

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe. Adobe assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, the Creative Cloud logo, and the Adobe Marketing Cloud logo are either registered trademarks or trademarks of Adobe in the United States and/or other countries.

All other trademarks are the property of their respective owners.

Adobe, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contents

Event Handling in AEM	3
Scheduled Jobs	4
Exercise 1: Schedule a Sling job	5
Task 1: Create a Sling Scheduler	5
Task 2: Create JSON configuration files	9
Task 3: Install via command line	10
Task 4: Verify configurations	10
Working with Sling jobs	11
Exercise 2: Consume Sling job	13
Task 1: Create a Sling Job Consumer	13
Task 2: Install via command line	17
Task 3: Test the service	18
Sling Resource Change Listening	19
Exercise 3: Create a resource listener	20
Task 1: Create a Sling Resource Listener	20
Task 2: Install via command line	23
Task 3: Test the service	23
References	25

Event Handling in AEM

Introduction

With Adobe Experience Manager (AEM), there are different approaches that can be taken regarding event management. Depending on the complexity, the nature of the payload or intensity of process, and inherent requirements or business roles involved, we can consider the following techniques:

- Events at the OSGi container level, with the OSGi Event Admin
- Events at the Sling level, with Sling Jobs (that you can schedule) and the ResourceChangeListener
- Events at the level of AEM, with different high-level objects (for example, workflow process steps, polling importers, replication actions, notifications, or auditing - all based on Sling Jobs)
- Events at the repository level, with JCR observation

Objectives

After completing this course, you will be able to:

- Describe Sling scheduler
- Create a Job consumer
- Discuss event modeling
- Create a resource listener

Scheduled Jobs

Scheduled Jobs are put in the queue at a specific time (optionally periodically). For that the `ScheduleBuilder` must be used which is retrieved via `JobBuilder.schedule()`.

An example code for scheduling a job looks like this:

```
import org.apache.sling.jobs.JobManager;
import org.apache.sling.event.jobs.JobBuilder.ScheduleBuilder;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;

@Component
public class MyComponent {

    @Reference
    private JobManager jobManager;

    public void startScheduledJob() {
        ScheduleBuilder scheduleBuilder =
            jobManager.createJob("my/special/jobtopic").schedule();
        scheduleBuilder.daily(0,0); // execute daily at midnight
        if (scheduleBuilder.add() == null) {
            // something went wrong here, use
            scheduleBuilder.add(List<String>) instead to get further information about
            the error
        }
    }
}
```

Internally the scheduled Jobs use the Commons Scheduler Service. But in addition they are persisted (by default below `/var/eventing/scheduled-jobs`) and survive therefore even server restarts. When the scheduled time is reached, the job is automatically added as regular Sling Job through the `JobManager`.

Exercise 1: Schedule a Sling job

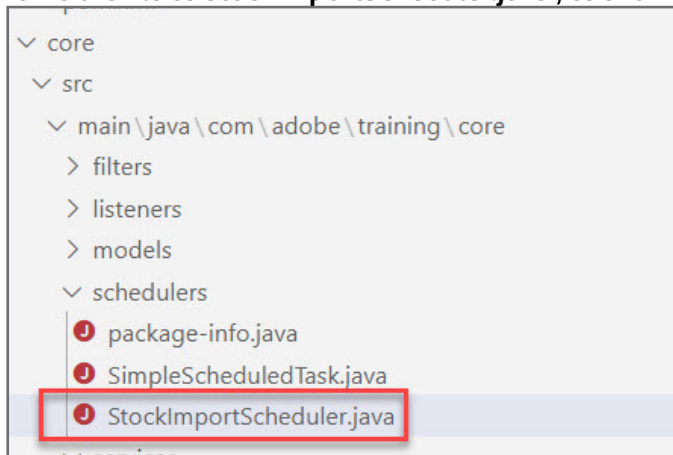
In this exercise, you will create an OSGi component that creates a new scheduled Sling Job based on the OSGi configs provided. This OSGi component is also a config factory implying there can be many instances. Note that this component only adds scheduled jobs to the queue and does not actually process the Jobs.

This exercise includes the following tasks:

1. Create a Sling Scheduler
2. Create JSON configuration files
3. Install via command line
4. Verify configurations

Task 1: Create a Sling Scheduler

1. Open the IDE containing your Maven project for this course if not already opened.
2. In the IDE navigation on the left, find the **core** module.
3. Navigate to **core > src > main/java/com/adobe/training/core**.
4. Right-click the folder **schedulers** and choose **New File**.
5. Name the file as **StockImportScheduler.java**, as shown:



- To add code to your **StockImportScheduler.java**, open up the **Exercise_Files-EC** folder for this course and navigate to **/core/src/main/java/com/adobe/training/core/schedulers/**.
- Open **StockImportScheduler.java** and copy the contents.
- In your IDE, paste the contents into your Maven Project > **StockImportScheduler.java**.
- Double-click **StockImportScheduler.java**. The file opens in the editor, as shown:

```
package com.adobe.training.core.schedulers;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

//org.apache.sling.event.* api available in Sling 9 documentation:
import org.apache.sling.event.jobs.JobBuilder;
import org.apache.sling.event.jobs.JobBuilder.ScheduleBuilder;
import org.apache.sling.event.jobs.JobManager;
import org.apache.sling.event.jobs.ScheduledJobInfo;

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

/**
 * This class adds a Sling Job to the job queue so that a job consumer can process
 * work. Sling Jobs are guaranteed to be proceed and the scheduler can be configured
 * based on an OSGi config node.
 */
@Component(immediate = true,
           configurationPid = "com.adobe.training.core.schedulers.StockImportScheduler",
           configurationPolicy = ConfigurationPolicy.REQUIRE)
@Designate(ocd = StockImportScheduler.StockImportConfiguration.class, factory=true)
public class StockImportScheduler {
    public static final String JOB_TOPIC_STOCKIMPORT = "com/adobe/training/core/jobs/stockimportjob";
    public static final String JOB_PROP_SYMBOL = "symbol";
    public static final String JOB_PROP_URL = "url";
    public static final String DEFAULT_IMPORT_URL = "https://raw.githubusercontent.com/Adobe-Marketing-Cloud/ADLS-Samples/master/stock-data/";
}
```

```

private final Logger logger = LoggerFactory.getLogger(getClass());
// Convenience string to find the log messages for this training example class
// Logs can be found in crx-quickstart/logs/error.log
private String searchableLogStr = "*****";

@ObjectClassDefinition(name = "Training Stock Importer")
public @interface StockImportConfiguration {
    @AttributeDefinition(
        name = "Stock Symbol",
        description = "Characters representing the stock to be imported",
        type = AttributeType.STRING
    )
    public String symbol() default "";

    @AttributeDefinition(
        name = "Expression",
        description = "Run every so often as defined in the cron-job expression.",
        type = AttributeType.STRING
    )
    public String cronExpression() default "0 0/2 * * * ?";

    @AttributeDefinition(
        name = "Stock URL",
        description = "URL to request the stock data to be imported",
        type = AttributeType.STRING
    )
    public String stock_url() default DEFAULT_IMPORT_URL;
}

@Reference
private JobManager jobManager;

private int schedulerID;

private JobBuilder jobBuilder;
private ScheduleBuilder scheduleBuilder;
private ScheduledJobInfo theScheduledJob;

@Activate @Modified
protected void activate(StockImportConfiguration config) {
schedulerID;    logger.info(searchableLogStr + "StockImport ScheduledJob '{}' with ID: '{}' Activated", config.symbol(),
                schedulerID = config.symbol().hashCode();
                startScheduledJob(config);
    }

@Modified
protected void modified(StockImportConfiguration config) {
    removeScheduler(config);
    schedulerID = config.symbol().hashCode() + 1; //updates schedulerID
    startScheduledJob(config);
}

@Deactivate
protected void deactivate(StockImportConfiguration config) {
    removeScheduler(config);
}

private void startScheduledJob(StockImportConfiguration config){
    jobBuilder = jobManager.createJob(StockImportScheduler.JOB_TOPIC_STOCKIMPORT);
    // Create a properties map that contains the configurations we want to pass to the job
    HashMap<String, Object> jobProps = new HashMap<>();
    jobProps.put(JOB_PROP_SYMBOL, config.symbol());
}

```

```

        jobProps.put(JOB_PROP_URL, config.stock_url());

        jobBuilder.properties(jobProps);
        scheduleBuilder = jobBuilder.schedule();
        scheduleBuilder.cron(config.cronExpression());
        theScheduledJob = scheduleBuilder.add();
        if(theScheduledJob == null){
            List<String> errors = new ArrayList<>();
            scheduleBuilder.add(errors);
        } else {
            " " logger.info(searchableLogStr + "ScheduledJob added to the Queue. Topic: " + theScheduledJob.getJobTopic() +
                + "Properties: " + theScheduledJob.getJobProperties().toString() + " "
                + "Next Execution: " + theScheduledJob.getNextScheduledExecution().toString());
        }
    }

    private void removeScheduler(StackImportConfiguration config) {
        if(theScheduledJob != null) {
            schedulerID; logger.info(searchableLogStr + "Removing '{}' ScheduledJob, with ID: '{}', config.symbol(),
                theScheduledJob.unschedule();
        }
    }
}

```

10. Open a command prompt to the location of your Maven project. For example: **C:/adobe/<myproject >**



Note: If you are using an IDE with an integrated terminal such as Visual Studio Code, you can run your Maven commands there instead since it is already open to your project.

11. In the command prompt run the command:

```
$ mvn clean install -Padobe-public -PautoInstallSinglePackage
```

Your project has now successfully installed into your local AEM Server

12. Go to <http://localhost:4502/system/console/configMgr> and search for **Training Stock Importer**.
13. Click on **Training Stock Importer**. The **Training Stock Importer** window opens, as shown:

Configuration Information	
Persistent Identity (PID)	[Temporary PID replaced by real PID upon save]
Factory Persistent Identifier (Factory PID)	com.adobe.training.core.schedulers.StockImportsScheduler
Configuration Binding	Unbound or new configuration

14. Notice how you now have the ability to add a Stock Symbol and then a cron expression. In order to set these values, you need to create some configuration files.

Task 2: Create JSON configuration files

1. In your IDE, navigate to **ui.config > src/main/content/jcr_root > apps > training > osgiconfig**.
2. Right-click the folder **config** and choose **New file**.
3. Enter the name as **com.adobe.training.core.schedulers.StockImportScheduler~adbe.cfg.json**.



Note: Because the StockImportScheduler has a factory configuration, you can have many configurations for it. The ~adbe is a unique identifier for this specific configuration.

4. Double-click **com.adobe.training.core.schedulers.StockImportScheduler~adbe.cfg.json** to open it in the editor.
5. To add the json properties to your file, open up the **Exercise_Files-EC** folder for this course and navigate to **ui.config/src/main/content/jcr_root/apps/training/osgiconfig/config**.
6. Open the file **com.adobe.training.core.schedulers.StockImportScheduler~adbe.cfg.json** using a text editor. Copy the contents and paste it to the file **com.adobe.training.core.schedulers.StockImportScheduler~adbe.cfg.json** in your IDE, as shown:

```
ui.config > src > main > content > jcr_root > apps > training > osgiconfig > config > com.adobe.training.core.schedulers.StockImportScheduler~adbe.json > ...  
1 {  
2   "symbol": "ADBE",  
3   "stock.url": "https://raw.githubusercontent.com/Adobe-Marketing-Cloud/ADLS-Samples/master/stock-data/",  
4   "cronExpression": "0 0/2 * * * ?"  
5 }
```

7. Select **File > Save** from the menu to save the file.

To schedule another stock job, you need to create another json configuration file:

8. In your IDE, navigate to **ui.config > src/main/content/jcr_root > apps > training > osgiconfig**.
9. Right-click the folder **config** and choose **New file**.
10. Enter the name as **com.adobe.training.core.schedulers.StockImportScheduler~msft.cfg.json**.
11. Double-click **com.adobe.training.core.schedulers.StockImportScheduler~msft.cfg.json** to open it in the editor.
12. To add the json properties to your file, open up the **Exercise_Files-EC** folder for this course and navigate to **ui.config/src/main/content/jcr_root/apps/training/osgiconfig/config**.
13. Open the file **com.adobe.training.core.schedulers.StockImportScheduler~msft.cfg.json** using a text editor. Copy the contents and paste it to the file **com.adobe.training.core.schedulers.StockImportScheduler~msft.cfg.json** in your IDE, as shown:

```
ui.config > src > main > content > jcr_root > apps > training > osgiconfig > config > com.adobe.training.core.schedulers.StockImportScheduler~msft.cfg.json > ...  
1 {  
2   "symbol": "MSFT",  
3   "stock.url": "https://raw.githubusercontent.com/Adobe-Marketing-Cloud/ADLS-Samples/master/stock-data/",  
4   "cronExpression": "0 0/2 * * * ?"  
5 }
```

14. Select **File > Save** from the menu to save the file.

Task 3: Install via command line

1. Open a command prompt to the location of your Maven project. For example: **C:/adobe/<myproject >**



Note: If you are using an IDE with an integrated terminal such as Visual Studio Code, you can run your Maven commands there instead since it is already open to your project.

2. In the command prompt run the command:

```
$ mvn clean install -Padobe-public -PautoInstallSinglePackage
```

Your project has now successfully installed into your local AEM Server

Task 4: Verify configurations

1. Navigate to the Web Console in AEM (<http://localhost:4502/system/console/configMgr>). The **Adobe Experience Manager Web Console** page opens
2. Go to **Sling > Jobs**.
3. Press Ctrl+F to open a “find” window. In the window, type these words to search: **StockImportScheduler**.
4. Observe the scheduled jobs, as shown:

Scheduled Jobs	
Schedule	Job Topic
1	com/adobe/training/core/jobs/stockimportjob
2	com/adobe/training/core/jobs/stockimportjob
No active queues.	

5. Go to the Web console > **Configurations**.
6. Press Ctrl+F to open a “find” window. In the window, type these words to search: **Training Stock Importer**.
7. Observe the **adbe** and **msft** configurations that are set because of the configuration files for the scheduler.

Training Stock Importer	
✓	com.adobe.training.core.schedulers.StockImportScheduler~adbe
✓	com.adobe.training.core.schedulers.StockImportScheduler~msft

Working with Sling jobs

Every time there is a need to process a given event only once and ensure the event is not missed, you can leverage the Sling Job processing mechanism. You can see it as if you wanted to print a document with a specific printer on a network: you do not care if the printer is already busy or even switched off, any printing job sent to it will be processed once the printer will be ready, and only that chosen printer will do it.

Contrary to OSGi events, Sling Jobs are guaranteed to be processed only once. On the other hand, they add some overhead, so they would be used only when really necessary (the other methods discussed in this module could be a better choice in some cases).

Working with Sling Jobs requires the following components:

- A job manager, used to add a job to a queue with data associated to that job
- A job consumer, to process that job and remove it from the queue

Because of the uniqueness of the process, once a job consumer has processed, it gets removed from the queue and no other consumer would be able to process it (which is a case we should avoid anyway). Similarly, if no job consumer is registered to process a given job topic, the job stays in the queue (which is again something that would be considered a deployment error).

Adding a job consists of calling the method **addJob()** of the **org.apache.sling.event.jobs.JobManager** service and passing to it a topic (a string identifying the job) and a **HashMap** to carry the job information. A component that needs to consume a job for processing must implement the **org.apache.sling.event.jobs.consumer.JobConsumer** interface and override the process method to get the **Job** object **org.apache.sling.event.jobs.Job**, as shown below:

```
import org.apache.sling.event.jobs.Job;
import org.apache.sling.event.jobs.consumer.JobConsumer;
@Component (service = JobConsumer.class,
            Property = JobConsumer.PROPERTY_TOPICS + "=com/adobe/training/core/myjobtopic")

public class MyTopicProcessor implements JobConsumer {
    @Override
    public JobResult process(final Job job) {
        // do something
    }
}
```

By convention, the topic name is built by using the bundle symbolic name for the prefix (with slashes instead of dots) and suffixed with a custom label identifying the topic.

Monitoring

To view all registered topics with their statistics, go to the Web Console under **Sling > Jobs** in the Web Console:

<http://localhost:4502/system/console/slingevent>



Note: This console only shows topics that were processed by a Job consumer. Therefore, if a job is added and never consumed, the corresponding topic will not be visible. For debugging purposes, you can find custom jobs in the repository under **/var/eventing/jobs**.

Exercise 2: Consume Sling job

In the previous exercise, you added a scheduled Sling Job with the JobManager. In this exercise, you will process the Sling Job that was put in the Job Queue. The Job consumer you will build imports data from an external API endpoint and then writes it into the JCR. For this example, we are using a dummy stock data endpoint located in git, though in realistic scenarios this method could be used for integrations and connecting other 3rd party sources to AEM.

This exercise includes the following tasks:

1. Create a Sling Job Consumer
2. Install via command line
3. Test the service

Task 1: Create a Sling Job Consumer

1. In your IDE, navigate to **core > src > main/java/com/adobe/training/core**.
2. Right-click the folder **core** and choose **New File**.
3. Rename the file as **StockDataWriterJob.java**.
4. Double-click **StockDataWriterJob.java** to edit the file.
5. Copy the contents from **Exercise_Files-EC** under **/core/src/main/java/com/adobe/training/**
6. **core/ StockDataWriterJob.java** to **StockDataWriterJob.java** in your IDE, as shown:

```
package com.adobe.training.core;
```

```
import java.io.IOException;
```

```
import java.io.InputStream;
```

```
import java.io.InputStreamReader;
```

```
import java.net.SocketTimeoutException;
```

```
import java.net.URL;
```

```
import java.time.Instant;
```

```
import java.time.LocalDateTime;
```

```
import java.time.ZonedDateTime;
```

```
import java.time.ZonedDateTime;
```

```
import java.time.format.DateTimeFormatter;
```

```
import java.util.HashMap;
```

```
import java.util.Map;
```

```
import javax.jcr.RepositoryException;
```

```
import javax.net.ssl.HttpURLConnection;
```

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

```
import org.apache.sling.api.resource.ModifiableValueMap;
```

```

import org.apache.sling.api.resource.PersistenceException;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.api.resource.ResourceResolverFactory;
import org.apache.sling.api.resource.ResourceUtil;
import org.apache.sling.event.jobs.Job;
import org.apache.sling.event.jobs.consumer.JobConsumer;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Reference;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.day.cq.commons.jcr.JcrConstants;
import com.fasterxml.jackson.core.JsonFactory;
import com.fasterxml.jackson.core.JsonParseException;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.adobe.training.core.schedulers.StockImportScheduler;
/**
 * This job consumer takes in a data source url and stock symbol
 * and creates the node structure below.
 *
 * /content/stocks/
 * + <STOCK_SYMBOL> [sling:OrderedFolder]
 * + trade [nt:unstructured]
 *   - companyName = <value>
 *   - sector = <value>
 *   - lastTrade = <value>
 *   - timeOfUpdate = <value>
 *   - dayOfLastUpdate = <value>
 *   - openPrice = <value>
 *   - rangeHigh = <value>
 *   - rangeLow = <value>
 *   - volume = <value>
 *   - upDownPrice = <value>
 *   - week52High = <value>
 *   - week52Low = <value>
 *   - ytdChange = <value>
 */
@Component(
    immediate = true,
    service = JobConsumer.class,
    property = {
        JobConsumer.PROPERTY_TOPICS + "=" + StockImportScheduler.JOB_TOPIC_STOCKIMPORT
    }
)

public class StockDataWriterJob implements JobConsumer {

    private final Logger logger = LoggerFactory.getLogger(getClass());
    // Convenience string to find the log messages for this training example class
    // Logs can be found in crx-quickstart/logs/error.log
    private String searchableLogStr = "&&&&";

    //Public values for stock data
    public static final String STOCK_IMPORT_FOLDER = "/content/stocks";
    public static final String COMPANY = "companyName";
    public static final String SECTOR = "sector";
    public static final String LASTTRADE = "lastTrade";
    public static final String UPDATETIME = "timeOfUpdate";
    public static final String DAYOFUPDATE = "dayOfLastUpdate";
    public static final String OPENPRICE = "openPrice";
    public static final String RANGEHIGH = "rangeHigh";

```

```

public static final String RANGELOW = "rangeLow";
public static final String VOLUME = "volume";
public static final String UPDOWN = "upDown";
public static final String WEEK52LOW = "week52Low";
public static final String WEEK52HIGH = "week52High";
public static final String YTDCHANGE = "ytdPercentageChange";

@Reference
private ResourceResolverFactory resourceResolverFactory;

/**
 * Method that runs on the desired schedule.
 * Request the data with the stock symbol and get the returned JSON
 * Write the JSON to the JCR
 */
@Override
public JobResult process(Job job) {

    //extract properties added to the Job in Scheduler:
    String symbol = job.getProperty(StockImportScheduler.JOB_PROP_SYMBOL).toString().toUpperCase();
    String stock_url = job.getProperty(StockImportScheduler.JOB_PROP_URL).toString();

    //https://raw.githubusercontent.com/Adobe-Marketing-Cloud/ADLS-Samples/master/stock-data/
    String stockUrl = stock_url + symbol + ".json";

    HttpURLConnection request = null;
    try {
        URL sourceUrl = new URL(stockUrl);
        request = (HttpURLConnection) sourceUrl.openConnection();
        request.setConnectTimeout(5000);
        request.setReadTimeout(10000);
        request.connect();

        // Convert data return to a JSON object
        ObjectMapper objMapper = new ObjectMapper();
        JsonFactory factory = new JsonFactory();
        JobResult jobResult = null;
        if(request != null) {
            //Create a JsonParser based on the stream from the request content
            try(JsonParser parser = factory.createParser(new InputStreamReader((InputStream) request.getContent()))){
                //Create a Map from the JsonParser
                Map<String, String> allQuoteData = objMapper.readValue(parser,
                    new TypeReference<Map<String, String>>(){});
                logger.info("Last trade for stock symbol {} was {}", symbol, allQuoteData.get("latestPrice"));
                //Use the map to write nodes and properties to the JCR
                jobResult = writeToRepository(symbol, allQuoteData);
            }
        }
        catch (RepositoryException e) {
            logger.error(searchableLogStr + "Cannot write stock info for " + symbol + " to the JCR: ", e);
            return JobConsumer.JobResult.FAILED;
        }
        catch (JsonParseException e) {
            logger.error(searchableLogStr + "Cannot parse stock info for " + symbol, e);
            return JobConsumer.JobResult.FAILED;
        }
        catch (IOException e) {
            logger.error(searchableLogStr + "IOException: ", e);
            return JobConsumer.JobResult.FAILED;
        }
    }
    return jobResult;
}
catch (SocketTimeoutException e) {
    logger.error(searchableLogStr + "Five Second Timeout occurred.");
    return JobConsumer.JobResult.FAILED;
}
}
catch (IOException e) {

```

```

        logger.error(searchableLogStr + "The stock symbol: " + symbol + " does not exist...");
        return JobConsumer.JobResult.FAILED;
    }
}

/**
 * Creates the stock data structure
 *
 * + <STOCK_SYMBOL> [sling:OrderedFolder]
 * + trade [nt:unstructured]
 *   - companyName = <value>
 *   - sector = <value>
 *   - lastTrade = <value>
 *   - timeOfUpdate = <value>
 *   - dayOfLastUpdate = <value>
 *   - openPrice = <value>
 *   - rangeHigh = <value>
 *   - rangeLow = <value>
 *   - volume = <value>
 *   - upDownPrice = <value>
 *   - week52High = <value>
 *   - week52Low = <value>
 *   - ytdChange = <value>
 * @return
 */
private JobResult writeToRepository(String stockSymbol, Map<String, String> quoteData) throws RepositoryException {

    logger.info(searchableLogStr + "Stock Symbol: " + stockSymbol);
    logger.info(searchableLogStr + "JsonObject to Write: " + quoteData.toString());

    //Get the service user (training-user) that belongs to the training.core:training subservice
    Map<String, Object> serviceParams = new HashMap<>();
    serviceParams.put(ResourceResolverFactory.SUBSERVICE, "training");

    try (ResourceResolver resourceResolver = resourceResolverFactory
        .getServiceResourceResolver(serviceParams)) {

        // Transform the time stamp into a readable format
        Zoned timeZone = Zoned.of("America/New_York");
        long latestUpdateTime = Long.parseLong(quoteData.get("latestUpdate"));
        LocalDateTime timePerLatestUpdate = LocalDateTime.ofInstant(Instant.ofEpochMilli(latestUpdateTime),
            timeZone);
        ZonedDateTime timeWithZone = ZonedDateTime.of(timePerLatestUpdate, timeZone);
        DateTimeFormatter timeFormatter = DateTimeFormatter.ofPattern("hh:mm a zz");
        //will store timeOfUpdate as: Hour:Minute AM/PM, TimeZone e.g. 11:34 AM, EDT
        String updateTimeOfDay = timeWithZone.format(timeFormatter);
        DateTimeFormatter dayFormatter = DateTimeFormatter.ofPattern("E MMMM d, yyyy");
        String dayOfUpdate = timeWithZone.format(dayFormatter);

        //Create variables in specific data type and put them into a map
        Double lastPrice = Double.parseDouble(quoteData.get("latestPrice"));
        Double open = Double.parseDouble(quoteData.get("open"));
        Double high = Double.parseDouble(quoteData.get("high"));
        Double low = Double.parseDouble(quoteData.get("low"));
        Long latestVolume = Long.parseLong(quoteData.get("latestVolume"));
        Double change = Double.parseDouble(quoteData.get("change"));
        Double week52High = Double.parseDouble(quoteData.get("week52High"));
        Double week52Low = Double.parseDouble(quoteData.get("week52Low"));
        Double ytdChange = Double.parseDouble(quoteData.get("ytdChange"));

        String stockPath = STOCK_IMPORT_FOLDER + "/" + stockSymbol;
        String tradePath = stockPath + "/trade";
        Resource trade = resourceResolver.getResource(tradePath);

        //Test if stock import folder exists, otherwise create it
    }
}

```



```

Resource stockFolder = ResourceUtil.getOrCreateResource(resourceResolver, stockPath, "", "", false);

if (trade == null) {
    // set jcr:primaryType to nt:unstructured when resource is created
    Map<String, Object> stockData = new HashMap<String, Object>() {
        private static final long serialVersionUID = 1L;
        {
            put(JcrConstants.JCR_PRIMARYTYPE, JcrConstants.NT_UNSTRUCTURED);
        }
    };

    trade = resourceResolver.create(stockFolder, "trade", stockData);
}

ModifiableValueMap stockData = trade.adaptTo(ModifiableValueMap.class);

stockData.put(COMPANY, quoteData.get("companyName"));
stockData.put(SECTOR, quoteData.get("sector"));
stockData.put(UPDATETIME, UpdateTimeOfDay);
stockData.put(DAYOFUPDATE, dayOfUpdate);
stockData.put(LASTTRADE, lastPrice);
stockData.put(OPENPRICE, open);
stockData.put(RANGEHIGH, high);
stockData.put(RANGELOW, low);
stockData.put(VOLUME, latestVolume);
stockData.put(UPDOWN, change);
stockData.put(WEEK52HIGH, week52High);
stockData.put(WEEK52LOW, week52Low);
stockData.put(YTDCHANGE, ytdChange);

logger.info(searchableLogStr + "Updated trade data for " + stockSymbol);

//Write data into the JCR
resourceResolver.commit();

} catch (LoginException | PersistenceException e) {
    logger.error(searchableLogStr + "Exception with writing resource: ", e);
    return JobConsumer.JobResult.FAILED;
}

return JobConsumer.JobResult.OK;
}
}

```

7. Save the changes.

Task 2: Install via command line

1. Open a command prompt to the location of your Maven project. For example: C:/adobe/<myproject>



Note: If you are using an IDE with an integrated terminal such as Visual Studio Code, you can run your Maven commands there instead since it is already open to your project.

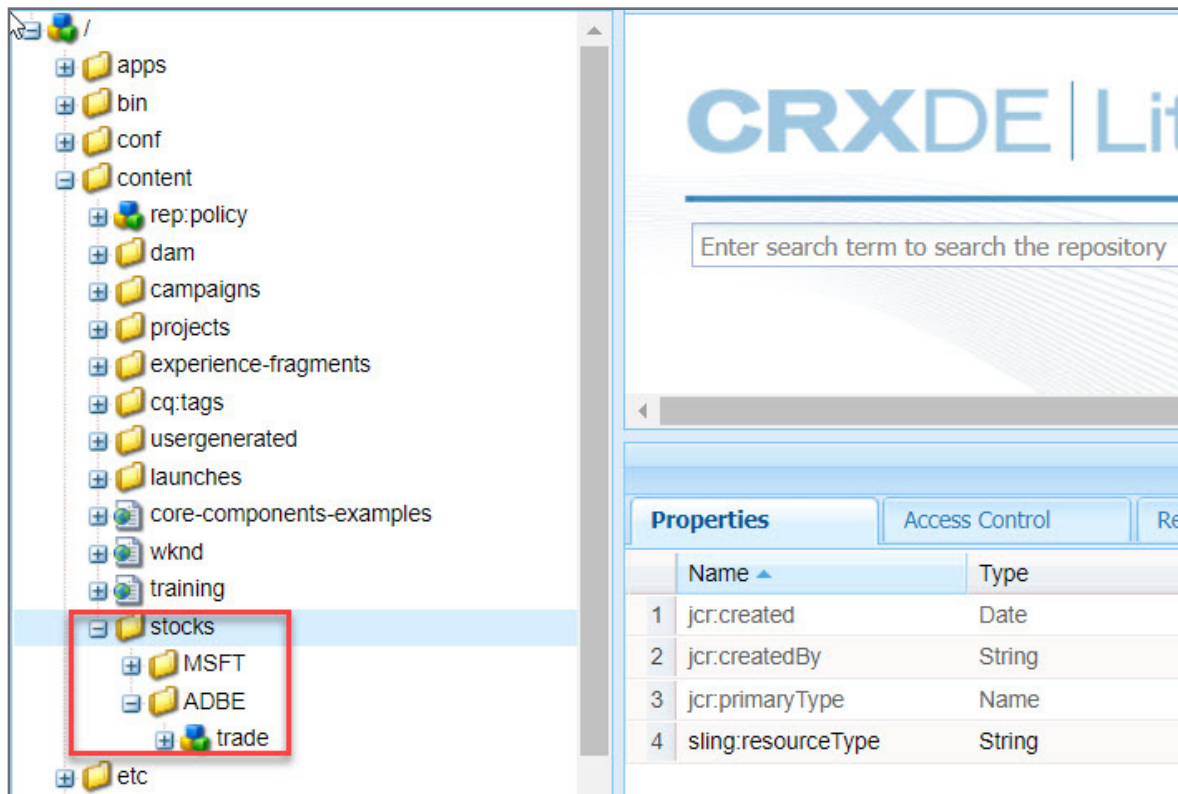
2. In the command prompt run the command:


```
$ mvn clean install -Padobe-public -PautoInstallSinglePackage
```

Your project has now successfully been installed into your local AEM Server.


Task 3: Test the service

1. Refresh CRXDE Lite. Verify the **stocks** folder has been created under the **/content** folder, as shown:



 **Note:** If you do not see these nodes right away, wait for 2 minutes since that is the schedule of the cron job configured.

2. Because **stocks** and the **symbols** are folders, we can see this information in the **Sites** console. Go to <http://localhost:4502/sites.html/content>.
3. Notice how you can see the **stocks** folder and the **symbol** folders under it.

 **Note:** The data cannot be viewed in this console since it is not an AEM page, but the UI suggests an author could create new symbols to be imported. This is what you will do in the next exercise.

Sling Resource Change Listening

For a component to be notified of changes happening in the repository, some techniques with fewer overhead than Sling Jobs and a richer interface than OSGi events can be considered. For example, the JCR Observation, which is a request in the JCR specification, is typically a mechanism that allows the ability to monitor changes on nodes or properties in a JCR repository, at a certain path and under certain conditions on nodetypes or UUIDs. This can be achieved by implementing the **javax.jcr.observation.EventListener interface** and creating an event listener with the `javax.jcr.observation.ObservationManager` interface.

However, because it is usually a better practice to use higher-level APIs, we can conveniently do the same with Sling and the **org.apache.sling.api.resource.observation.ResourceChangeListener** interface.

The **ResourceChangeListener** is registered with a mandatory property specifying the paths under which the change of the resource (or resource provider) is to be monitored, and optionally with the type of changes expected.

Resource changes can then be caught by overriding the method `onChange()` as shown in this example:

```
import org.apache.sling.api.resource.observation.ResourceChange;
import org.apache.sling.api.resource.observation.ResourceChangeListener;
@Component (immediate = true,
    service = ResourceChangeListener.class
    property = {"resource.paths=/content/mypath ",
        "resource.change.types=CHANGED"})

public class MyResourceListener implements ResourceChangeListener {
    @Override
    public void onChange(List<ResourceChange> changes) {
        // do something
    }
}
```

The advantage of this method over OSGi Events or Sling Jobs is that it allows the ability to restrict events to one (or several) specified path(s). Another advantage is that bulk listening operations will result in a single thread (the listener will only be executed once), whereas with OSGi Events or Sling Jobs, the handler would be called as many times as the amount of operations.

For these reasons, listening to only resource-related events is preferable, using the `ResourceChangeListener` for performance considerations.

Exercise 3: Create a resource listener

Scenario: An author goes to `sites.html` and creates a new Stock symbol folder under `/content/stocks`. This folder creates an OSGi config for the stock scheduler to import that stock symbol. If an author deletes a stock symbol folder, the OSGi config is removed.

In this exercise, you will create a resource listener that listens for ADDED or REMOVED resource changes made to `/content/stocks` and then add/remove OSGi configurations for the `StockImportScheduler`.

This exercise includes the following tasks:

1. Create a Sling Resource Listener
2. Install via command line
3. Test the service

Task 1: Create a Sling Resource Listener

In this task, you will create a java class named **StockListener.java**. This Sling listener listens to the `StockDataWriterJob.STOCK_IMPORT_FOLDER` (`/content/stocks`) location and creates a new scheduler config for each new stock folder added.

1. In your IDE, navigate to **core > src > main/java/com/adobe/training/core**.
2. Right-click **listeners** and choose **New File**.
3. Rename the file as **StockListener.java**.
4. Double-click **StockListener.java** to edit the file.
5. Copy the contents from **Exercise_Files-EC** under **/core/src/main/java/com/adobe/training/core/listeners/StockListener.java** to **StockListener.java** in your IDE, as shown:

```
package com.adobe.training.core.listeners;
import java.io.IOException;
import java.util.Dictionary;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.List;
import java.util.Map;
import com.adobe.training.core.StockDataWriterJob;
import com.adobe.training.core.schedulers.StockImportScheduler;
import org.apache.sling.api.resource.LoginException;
import org.apache.sling.api.resource.PersistenceException;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.api.resource.ResourceResolverFactory;
```

```

import org.apache.sling.api.resource.ResourceUtil;
import org.apache.sling.api.resource.observation.ResourceChange;
import org.apache.sling.api.resource.observation.ResourceChangeListener;
import org.osgi.framework.InvalidSyntaxException;
import org.osgi.service.cm.Configuration;
import org.osgi.service.cm.ConfigurationAdmin;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Reference;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * This Sling listener listens to the StockDataWriterJob.STOCK_IMPORT_FOLDER location and creates a
 * new scheduler config for each new stock folder added.
 *
 * To add a symbol from the UI, go to AEM Navigation > Sites > stocks and click the blue Create > Folder
 * Add the Stock symbol as the Title. Dummy stock data is available for ADBE,MSFT,GOOG,AMZN,APPL,WDAY
 *
 * Learn more about creating OSGi configurations programmatically:
 * http://www.nateyolles.com/blog/2015/10/creating-osgi-configurations-in-aem-and-sling
 */

@Component( immediate = true,
    property = {"resource.paths=" + StockDataWriterJob.STOCK_IMPORT_FOLDER,
        "resource.change.types=ADDED",
        "resource.change.types=REMOVED"
    })

public class StockListener implements ResourceChangeListener{
    private final String stockImportSchedulerPID = "com.adobe.training.core.schedulers.StockImportScheduler";

    private final Logger logger = LoggerFactory.getLogger(getClass());
    // Convenience string to find the log messages for this training example class
    // Logs can be found in crx-quickstart/logs/error.log
    private String searchableLogStr = "$$$$";

    // Service to get OSGi configurations
    @Reference
    private ConfigurationAdmin configAdmin;
    // Service to add/remove resources if needed
    @Reference
    private ResourceResolverFactory resourceResolverFactory;

    @Override
    public void onChange(List<ResourceChange> changes) {
        for (final ResourceChange change : changes) {
            logger.info(searchableLogStr + "Resource Change Detected: {}", change);

            //Get the folder name from the path. Ex: /content/stocks/adbe > adbe
            String folderName = change.getPath().substring(change.getPath().lastIndexOf("/") + 1);
            //In this example a stock symbol must be 4 characters and not be the 'trade' node from the
            if((folderName.length() == 4) && (folderName.matches("[a-zA-Z]*$")) && !folderName.

StockDataWriterJob
equals("trade")) {


                //Check if the added folder is uppercase. If it's not, autofix
                if(!folderName.equals(folderName.toUpperCase()) && change.getType().
                    //Get the service user (training-user) that belongs to the training.
                    Map<String, Object> serviceParams = new HashMap<>();
                    serviceParams.put(ResourceResolverFactory.SUBSERVICE,
                        try (ResourceResolver resourceResolver =
                            resourceResolverFactory.getServiceResourceResolver(serviceParams)) {

```


6. Examine the code and observe the following service and function.
 - **ConfigurationAdmin** service allows us to get/set OSGi configurations programmatically.
 - › **onChange()** – listens for changes to /content/stocks
 - » If the created folder is not uppercase, this is autofixed
 - » If it was a folder added, an OSGi config is created
 - » If it was a folder deleted, an OSGi config is deleted
7. Save the changes.

Task 2: Install via command line

1. Open a command prompt to the location of your Maven project. For example: C:/adobe/<myproject >

 **Note:** If you are using an IDE with an integrated terminal such as Visual Studio Code, you can run your Maven commands there instead since it is already open to your project.

2. In the command prompt run the command:

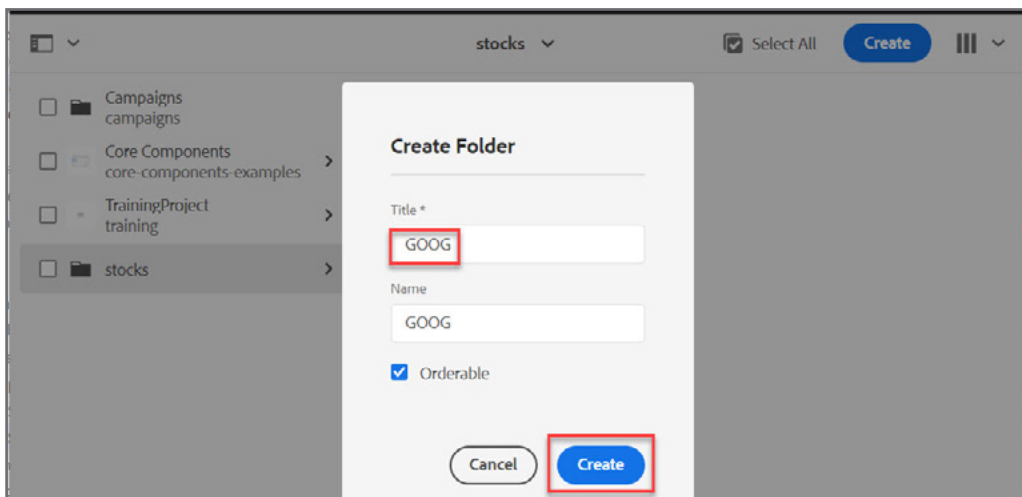
```
$ mvn clean install -Padobe-public -PautoInstallSinglePackage
```

Your project has now successfully installed into your local AEM Server.

Task 3: Test the service

1. Go to the browser that has the AEM author service running, and then go to **Sites > stocks**.
2. Click **Create > Folder**. The new folder is created.
3. Enter a stock symbol as the title (for example, **GOOG**, **AMZN**, **APPL**, or **WDAY**).

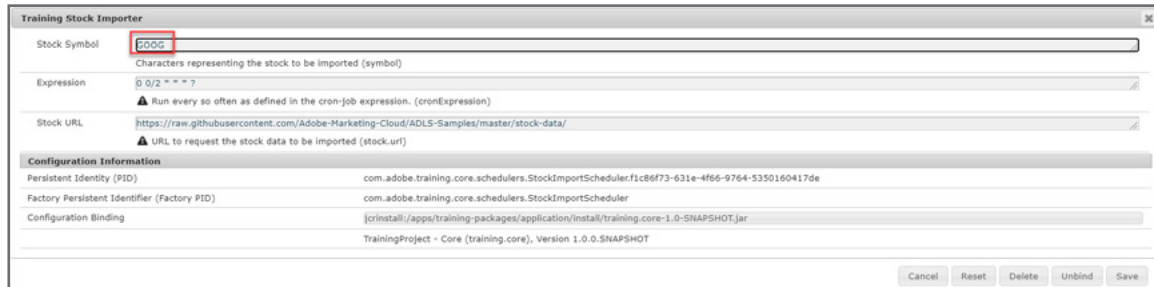
Click **Create**, as shown:



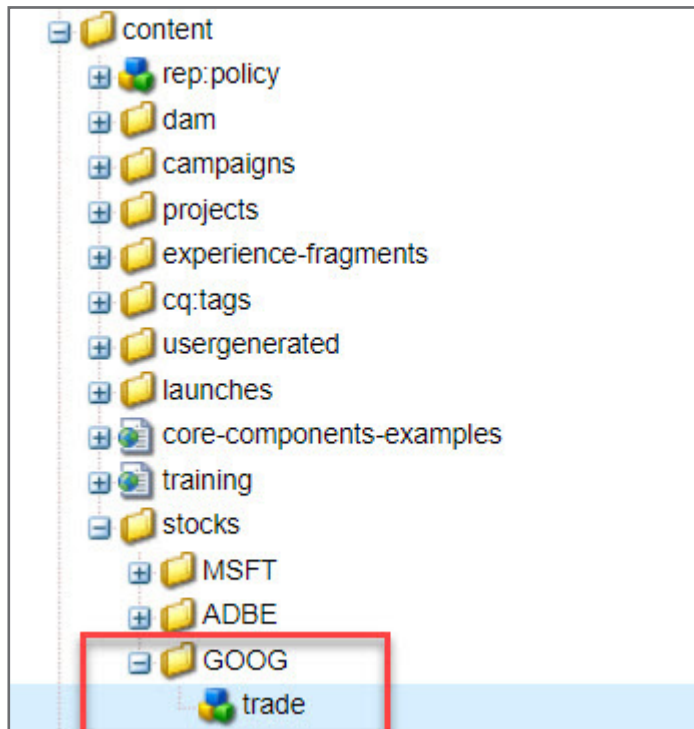
The folder is created.

This newly created folder will trigger the StockListener and auto create an OSGi configuration for the StockImportScheduler for that symbol.

4. Go to <http://localhost:4502/system/console/configMgr> and verify the new **Training Stock Importer** config, as shown:



5. Click **Cancel** to close the config window.
6. Wait for 2 minutes and then go to **CRXDE Lite** > **/content/stocks/<yourSymbol>** to observe the imported data, as shown:



7. Similarly, delete a stock folder under Sites > stocks and verify the Training Stock Importer config is deleted.

References

For more information on the events in your AEM Web Console:

<http://localhost:4502/system/console/events>

<http://localhost:4502/system/console/slingevent>