

(

name="Enabled scheduler",

type=AttributeType.***BOOLEAN***,

description="Enable the scheduler"

)

**public** **boolean** Enabledscheduler() **default** **true**;

@AttributeDefinition(

name="Expression",

type = AttributeType.***STRING***,

description = "enter the Expression"

)

**public** String getExpressi(); }

***SCHEDULERS CODE FOR CORN EXPRESSION***

**package** com.adobe.aem.guides.demo.core.schedulers;

**import** java.util.HashMap;

**import** java.util.Map;

**import** javax.jcr.Session;

**import** org.apache.sling.api.resource.LoginException;

**import** org.apache.sling.api.resource.ResourceResolver;

**import** org.apache.sling.api.resource.ResourceResolverFactory;

**import** org.apache.sling.commons.scheduler.ScheduleOptions;

**import** org.apache.sling.commons.scheduler.Scheduler;

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

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

**import** org.osgi.service.component.annotations.Deactivate;

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

**import** org.osgi.service.component.annotations.Reference;

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

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** com.day.cq.replication.ReplicationActionType;

**import** com.day.cq.replication.Replicator;

**import** com.day.cq.wcm.api.Page;

**import** com.day.cq.wcm.api.PageManager;

@Designate(ocd = schedulerpractisemrng.**class**)

@Component(service = Runnable.**class**, immediate = **true**)

**public** **class** schedulermrnng **implements** Runnable {

**private** **static** **final** Logger ***log*** = LoggerFactory.*getLogger*(schedulermrnng.**class**);

@Reference

**private** Scheduler scheduler;

@Reference

**private** ResourceResolverFactory resourceResolverFactory;

@Reference

**private** Replicator replicator;

**private** **static** **final** String ***SERVICE\_USER*** = "hemanth";

**private** String cronExpression;

@Modified

**public** **void** modify(schedulerpractisemrng sch) {

**this**.cronExpression = sch.getExpressi();

addscheduler(sch); // Re-schedule with the new cron expression

}

@Activate

**public** **void** activation(schedulerpractisemrng sch) {

**this**.cronExpression = sch.getExpressi();

// addscheduler(sch);

}

**public** **void** addscheduler(schedulerpractisemrng sch) {

***log***.info("Scheduler is created");

**if** (sch.Enabledscheduler()) {

ScheduleOptions scheduleOptions = scheduler.EXPR(cronExpression);

scheduleOptions.canRunConcurrently(sch.canrunconcurrently());

scheduleOptions.name(sch.getservice\_name());

scheduler.schedule(**this**, scheduleOptions);

}

}

@Deactivate

**public** **void** dectivate(schedulerpractisemrng sch) {

removescheduler(sch);

}

**public** **void** removescheduler(schedulerpractisemrng sch) {

***log***.info("Job is unscheduled");

scheduler.unschedule(sch.getservice\_name());

}

@Override

**public** **void** run() {

***log***.info("Scheduler is running");

***log***.info("My cron expression: " + cronExpression);

publishPages("/content/Demo/us/en");

}

**PUBLISH PAGES AUTOMATICALLY BY CORN EXP**

**private** **void** publishPages(String path) {

**try** (ResourceResolver resourceResolver = getServiceResourceResolver()) {

PageManager pageManager = resourceResolver.adaptTo(PageManager.**class**);

**if** (pageManager != **null**) {

Page page = pageManager.getPage(path);

**if** (page != **null**) {

page.listChildren().forEachRemaining(childPage -> {

**try** {

***log***.info("Publishing page: {}", childPage.getPath());

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

replicator.replicate(session, ReplicationActionType.***ACTIVATE***, childPage.getPath());

} **catch** (Exception e) {

***log***.error("Error while publishing page: {}", childPage.getPath(), e);

}

});

} **else** {

***log***.warn("No page found at path: {}", path);

}

}

} **catch** (LoginException e) {

***log***.error("Error obtaining service resource resolver", e);

}

}

**private** ResourceResolver getServiceResourceResolver() **throws** LoginException {

Map<String, Object> param = **new** HashMap<>();

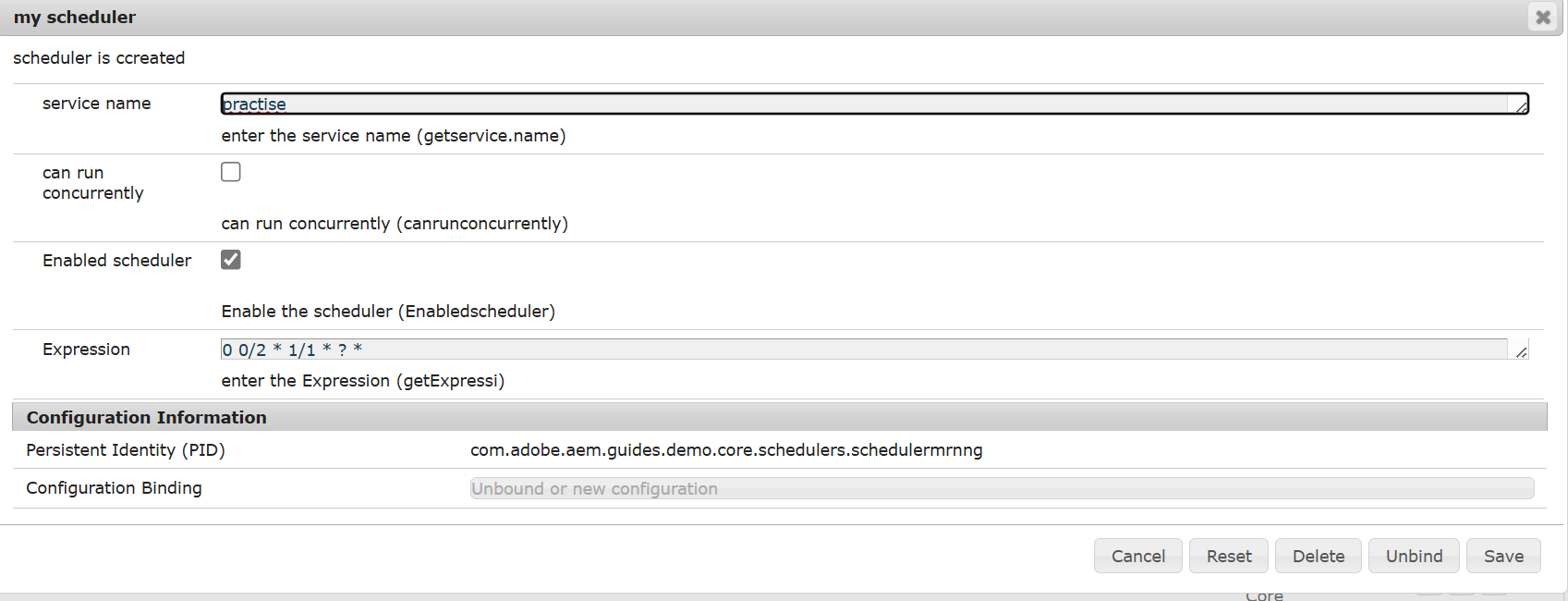
param.put(ResourceResolverFactory.***SUBSERVICE***, ***SERVICE\_USER***);

**return** resourceResolverFactory.getServiceResourceResolver(param);

}

}

**OSGI CONFIGURATION**



**LOG MESSAGE FOR PAGE PUBLISHED**

