Adobe Experience Manager

Extend and Customize Adobe Experience Manager v6.x

Student Guide: Volume 2

ADOBE TRAINING SERVICES

Extend and Customize Adobe Experience Manager v6.x

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

October 25, 2016

Contents

CHAPTER EIGHT: INDEXING AND SEARCH

Overview

This chapter explains the indexing capabilities of Adobe Experience Manager. You will learn to configure indexes such as Lucene and Solr. You will also learn to use the indexing tools available out of the box. The Java Query Object Model is another section uses examples to explain indexing.

Objectives

By the end of this chapter, you will be able to:

- debug and log queries
- create a search servlet

Query Index

Oak does not index content by default as Jackrabbit 2 does. You need to create custom indexes when necessary, much like in traditional RDBMSs. If there is no index for a specific query, the repository will be traversed. That is, the query will still work but will probably be very slow.

If Oak encounters a query without an index, a WARN level log message displays:

```
*WARN* Traversed 1000 nodes with filter Filter(query=select...) consider creating an
index or changing the query
```

If this is the case, an index might need to be created, or the condition of the query might need to be changed to take advantage of an existing index.

If a query reads more than 10000 nodes in memory, then the query is cancelled with an UnsupportedOperationException saying that "*The query read more than 10000 nodes in memory. To avoid running out of memory, processing was stopped.*" As a workaround, you can change this limit using the system property "oak.queryLimitInMemory."

Query Indices are defined under the `oak:index` node.

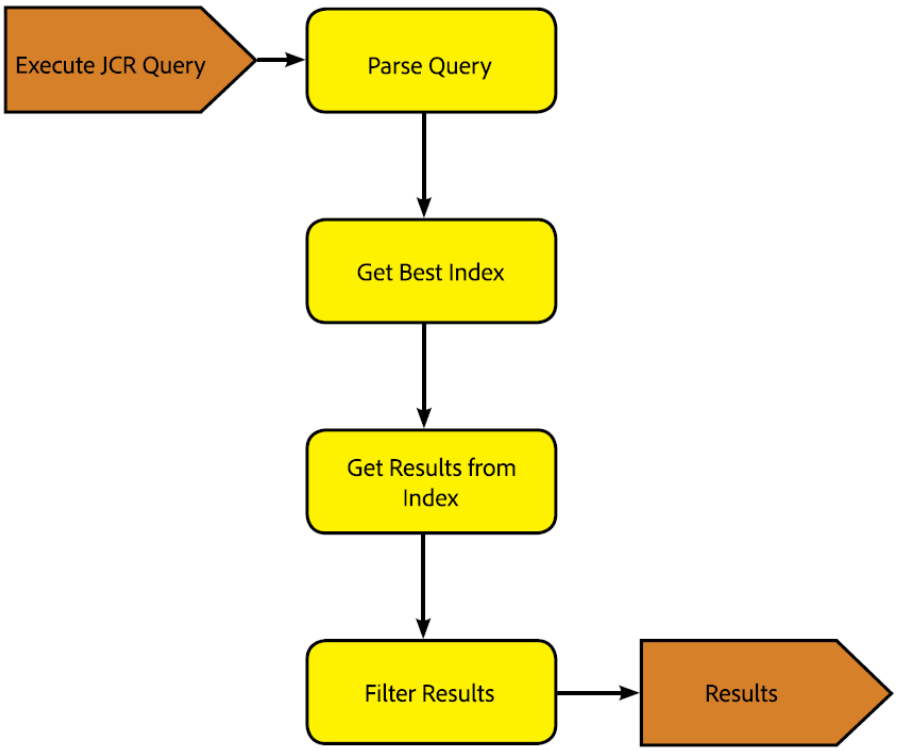
Supported Query Languages

The Oak query engine supports the following languages:

- XPath
- SQL
- SQL-2
- JQOM

Oak Query Implementation Overview

The new Apache Oak based backend allows different indexers to be plugged into the repository. The standard indexer is the Property Index, for which the index definition is stored in the repository itself. External full text indexers can also be used with Adobe Experience Manager. Implementations for Apache Lucene and Solr are available by default. The Traversal Index indexer is the one used if no other indexer is available. This means that the content is not indexed and all content nodes are traversed to find matches to the query. If multiple indexers are available for a query, each available indexer estimates the cost of executing the query. Oak then chooses the indexer with the lowest estimated cost.



The above diagram is a high-level representation of the query execution mechanism of Apache Oak.

First, the query is parsed into an Abstract Syntax Tree then the query is checked and transformed into SQL-2, which is the native language for Oak queries.

After the query is checked and transformed into SQL-2, each index is consulted to estimate the cost for the query. Once completed, the results from the most cost-effective index are retrieved. Finally, the results are filtered, both to ensure that the current user has read access to the result and that the result matches the complete query.

Query Processing on Cost Calculations

Internally, the query engine uses a cost-based query optimizer that asks all the available query indexes for the estimated cost to process the query. It then uses the index with the lowest cost.

By default, the following indexes are available:

- Property index for each indexed property
- Full-text index, which is based on Apache Lucene/Solr
- Node type index, which is based on an property index for the properties jcr:primaryType and jcr:mixins
- Traversal index that iterates over a subtree

If no index can efficiently process the filter condition, the nodes in the repository are traversed at the given subtree. Usually, data is read from the index and repository while traversing over the query result. There are exceptions however, where all data is read in memory when the query is executed—when using a full-text index and when using an "order by" clause.

Native Queries

To take advantage of features available in full-text index implementations such as Apache Lucene and Apache Lucene Solr, so-called native constraints are supported. These constraints are passed directly to the full-text index. This is supported for both XPath and SQL-2. For XPath queries, the name of the function is `rep:native,` and for SQL-2, it is `native`.

The first parameter is the index type and currently supported parameters are Solr and Lucene. The second parameter is the native search query expression.

For SQL-2, the selector name (if needed) is the first parameter, just before the language. If full-text implementation is not available, the queries will fail.

Configuring the Indexes

Indexes are configured as nodes in the repository under the `oak:index` node. The type of the index node must be `oak:QueryIndexDefinition`. Several configuration options are available for each indexer as node properties. For more information, see the following configuration details for Property Index, Lucene Index, and Solr Index.

The Property Index

The Property Index is generally useful for queries that have property constraints but are not full-text. It can be configured using the following procedure:

Open CRXDE Lite by going to http://localhost:4502/crx/de/index.jsp.

Create a new node under `oak:index`.

Name the node `PropertyIndex`, and set the node type to `oak:QueryIndexDefinition`.

Click **OK** then set the following properties for the new node:

Name	Type	Value
type	String	property
propertyNames	Name	jcr:uuid

Click **Save All** to save the changes – the **PropertyIndex** node will have three properties on its tab now.

This particular example will index the jcr:uuid property, whose job is to expose the Universally Unique Identifier (UUID) of the node it is attached to.

The Lucene Full-text Index

A full-text indexer based on Apache Lucene is available in Adobe Experience Manager 6.2. If a full-text index is configured, then all queries that have a full-text condition use the full-text index, no matter if there are other conditions that are indexed, and no matter if there is a path restriction.

If no full-text index is configured, then queries with full-text conditions may not work as expected. The query engine has a basic verification in place for full-text conditions, but it does not support all features that Lucene does and it will traverse all nodes if there are no indexed constraints.

As the index is updated through an asynchronous background thread, some full-text searches will be unavailable for a small window of time until the background processes are finished.

You can configure a Lucene full-text index using the following procedure:

Open CRXDE Lite and create a new node under oak:index.

Name the node **LuceneIndex** and set the node type to `oak:QueryIndexDefinition`.

Add the following properties to the node:

Name	Type	Value
type	String	lucene
async	String	async

Save the changes.

The Solr Index

You can use the Solr index for full-text search, as well as search by path, property restrictions, and primary type restrictions.

- It can be used for any type of JCR query.
- Its integration with Adobe Experience Manager occurs at the repository level.
- It can be configured to work as an embedded server with the Adobe Experience Manager instance, or as a remote server.

To configure Adobe Experience Manager with an embedded Solr server:

Navigate to the Web Console at http://localhost:4502/system/console/configMgr

Search for `Oak Solr server provider`.

Click the Edit icon, and in the resulting dialog box, select the server type as `Embedded Solr` from the drop-down list.

Click **Save**.

Search for `Oak Solr embedded server configuration`.

Click the Edit icon, and update the configuration. The Solr home directory configuration will look for a folder with the same name in the Adobe Experience Manager installation folder.

Open CRXDE Lite, and login as Admin.

Under oak:index, create a node called `solrIndex`, of type `oak:QueryIndexDefinition` with the following three properties (enter the information for each field in each line below by clicking Add to create a new row on the Properties tab for the corresponding lines and values below):

a. Name: type, Type: String, Value: solr

b. Name: async, Type: String, Value: async

c. Name: reindex, Type: Boolean, Value: true

Click **Save All** in the upper-left corner.

Temporarily Disabling an Index

To temporarily disable an index (for example, for testing), set the index type to disabled. While the index type is not set, the index is not updated, so if you enable it again, it might not be correct. This is especially important for synchronous indexes.

Indexing Tools

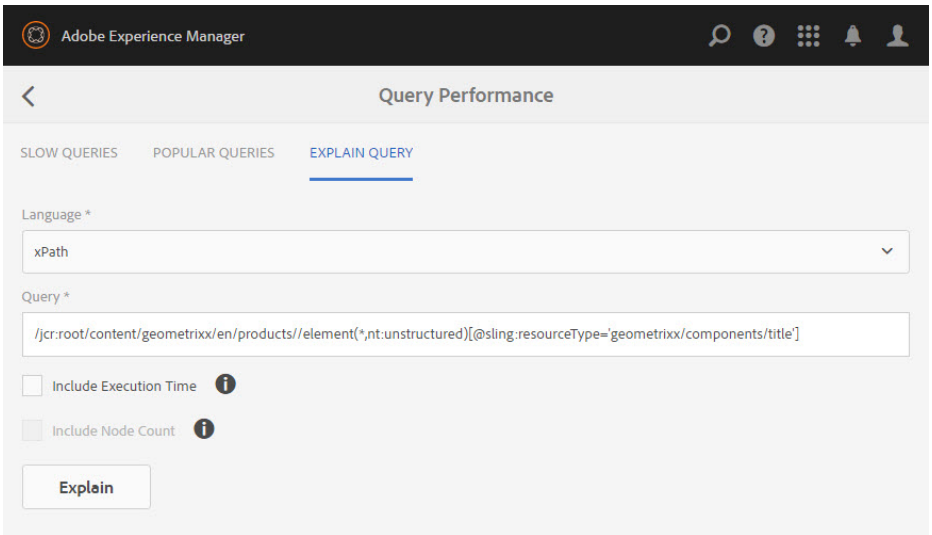
The Adobe Consulting Service Commons toolkit introduces a set of indexing tools in an effort to simplify management and maintenance of indexes in the underlining data storage layer of Adobe Experience Manager, Apache Oak.

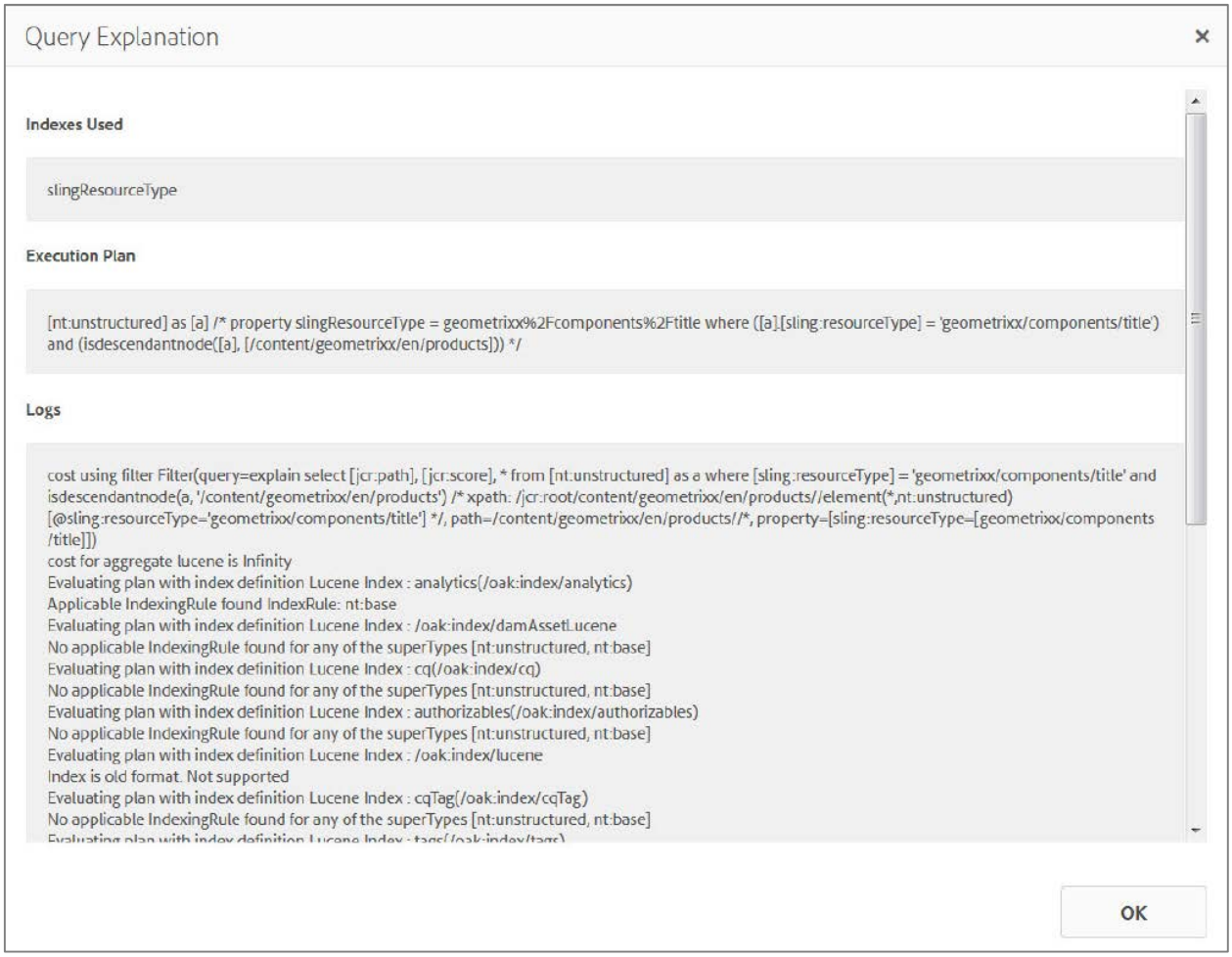
Explain Query Tool

This is a tool designed to help administrators understand how queries are executed. Oak attempts to figure out the best way to execute any given query, based on the Oak indexes defined in the repository under the oak:index node. Depending on the query, different indexes may be chosen by Oak. Understanding how Oak is executing a query is the first step to optimizing the query.

Features:

- Supports the Xpath, JCR-SQL, and JCR-SQL2 query languages
- Reports the actual execution time of the provided query
- Detects slow queries and warns about queries that could be potentially slow
- Reports the Oak index used to execute the query
- Displays the actual Oak Query engine explanation
- Provides click-to-load list of Slow and Popular queries





The first entry in the Query Explanation section is the actual explanation. The explanation will show the type of index that was used to execute the query.

The second entry is the query plan. Selecting the **Include execution time** check box before running the query will also show the amount of time the query was executed in, allowing for more information that can be used for optimizing the indexes for your application or deployment.

The tool will also build a list of popular and slow queries that can be automatically loaded in the UI by selecting their name in the predefined list.

Oak Index Manager

The Oak Index Manager is a simple Web UI created to facilitate index management such as maintaining indexes, viewing their status, or triggering re-indexes. You can the UI to filter indexes in the table by typing in the filter criteria in the search box in the upper-left corner of the screen.

You can trigger a single reindex by clicking the icon in the Reindex column for the respective index. You can trigger a bulk reindex by selecting multiple indexes and clicking the Bulk Reindex button.

With Adobe Experience Manager 6.2, both the Explain Query and Oak Index Manager tools will be available out-of-the-box. You can access them by going to **Tools** > **Operations** > **Dashboard** > **Diagnosis** from the Adobe Experience Manager Welcome screen.

Repository Configuration

You can use the ConfigMgr (http://localhost:4502/system/console/configMgr) to configure the repository.

- External Login module: Oak has a new concept of external login modules. JAAS login modules can be deployed as OSGi bundles with startup level 15. The external login module can be activated by the Apache Oak External Login Module. Configuration of the synchronization of the user data with the external login module is configured in the section, Apache Oak Default Sync Handler.
- LDAP configuration: With this, you can configure LDAP. This configuration is available in the system console section, Apache Oak LDAP Identity Provider.

Basic Content Access

The JCR provides fast access by path and ID. The underlying storage is addressed by ID, but path traversal is also required for ACL checks. The caches are optimized for a reasonably sized active working set. The typical web access pattern:

- A handful of key resources
- A long tail of less frequently accessed content
- Few writes

Writing can result in a performance hit especially when updating nodes with many child nodes.

Batch Processing

Batch processing presents two separate issues: read and write.

When reading lots of content:

- Implement tree-traversal as the best approach, but this approach will flood the caches.
- Schedule for off-peak times.
- Add an explicit delay (used by the garbage collectors).
- Use a dedicated cluster node for batch processing.

When writing lots of content (including deleting large sub-trees):

- Remember, the entire transient space is kept in memory and committed atomically.
- Split the operation to smaller pieces.
- Save after every ~1,000 nodes.
- Leverage the data store if possible.

CRX Search Features Not Specified by the JSR

The following features are not specified by the JSR, but have been added to the CRX as a part of the Jackrabbit implementation:

- Extract text from binary content.
- Get a text excerpt with highlighted words that matched the query (ExcerptProvider).
- Search for a term and its synonyms (SynonymSearch).
- Search for similar nodes (SimilaritySearch).
- Define index aggregates, rules, and scores (IndexingConfiguration).
- Check the spelling of a full-text query statement (SpellChecker).

Query Syntax

As specified, JSR 170: SQL and XPath syntaxes have the same feature set. Since JSR-283, Abstract Query Model (AQM) defines the structure and semantics of a query. The specification defines two language bindings for AQM:

- JCR-SQL2 (Grammar: http://www.h2database.com/jcr/grammar.html)
- JCR-JQOM

Basic AQM Concepts

A query has one or more selectors. When the query is evaluated, each selector independently selects a subset of the nodes in the workspace based on Node type.

`Join` transforms the multiple sets of nodes selected by each selector into a single set. The `join` type can be inner, left-outer, or right-outer.

AQM Concepts: Constraints

A query can specify a constraint to filter the set of node-tuples. The constraints may be any combination of:

- Absolute or relative path. For example, nodes that are children of /pictures
- Name of the node.
- Value of a property. For example, nodes whose jcr:created property is after 2007-03-14T00:00:00.000Z.
- Length of a property. For example, nodes whose jcr:data property is longer than 100 KB.
- Existence of a property. For example, nodes with a jcr:description property.
- Full-text search. For example, nodes that have a property containing the phrase "beautiful sunset."

Search Basics

Defining and executing a JCR-level search requires the following logic:

```
QueryManager qm = session.getWorkspace().getQueryManager();
```

Get the QueryManager for the Session/Workspace:

```
Query q = qm.createQuery("select * from moviedb:movie order by Title",Query.SQL);
```

Create the query statement:

Execute the query:

```
QueryResult res = q.execute();
```

Iterate through the results:

```
NodeIterator nit = res.getNodes();
```

Query Examples—SQL2

Find all nt:folder nodes.

```
SELECT * FROM [nt:folder]
```

Find all files under /var. Exclude files under /var/classes.

```
SELECT * FROM [nt:file] AS files
WHERE ISDESCENDANTNODE(files, [/var])
AND (NOT ISDESCENDANTNODE(files, [/var/classes]))
```

Find all files under /var (but not under /var/classes) created by existing users. Sort the results in ascending order by jcr:createdBy and jcr:created.

```
SELECT * FROM [nt:file] AS file
INNER JOIN [rep:User] AS user ON file.[jcr:createdBy] = user.
[rep:principalName]
WHERE ISDESCENDANTNODE(file, [/var])
AND (NOT ISDESCENDANTNODE(file, [/var/classes]))
ORDER BY file.[jcr:createdBy], file.[jcr:created]
```

Java Query Object Model

Java Query Object Model (JQOM) is a mapping of AQM to Java API, and expresses a query as a tree of Java objects. The package `javax.jcr.query.qom` defines the API.

A query is built using `QueryObjectModelFactory` and its `createQuery` method. For example:

```
QueryManager qm = session.getWorkspace().getQueryManager();
QueryObjectModelFactory qf = qm.getQOMFactory();
QueryObjectModel query = qf.createQuery( ... );
```

JQOM Examples

Find all nt:folder nodes.

```
QueryObjectModelFactory qf = qm.getQOMFactory();
Source source = qf.selector("nt:folder", "ntfolder");
QueryObjectModel query = qf.createQuery(source, null,
null,null);
```

Find all files under /var. Exclude files under /var/classes.

```
QueryObjectModelFactory qf = qm.getQOMFactory();
Source source = qf.selector("nt:file", "files");
Constraint pathConstraint = qf.and(qf.descendantNode("files",
"/var"), qf.not(qf.descendantNode("files", "/var/classes")));
QueryObjectModel query = qf.createQuery(source,
pathConstraint, null,null);
```

Find all files under /var (but not under /var/classes) created by existing users. Sort the results in ascending order by jcr:createdBy and jcr:created.

```
QueryObjectModelFactory qf = qm.getQOMFactory();
Source source = qf.join(qf.selector("nt:file", "file"), qf.
selector("rep:User", "user"), QueryObjectModelFactory.JCR _
JOIN _ TYPE _ INNER,qf.equiJoinCondition("file","jcr:createdBy",
"user","rep:principalName"));
Constraint pathConstraint = qf.and(qf.descendantNode("file", "/
var"), qf.not(qf.descendantNode("file", "/var/classes")));
Ordering orderings[] = {qf.ascending(qf.propertyValue("file",
"jcr:createdBy")), qf.ascending(qf.propertyValue("file",
"jcr:created")) };
QueryObjectModel query = qf.createQuery(source,
pathConstraint, orderings, null);
```

Search Performance

Which JCR search methodologies are the fastest?

- Constraints on properties, Node types, full-text
- Typically O(n), where n is the number of results, vs. to the total number of nodes

Which JCR search methodologies are fast?

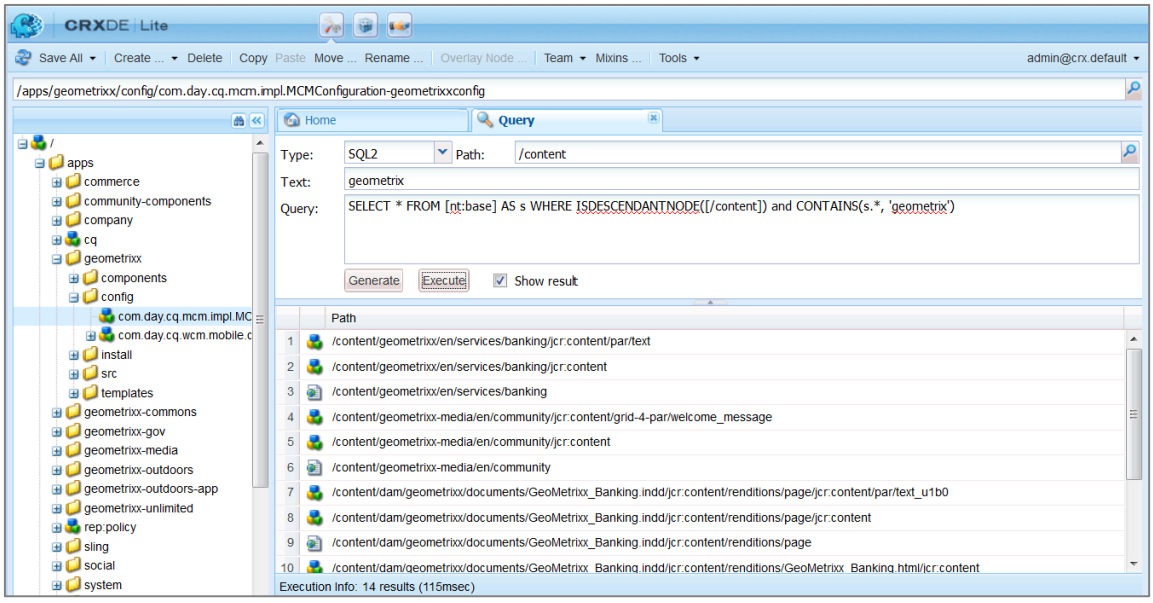
- Constraints on the path

Which JCR methodologies need some planning?

- Constraints on the child axis
- Sorting, limit/offset
- Joins

Testing Queries

You can test your queries using CRXDE Lite. Access the Query Tool on the Tools menu from the top toolbar in CRXDE Lite.



Perform Task – Creating query logger, from the Lab Activity section.

Perform Task – Create a servlet for search, from the Lab Activity section.

Chapter 8 Lab Activity

Scenario

You need to search and analyze the log files created for a logger.

Challenge

A logger needs to be created before you can implement the search and analyze functionality as per the requirement.

Pre-requisites

Existing Adobe Experience Manager Author server running on your machine and existing trainingproject.

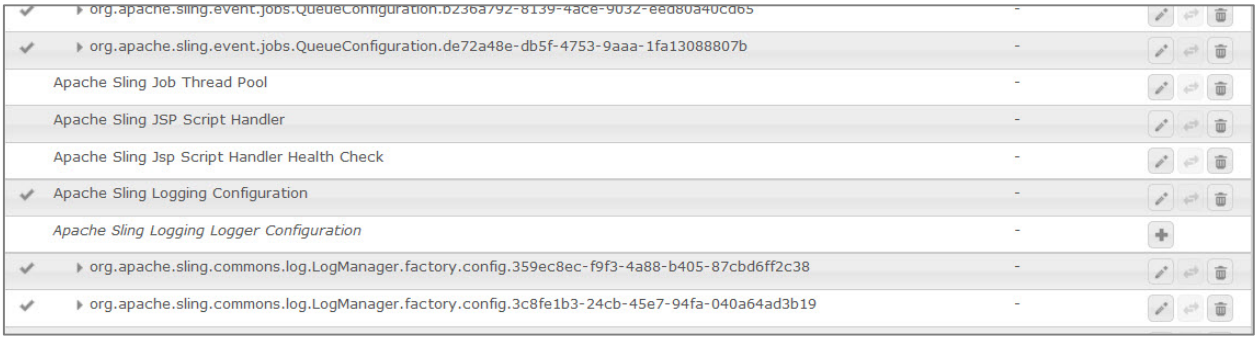
Steps

1. Task – Creating a query logger

You need to create a logger, which will log the events for **org.apache.jackrabbit.oak.query** into query.log file. To accomplish this, follow the steps below:

Open the configuration manager at http://localhost:4502/system/console/configMgr

Search for **Apache Sling Logging Logger configuration** and click on + sign to add your logger configuration.



Provide the logger information as:

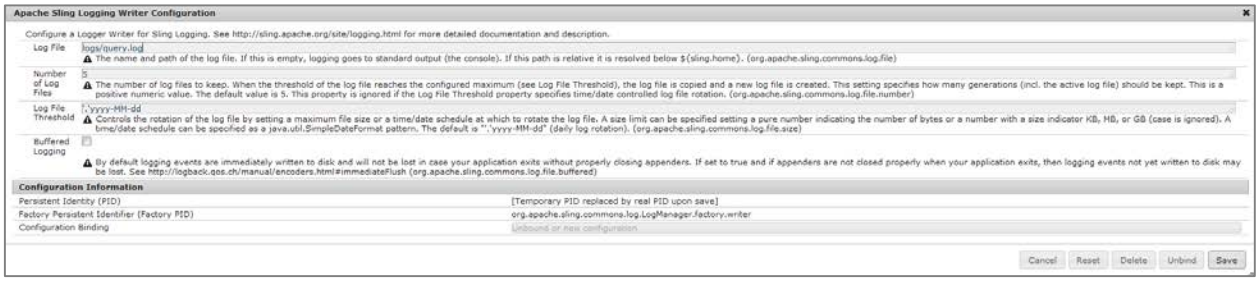
Log Level	Debug
Log File	logs/query.log
Message Pattern	{0,date,dd.MM.yyyy HH:mm:ss.SSS} *{4}* [{2}] {3} {5}
Logger	org.apache.jackrabbit.oak.query

Click **Save**. The new Logging Logger will direct its output to a new log file named query.log.

Search for **Apache Sling Logging Writer Configuration** and click on + sign to add your logger configuration.

Update the root logger configuration as follows:

Log File	logs/query.log
Number of Log files	5
Log File Threshold	'.'yyyy-MM-dd



Click **Save**. The new Logging Writer will control the number of files that exist for the **query.log** before they begin to roll (overwrite).

2. Task - Create a Servlet for Search

Now that you have configured logging for the query classes, it is time to create a servlet that implements a Sling selector. The servlet will perform a full-text search starting with the specified node. The result set will be a set of pages, but you will specify Node type nt:unstructured because you know that the jcr:content child of each page is of type nt:unstructured.

Open Eclipse and open **trainingproject**, create a servlet named **SearchServlet** under **com.adobe.training.core.servlets,** with the following code:

```
package com.adobe.training.core.servlets;
import java.io.IOException;
import javax.jcr.Node;
import javax.jcr.NodeIterator;
import javax.jcr.RepositoryException;
import javax.jcr.ValueFactory;
import javax.jcr.query.qom.Constraint;
import javax.jcr.query.qom.QueryObjectModel;
import javax.jcr.query.qom.QueryObjectModelFactory;
import javax.jcr.query.qom.Selector;
import javax.servlet.ServletException;
import org.apache.felix.scr.annotations.sling.SlingServlet;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.SlingHttpServletResponse;
import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
import org.apache.sling.commons.json.JSONArray;
import org.apache.sling.commons.json.JSONObject;

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

@SlingServlet(resourceTypes = "geometrixx/components/homepage", selectors = "search")
public class SearchServlet extends SlingSafeMethodsServlet {

   private static final long serialVersionUID = 3169795937693969416L;

   @Override
   public final void doGet(final SlingHttpServletRequest request, final
   SlingHttpServletResponse response)
         throws ServletException, IOException {
      response.setHeader("Content-Type", "application/json");
      JSONObject jsonObject = new JSONObject();
      JSONArray resultArray = new JSONArray();

      try {
         // this is the current node that is requested, in case of a page that is the
   jcr:content node
         Node currentNode = request.getResource().adaptTo(Node.class);
         PageManager pageManager =
   request.getResource().getResourceResolver().adaptTo(PageManager.class);
```

```
        // node that is the cq:Page containing the requested node
        Node queryRoot =
pageManager.getContainingPage(currentNode.getPath()).adaptTo(Node.class);

        String queryTerm = request.getParameter("q");
        if (queryTerm != null) {
            NodeIterator searchResults = performSearch(queryRoot, queryTerm);
            while (searchResults.hasNext())
resultArray.put(searchResults.nextNode().getPath());
            jsonObject.put("results", resultArray);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    //     response.getWriter().print("SQL VERSION");
    response.getWriter().print(jsonObject.toString());
    response.getWriter().close();
}

private NodeIterator performSearch(Node queryRoot, String queryTerm) throws
RepositoryException {

    // JQOM infrastructure
    QueryObjectModelFactory qf =
queryRoot.getSession().getWorkspace().getQueryManager().getQOMFactory();
    ValueFactory vf = queryRoot.getSession().getValueFactory();

    final String SELECTOR_NAME = "all results";
    final String SELECTOR_NT_UNSTRUCTURED = "nt:unstructured";
    // select all unstructured nodes
    Selector selector = qf.selector(SELECTOR_NT_UNSTRUCTURED, SELECTOR_NAME);

    // full text constraint
    Constraint constraint = qf.fullTextSearch(SELECTOR_NAME, null,
qf.literal(vf.createValue(queryTerm)));
    // path constraint
    constraint = qf.and(constraint, qf.descendantNode(SELECTOR_NAME,
queryRoot.getPath()));

    // execute the query without explicit order and columns
    QueryObjectModel query = qf.createQuery(selector, constraint, null, null);
    return query.execute().getNodes();

}

}
```

Build and deploy the bundle. The results are available at the following URL:
http://localhost:4502/content/geometrixx/en.search.html?q=ceo&wcmmode=disabled

Implement the same functionality using SQL2.

```
private NodeIterator performSearchWithSQL(Node queryRoot,String queryTerm) throws
    RepositoryException {
QueryManager qm = queryRoot.getSession().getWorkspace().getQueryManager();
Query query = qm.createQuery("SELECT * FROM [nt:unstructured]AS node WHERE ISDESCENDANTNODE([" +
    queryRoot.getPath() + "])and CONTAINS(node.*, '" + queryTerm + "')", Query.JCR_SQL2);
return query.execute().getNodes();
}
```

Scenario Conclusion

You wrote a servlet that implements a Sling selector, performs a full text search, and returns the query results as JSON.

CHAPTER NINE: CONFIGURING CUSTOM LOG FILES

Overview

Some Adobe Experience Manager log files provide detailed information about the current system state. In addition to the default system log files, you can also create and customize your own log files. They can help you better track messages produced by your own applications and separate them from the default log entries.

Adobe Experience Manager offers you the possibility to configure:

- global parameters for the central logging service.
- request data logging; a specialized logging configuration for request information.
- specific settings for the individual services; for example, an individual log file and format for the log messages.

This chapter provides instructions on configuring your own custom log files.

Objectives

By the end of this chapter, you will be able to:

- create custom log files

Understanding the Logging System

Logging in Adobe Experience Manager is based on Sling, and is supported by the `org.apache.sling.commons.log` bundle. This bundle has the following features:

- Implements the OSGi Log Service Specification and registers the `LogService` and `LogReader` services
- Exports four commonly used logging APIs:
  - Apache Commons Logging
  - Simple Logging Facade for Java (SLF4J)
  - log4j
  - java.util.logging
- Configures logging through Logback, which is integrated with the OSGi environment
- Allows logging to be configured both via editing Logback xml or via OSGi configurations

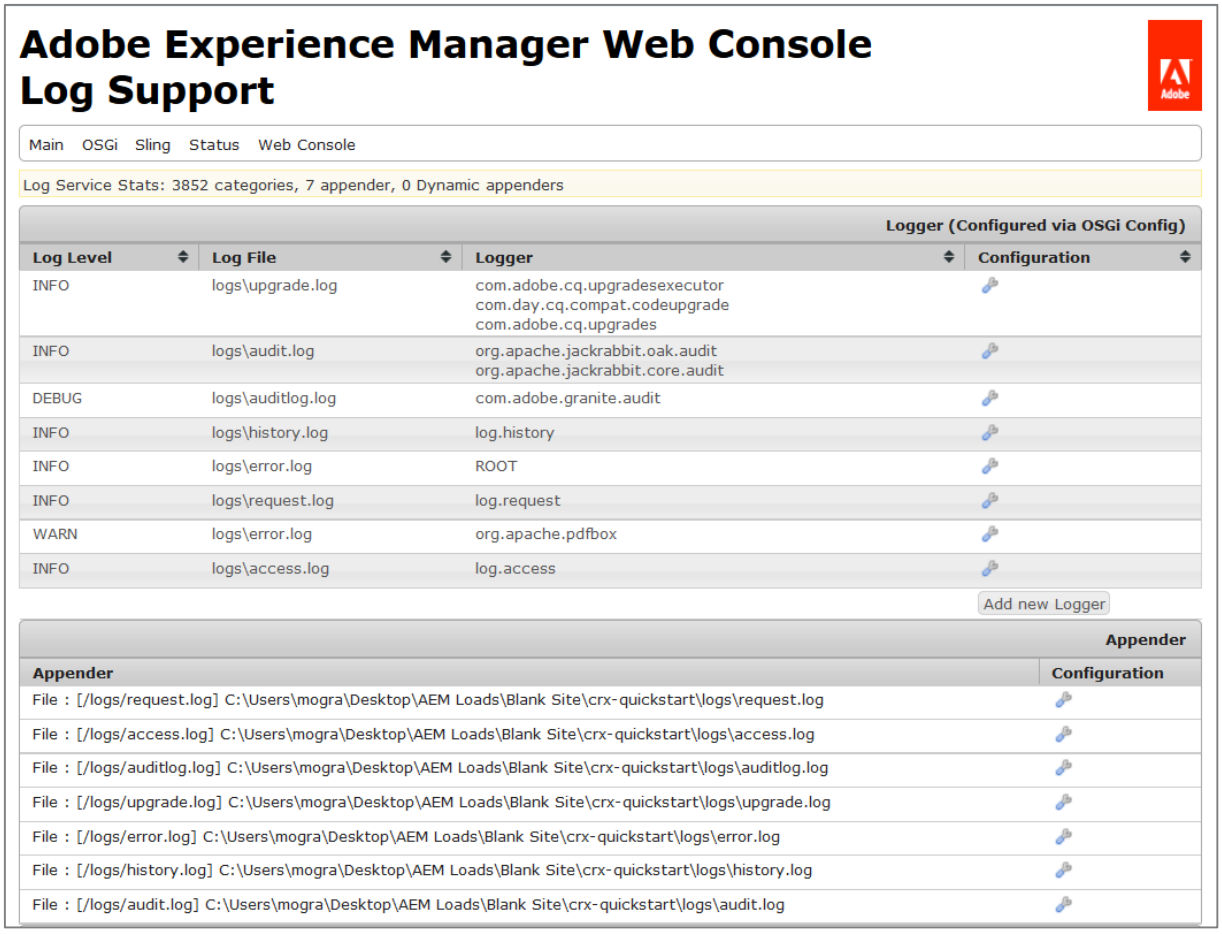
Types of Log Files in Adobe Experience Manager

Here is a brief description of the types of log files you will find within Adobe Experience Manager. These files are available in the installation directory: **`/crx-quickstart/logs`**

- `access.log`—registers all access requests sent to Adobe Experience Manager and the repository.
- `audit.log`—registers all moderation actions.
- `error.log`—registers all error messages.
- `request.log`—registers all access requests along with their responses.
- `stderr.log`—holds error messages generated during startup.
- `stdout.log`—holds logging messages indicating events during startup.
- `upgrade.log`—provides a log of all upgrade operations.

By default, the error, access, history and request logs rotate once per day. When this occurs, the existing log files are appended with a timestamp, and a new file is created.

In Adobe Experience Manager, you can view the log files through the Web Console at:
http://localhost:4502/system/console/slinglog



Loggers and Writers

Adobe Experience Manager's logging system consists of two elements—a **Logging Logger** and a **Logging Writer**. The Writer persists the data provided by the Logger to a configurable location. Usually, it is a file. For example, the **error.log file** is created by the Writer on a rotational basis. The Logger collects data from different components inside Adobe Experience Manager, filters them by requested severity level, and redirects the output to a configured Writer.

Adobe Experience Manager lets you configure two types of settings:

- Global logging. This defines:
  - logging level
  - central log file location
  - number of versions saved
  - version rotation; either maximum size or a time interval
  - format used when writing the log messages
- Loggers and Writers for an individual service. This defines:
  - specific logging level
  - individual log file location

- tnumber of versions to be kept
- version rotation; either maximum size or the time interval
- format used when writing the log messages
- logger (the OSGi service supplying the log messages)

1.1.1 Individual Service Loggers and Writers

You can channel log messages for a single service into a separate file. Adobe Experience Manager uses the following process to write log messages to file:

OSGi service (logger) writes a log message.

Logging Logger takes this message and formats it according to your specification.

Logging Writer writes all these messages to the physical file that you defined.

These elements are linked by the following parameters:

- **Logger (Logging Logger):** Defines the service(s) generating the messages.
- **Log File (Logging Logger):** Defines the physical file for storing the log messages. This parameter links a Logging Logger with a Logging Writer.
- **Log File (Logging Writer):** Defines the physical file that the log messages will be written to.

1.1.2 Standard Loggers and Writers

Once Adobe Experience Manager is installed, you have access to the following Writers and Loggers:

Logger	Links To
Apache Sling Customizable Request Data Logger (**`org.apache.sling.engine.impl.log.RequestLoggerService`**) Writes messages about requests to the request.log file.	Apache Sling Request Logger (**`org.apache.sling.engine.impl.log.RequestLogger`**) Writes messages to either request.log or access.log.
Apache Sling Logging Logger Configuration (**`org.apache.sling.commons.log.LogManager.factory.config`**) Writes information messages to logs/error.log.	Apache Sling Logging Writer Configuration (Writer) (**`org.apache.sling.commons.log.LogManager.factory.writer`**)
Apache Sling Logging Logger Configuration (**`org.apache.sling.commons.log.LogManager.factory.config.649d51b7-6425-45c9-81e6-2697a03d6be7`**) Writes Warning messages `to ../logs/error.log` for the service `org.apache.pdfbox`	Does not link to any specific Writer. It will create and use an implicit Writer with default configuration

Creating Your Own Loggers and Writers

You can define your own Logger / Writer pair as follows:

Create a new instance of the Factory Configuration Apache Sling Logging Logger Configuration.

d. Specify the Log File.

e. Specify the Logger.

f. Configure the other parameters as required.

Create a new instance of the Factory Configuration Apache Sling Logging Writer Configuration.

g. Specify the Log File.

h. Configure the other parameters as required.

There are two ways to configure the Adobe Experience Manager's Logging system—using the Apache Felix Web Console and CRX (CRXDE Lite). Adobe recommends configuring everything in CRX and using the Apache Web Console as a viewer only.

Creating the Logging Logger

In the following task, you will create a new Logger module based on a factory module (Apache Sling Logging Logger Configuration) using CRX. It should log protocol messages generated by the modules `org.apache.felix` and `com.day`, filtering out messages with log severity higher than 'Debug'.

Perform Task – Create Logging Logger, from the Lab Activity section.

Creating the Logging Writer

A logging Writer is only necessary for a configuration that is different than the default one. The default Writer will select a default size of 10 MB, and 5 files. Limit the file size of your Logger to 1 MB per file, and keep 3 log files on the hard disk.

Perform Task – Create Logging Writer, from the Lab Activity section.

Chapter 9 Lab Activity

Scenario

You need to keep track of all users logging into Adobe Experience Manager through a log file.

Challenge

Keeping a log of all users logging into Adobe Experience Manager is an ongoing process and would result in an extremely large file. You need to ensure that the log file holds the records of the current date, and moves historical logs into an archive. You will identify the criteria for limiting the file; the criteria being size of the file or date of creation.

Overview

> You will create a logger for **org.apache.sling.auth.core.impl.SlingAuthenticator** in a new log file named LoginTrace.log. This logger will capture the logging event for each user who logs in to the Adobe Experience Manager.
>
> You will create a logging writer that will rollover at the beginning of every minute while logging **org.apache.sling.auth.core.impl.SlingAuthenticator** events. Based on the size of the file or the date of creation of the file created by the logger, the file will be renamed with a time stamp, and a new file created. This is done to limit the size of the current log file.
>
> You need to identify a logger so that you can demonstrate the scenario in OOTB (Out Of The Box) Adobe Experience Manager instance. Creating multiple logging writers for the same log file will create a concurrent writer issue. You should be careful and keep the log file name in consideration while creating logging writer.

Pre-requisites

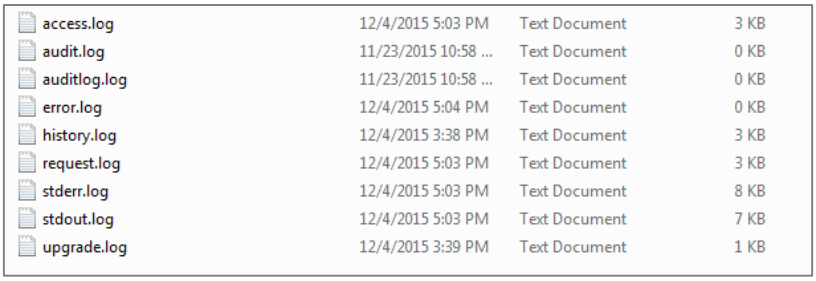
You need to be ready with the following:

- Running Adobe Experience Manager Author instance

Steps

1. Task - Create Logging Logger

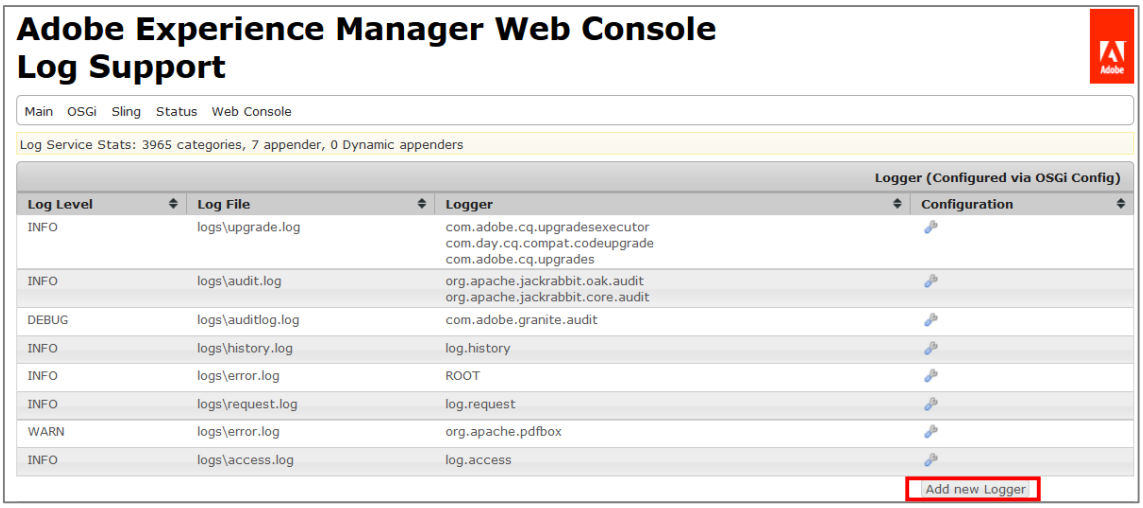
   Open the Adobe Experience Manager instance folder (**C:\adobe\AEM\author**) and navigate to **crx-quickstart\logs** directory where you will see the default logs being created.



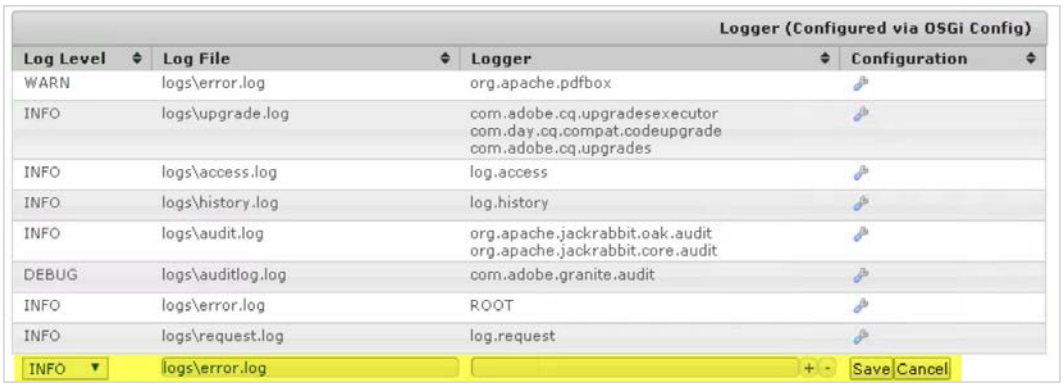
   After the new logger is created, you can see the log file in this folder. You can configure the log file location anywhere you want on your file system; however, we will use the current directory for this task.

   Open the Log Support console at: http://localhost:4502/system/console/slinglog

Click **Add new Logger**.



A new row is created (highlighted).

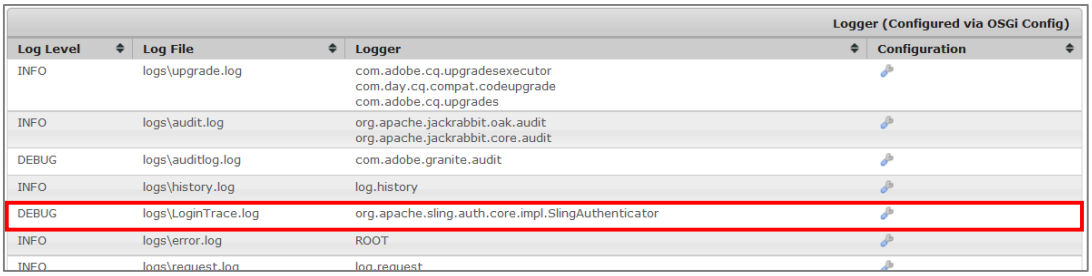


In the **Log Level** drop-down (where INFO is the default), select **Debug**.

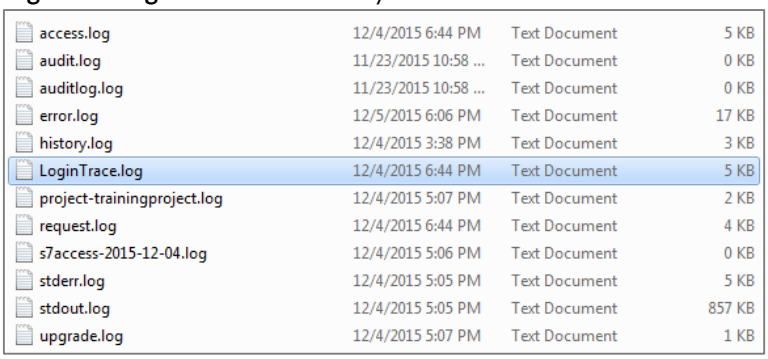
Enter the Log File name as logs\LoginTrace.log.

Enter the Logger as org.apache.sling.auth.core.impl.SlingAuthenticator.

Click **Save**. You will see that the logger is created successfully.



Navigate to **crx-quickstart\logs** directory again (**C:\adobe\AEM\author\crx-quickstart\logs**) and check if the **LoginTrace.log** is created successfully.



access.log	12/4/2015 6:44 PM	Text Document	5 KB
audit.log	11/23/2015 10:58 ...	Text Document	0 KB
auditlog.log	11/23/2015 10:58 ...	Text Document	0 KB
error.log	12/5/2015 6:06 PM	Text Document	17 KB
history.log	12/4/2015 3:38 PM	Text Document	3 KB
LoginTrace.log	12/4/2015 6:44 PM	Text Document	5 KB
project-trainingproject.log	12/4/2015 5:07 PM	Text Document	2 KB
request.log	12/4/2015 6:44 PM	Text Document	4 KB
s7access-2015-12-04.log	12/4/2015 5:06 PM	Text Document	0 KB
stderr.log	12/4/2015 5:05 PM	Text Document	5 KB
stdout.log	12/4/2015 5:05 PM	Text Document	857 KB
upgrade.log	12/4/2015 5:07 PM	Text Document	1 KB

Log out from Adobe Experience Manager, and then log in again. This will ensure that the events are captured in the log file. Observe that the log file contains entries as you configured them in the logger settings.

2. Task - Create Logging Writer [optional]

Log files can grow rather quickly and fill up available disk space. To cope with this growth, log files may be rotated in two ways:

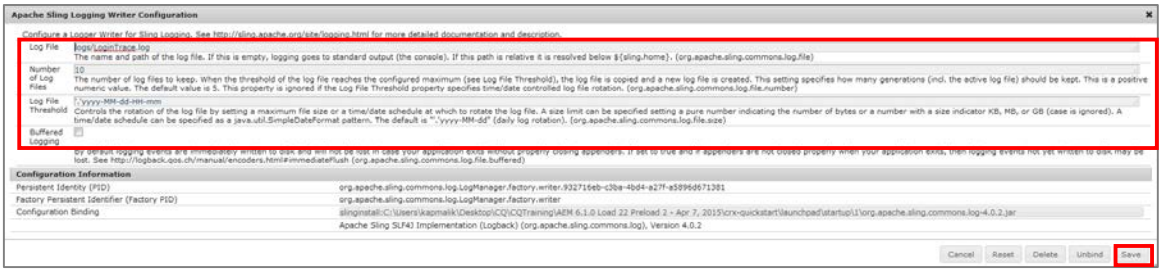
- At specific times
- When the log file reaches a configurable size

The first method is called Scheduled Rotation and is used by specifying a **SimpleDateFormat** pattern as the log file "size." The second method is called Size Rotation and is used by setting a maximum file size as the log file size.

In this task, you will create a logging writer, which will Rollover at the top of every hour and generate one log file each hour. Follow these steps:

Open configuration manager at http://localhost:4502/system/console/configMgr and search for **Apache Sling Logging writer Configuration** and click the '+' icon.

In the following fields, enter these details:
Log File: logs/LoginTrace.log
Number of Log Files: 10
Log File Threshold: '.'yyyy-MM-dd-HH-mm

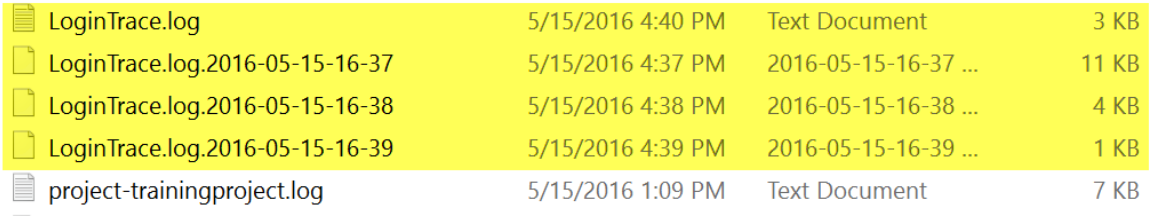


If you noticed, you had provided **Log File Threshold** as: **'.'yyyy-MM-dd-HH-mm**
That means, if there is any activity on **org.apache.sling.auth.core.impl.SlingAuthenticator**, then those will be

captured and saved to a new file every minute. Also, Adobe Experience Manager will keep the latest ten files and remove the older ones.

Click **Save**.

To test if the logging writer is creating the log file every minute, log in to Adobe Experience Manager and logout every minute and observe the **crx-quickstart\logs** directory (if you need help finding where the directory is, you can find it here: **C:\adobe\AEM\author\crx-quickstart\logs)**.



Observe the file with the name **LoginTrace.log.2015-12-04-19-47.** It has the date, hour, and minute details as per the Logging writer configuration.

Scenario Conclusion

You have created a **logging logger** to capture the events in **LoginTrace.log** and then configured a logging writer to write the log on a rotation basis. The Logging writer writes the logs every minute as per the configuration. You can modify this frequency based on your requirement.

CHAPTER TEN: DEVELOPING AND EXTENDING WORKFLOWS

Overview

This chapter deep dives into the workflow console available in Adobe Experience Manager. The hands-on exercises include step-by-step instructions on creating your own process steps that you can insert into your custom workflow.

Objectives

By the end of this chapter, you will be able to:

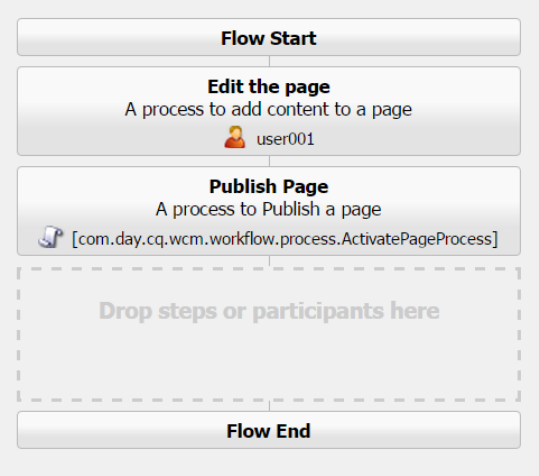
- use existing workflows.
- create your own custom workflows.
- create custom process steps using Java class

Introducing Workflows

In Adobe Experience Manager, workflows are used to automate processes and activities. They are a set of steps performed in a specific order. Each step performs a distinct activity such as activating a page or sending an email message. Workflows can interact with assets in the repository, user accounts, and services. You can pass data from one step to another, invoke workflows automatically, and create custom workflows.

An example of a workflow in Adobe Experience Manager is publishing a page. An editor needs to review a page then site administrator needs to activate it before it can be published. In Adobe Experience Manager, you can create a workflow that automates this process and notifies each of the participants when it is time to perform their assigned tasks. These workflows are specific to your business organization.

The following is a sample workflow used to edit and publish a page in Adobe Experience Manager.



Understanding the Workflow Objects

A workflow is associated with the following objects:

- **Model:** A model consists of nodes and transitions. The transitions connect the nodes and define the flow. The model always has a start node and an end node. Workflow models are versioned. Running workflow instances keep the initial workflow model version that is set when the workflow is started.
- **Steps:** Workflow models consist of a series of steps of various types, which can be extended with scripts to provide the functionality and control you require.
- **Transition:** A transition defines the link between two consecutive steps. You can apply rules to a transition.
- **WorkItem:** This is an object that represents a task or action in the workflow system at runtime. A workflow instance can have more than one WorkItems at the same time.
- **Payload:** It is an entity upon which a workflow instance acts. For example, a page in Adobe Experience Manager could be passed from step-to-step as a payload, which could be either as a JCR node, a JCR path, or an identifier.
- **Lifecycle:** The lifecycle of a workflow begins when the workflow is started, and ends when the end node is processed. You can apply the following actions on a workflow: terminate, suspend, resume, and restart. Completed and terminated instances are archived.

- **Inbox:** Logged-in users have their own workflow inbox in which the assigned WorkItems are accessible. The WorkItems are assigned either to a user itself or to the group to which the user belongs.

Executing an Existing Workflow

Adobe Experience Manager comes with a set of predefined workflows that perform common actions. You can customize those workflows if the built-in steps do not include all the necessary tasks. You can execute a workflow from any of the following places:

- Context menu
- Workflow console
- Site Admin (classic UI)
- Sidekick (classic UI)

Depending on the type of workflow, the payload goes through a series of steps until its completion. Workflow launchers let you automatically invoke a workflow based on certain conditions, such as page modification. You can set the Workflow Launcher configuration to start a workflow when a specified node selection or nodes of a specified type are created, modified, or deleted.

Perform Task – Execute a workflow from the workflow console, from the Lab Activity section.

Defining Workflow Models

Based on your business requirements, there are two actions that you can perform with respect to workflows: modifying an existing workflow, or creating a new one. You can use the Workflow Console to manage workflow models and launchers. A workflow model includes a Flow Start and a Flow End step. These steps indicate the beginning and end of the workflow. All other steps are contained within these two steps.

There are many steps available for a workflow. Two of the most common steps used in workflows are:

- Participant step: Requires manual intervention by a person to advance the workflow.
- Process step: Automatic actions that are executed by the system if specific conditions are met.

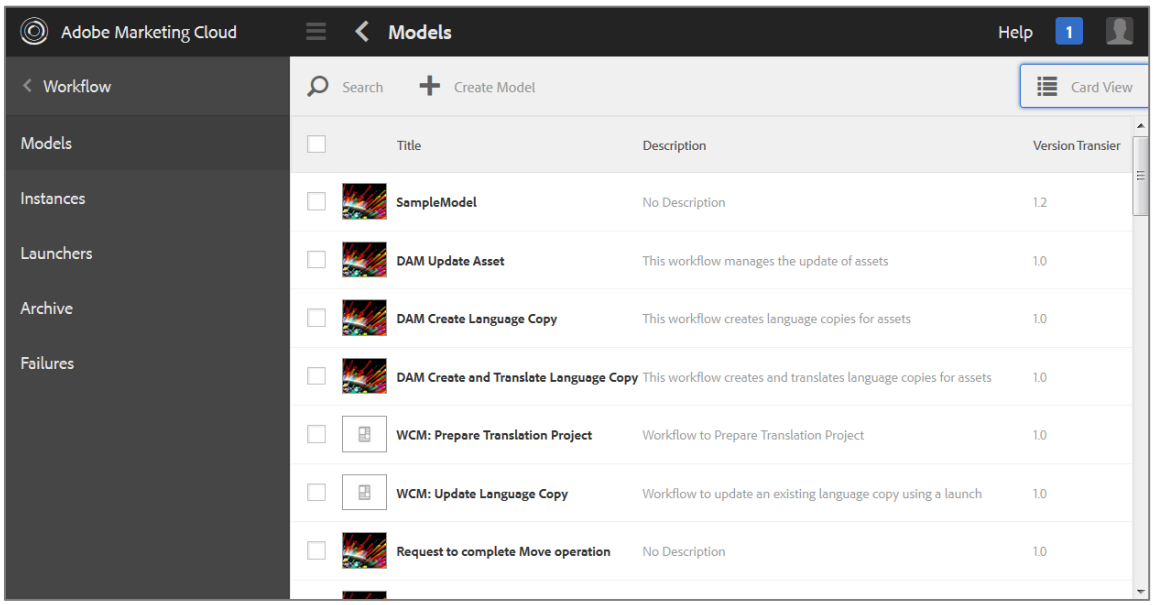
Every new model created includes a sample participant step, which you can either edit or remove. You can add and configure additional steps as required.

The Workflow Console

The Workflow Console is a centralized location for managing workflows. It has the following options:

- **Models:** Enables you to create, edit, or delete workflow models.
- **Instances:** Displays the details of workflow instances that are currently active. These instances are version-dependent.
- **Archive:** Provides access to the details of terminated workflow instances.
- **Launcher:** Allows you to define a workflow to be launched if a specific node was updated.

- **Failures:** Lets you monitor and manage failed workflow instances.



Understanding Workflow Steps

Before we start creating workflows, let us look at what a workflow step is. As mentioned earlier, a workflow consists of one or more steps. Each step can contain any number of actions and associated conditions. For example, a step in a publish workflow may involve approval from an editor. Some steps may require manual intervention, while others may be automatic.

In Adobe Experience Manager, there are a number of steps available for workflows such as Participant, Process, Create Task, Delete Node, Dialog Participant, Dynamic Participant, Form Participant, and many more. Two of the most commonly used workflow steps are Participant and Process.

Participant Step

- Enables you to assign a step to a user or a group of users. If the workflow is assigned to just one user, then that user needs to complete that task before the workflow can proceed to the next step. If the workflow is assigned to a group of users, then all those users need to complete the step.
- You can notify participants of their required action through email. Also, if configured, the participants will receive an email notification when the workflow is completed, or if the workflow is terminated.
- You can configure timeouts and timeout handlers for this step. Timeout is the period after which the step will be timed out. You can select between Off, Immediate, and, if you want to specify specific blocks of time: 1h, 6h, 12h, and 24h. The timeout handler controls the workflow when the step times out.

Process Step

- Executes an ECMA script or calls an OSGi service to automate the process.
- Offers built-in processes that you can use:
  - **Workflow control processes:** Control the behavior of the workflow, and do not perform any action on content.
  - **Basic processes:** Deleting a node, or logging a debug message.

- **WCM processes:** WCM-related tasks such as activating a page, or confirming registration.
- **Versioning processes:** Version-related tasks such as creating versions of the payload.
- **DAM processes:** DAM-related tasks such as creating thumbnails, creating sub-assets, and extracting metadata.
- **Collaboration Processes:** Related to the collaboration features of Adobe Experience Manager, such as that of social communities.

Developing Custom Steps

You can extend workflow steps with scripts to provide more functionality and control. You can create customized process steps using the following methods:

- Java class bundles: Create a bundle with the Java class, and then deploy the bundle into the OSGi container using the Web Console.
- ECMA scripts: Scripts are located in the JCR repository under `etc/workflows`, and they are executed from there. To use a custom script, create a new script with the extension `.ecma` under the same folder. The script will then show up in the process list for a process step.

An exercise later in the module shows how you can create a custom process step using Java, and how to include that step in a customized workflow.

Creating a Workflow

Now that you have learned about the basic workflow steps, and how to create custom process steps, you can now proceed to the creation of the actual workflow.

To create a workflow:

1. Open the Workflow Console, and create a new model.
2. Double-click the newly created model, and modify the steps by clicking and dragging the required workflow steps from the sidekick to the workflow.
3. Edit the properties of the steps.
4. Save the workflow.

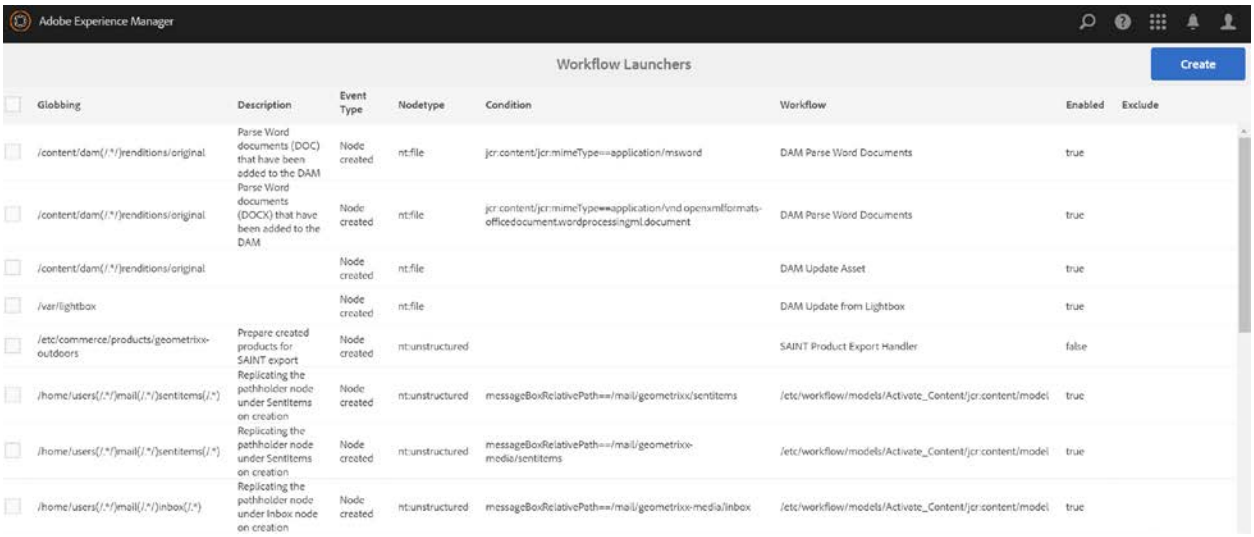
You can execute this workflow either manually, or by configuring a launcher. We talk about launcher later in the module.

Perform Task –Implement a process step in an existing workflow model, from the Lab Activity section.

Using the Workflow Launcher

The Workflow launcher enables you to invoke a workflow based on certain predefined conditions. These conditions are based on changes to the content located in the JCR. For example, when a page is modified, it can trigger a workflow.

You can configure the workflow launchers through the Workflow console (http://localhost:4502/libs/cq/workflow/admin/console/content/launchers.html).



For example, to publish a page after it is modified, you would use the following values for the properties:

- **Event type**: `modified:` Type of event that triggers the workflow.
- **Node type**: `nt:unstructured:` Type of node that is affected by the workflow.
- **Path**: `/content/Geometrixx:` Property that indicates the path of the node.
- **Workflow**: `PublishPage:` Property that indicates the workflow to be executed when the event occurs.
- **Activate**: `Enable`: Property that controls whether the launcher should be activated.
- **Run mode (s)**: `Author:` Property that indicates the type of server that the launcher applies to.

Difference in using Workflow Launchers and JCR Observations or Sling eventing:

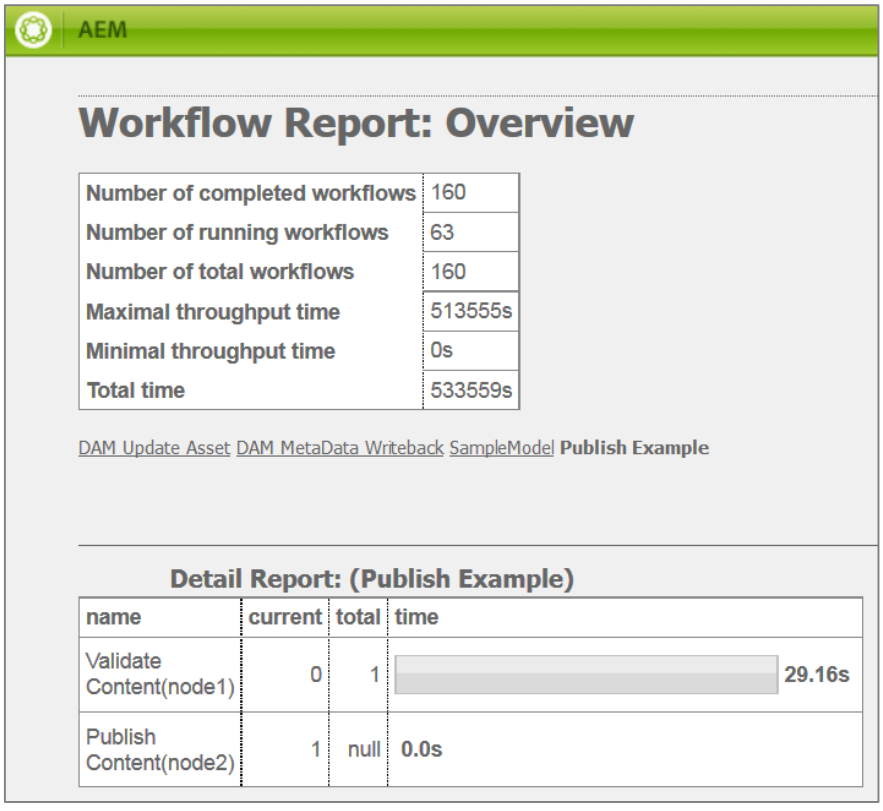
Workflow Launcher	JCR Observation / Sling Eventing
Has a UI	Does not have a UI
Easy to start or stop	Requires the use of Web Console to stop the listener component
Workflow model can be changed dynamically	Requires programmatic or configuration changes
Involves more overhead, and is ideal to use if there are only moderate amounts of event expected	Involves less overhead, and can handle more frequent events
It is cluster-aware, and runs only on the cluster master	Sling eventing is not cluster-aware

Monitoring Performance of Workflows

You can monitor the performance of workflows through its reporting interface, at:
http://localhost:4502/etc/reports/workflowreport.html

This interface provides the following information:

- Details on number of workflows and its duration
- Links for various workflows, and a breakdown of its steps



Chapter 10 Lab Activity

Scenario

You need to ensure that any new page created undergoes an activation process.

Challenge

You need to understand how to automatically trigger processes, as well as how to do them manually.

Overview

Adobe Experience Manager supports multiple ways to implement a process step. You can create your own workflow model as per your requirement.

You have custom or out-of-the-box workflows available in Adobe Experience Manager and they can be executed manually or automatically. To execute a workflow automatically, you should create a launcher that will trigger the workflow. You will manually start a workflow from the workflow console and implement a process step in an existing workflow.

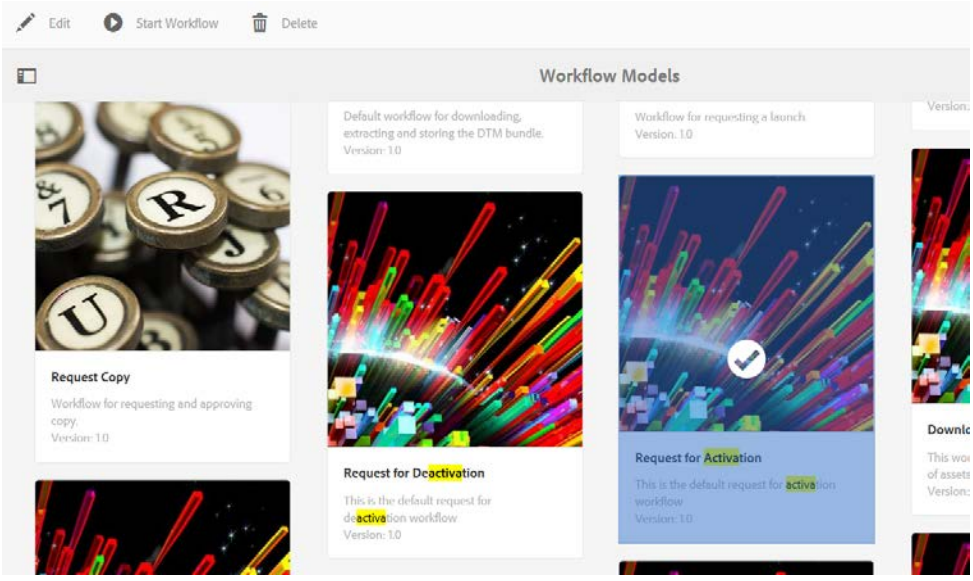
Pre-requisites

Running Author and Publish instances.

Steps

1. Task - Execute a workflow from workflow console

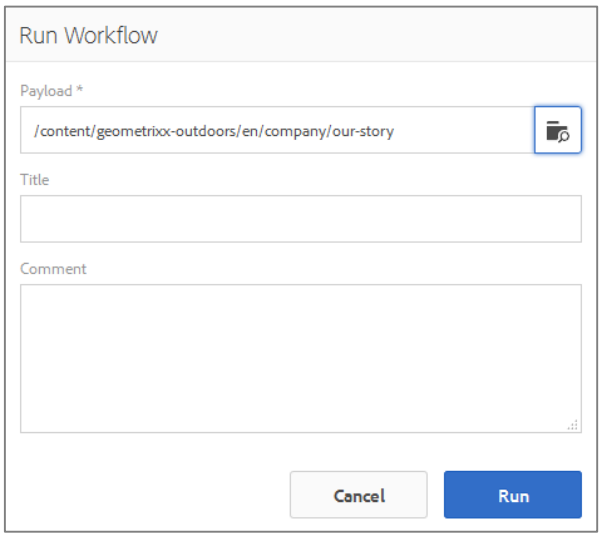
    5. In the Author instance, navigate to **Tools** > **Workflow** > **Models**, and click the checkmark on the **Request for Activation** workflow/card (you may want to search for the word "activation" to help you locate it). When you select it, a blue highlight appears (as shown).



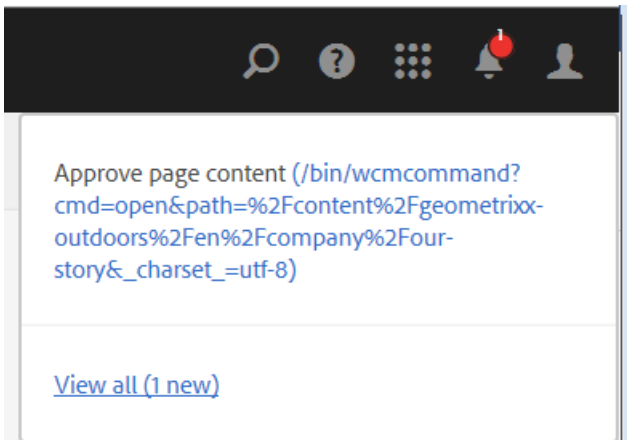
    6. Click **Start Workflow** in the top left corner to open the Run Workflow page.

    7. The Run Workflow page will ask for details such as Payload, Title, and Comment. Enter the page location: **/content/geometrixx-outdoors/en/company/our-story**) in the mandatory Payload field (as shown in the

following screenshot) and then click **Run**. It may take a few seconds, but you will see a "green form has been submitted successfully box" notification indicating the process has begun.



8. The workflow process will begin. A notification will appear in the upper-right to approve page content:



NOTE: The page will not be published until it is approved. Recall the process is two-fold: 1. Edit an initial version of the page, 2. Publish the page. Each of these steps requires approval.

2. Task - Implement a process step in an existing workflow model

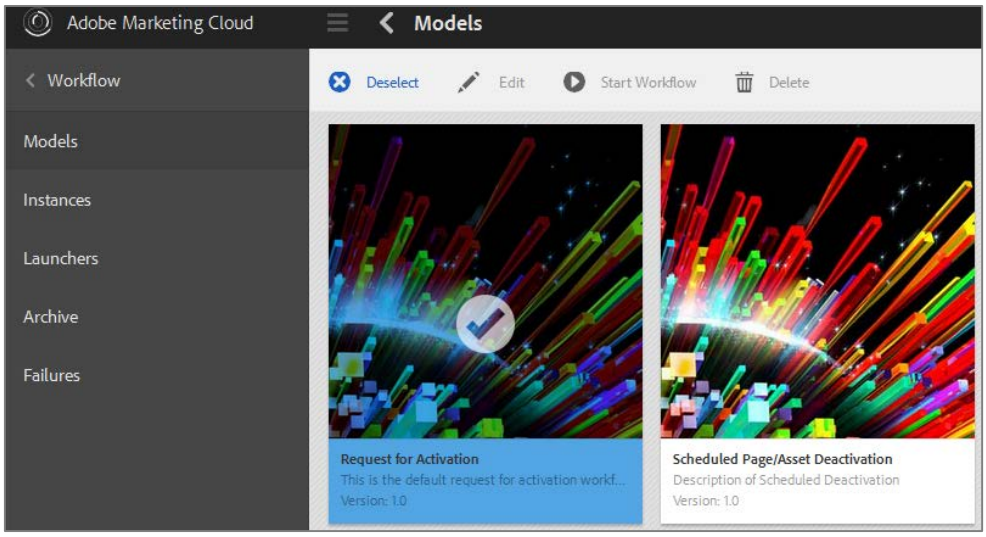
9. Open Eclipse. Use the training project and create new **MyProcess.java** class file under **training.core** > **src/main/java** > **com.adobe.training.core** with the following code:

```
package com.adobe.training.core;
import com.day.cq.workflow.WorkflowException;
import com.day.cq.workflow.WorkflowSession;
import com.day.cq.workflow.exec.WorkItem;
import com.day.cq.workflow.exec.WorkflowData;
import com.day.cq.workflow.exec.WorkflowProcess;
import com.day.cq.workflow.metadata.MetaDataMap;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Properties;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Service;
import org.osgi.framework.Constants;
import javax.jcr.Node;
import javax.jcr.RepositoryException;
@Component
@Service
@Properties({ @Property(name = Constants.SERVICE_DESCRIPTION, value = "A sample
    workflow process implementation."),
        @Property(name = Constants.SERVICE_VENDOR, value = "Adobe"),
        @Property(name = "process.label", value = "My Sample Workflow Process") })
public class MyProcess implements WorkflowProcess {
    private static final String TYPE_JCR_PATH = "JCR_PATH";
    public void execute(WorkItem item, WorkflowSession session, MetaDataMap args)
    throws WorkflowException {
        WorkflowData workflowData = item.getWorkflowData();
        if (workflowData.getPayloadType().equals(TYPE_JCR_PATH)) {
            String path = workflowData.getPayload().toString() + "/jcr:content";
            try {
                Node node = (Node) session.getSession().getItem(path);
                if (node != null) {
                    node.setProperty("approved", readArgument(args));
                    session.getSession().save();
                }
            } catch (RepositoryException e) {
                throw new WorkflowException(e.getMessage(), e);
            }
        }
    }
        }
        private boolean readArgument(MetaDataMap args) {
            String argument = args.get("PROCESS_ARGS", "false");
            return argument.equalsIgnoreCase("true");
```

10. This will create an OSGI service, which will be used as the workflow process step. The service adds the property **approved** to the page content node when the payload is a page.

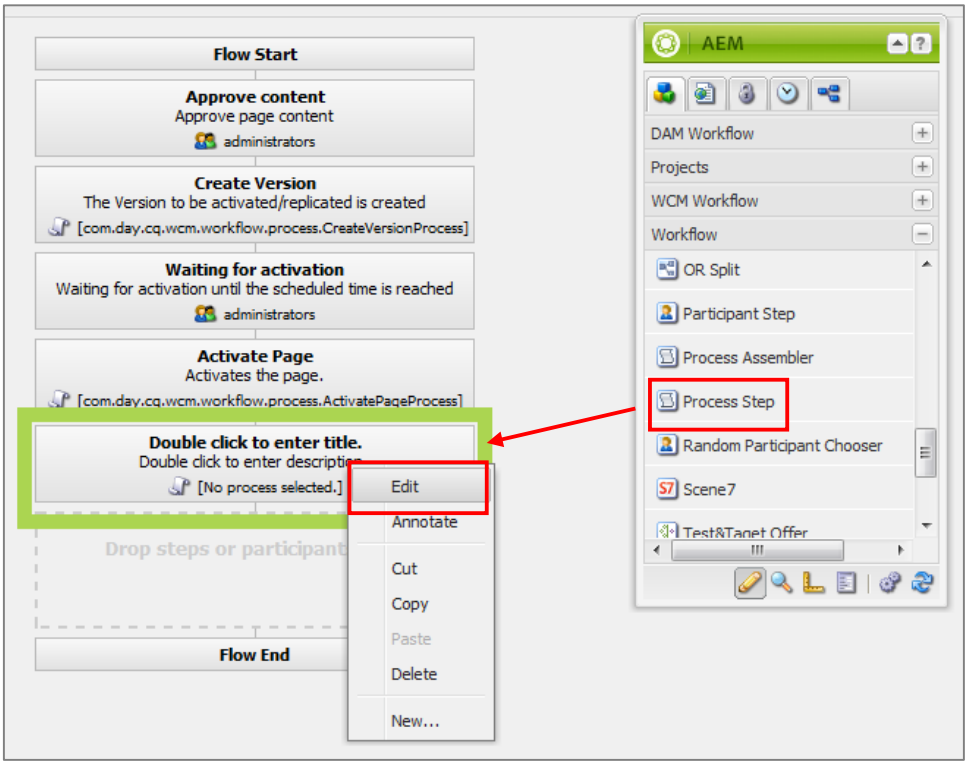
11. Navigate to **Tools** > **Workflow** > **Models**, and select **Request for Activation** workflow.

12. Click the checkmark then click **Edit**.

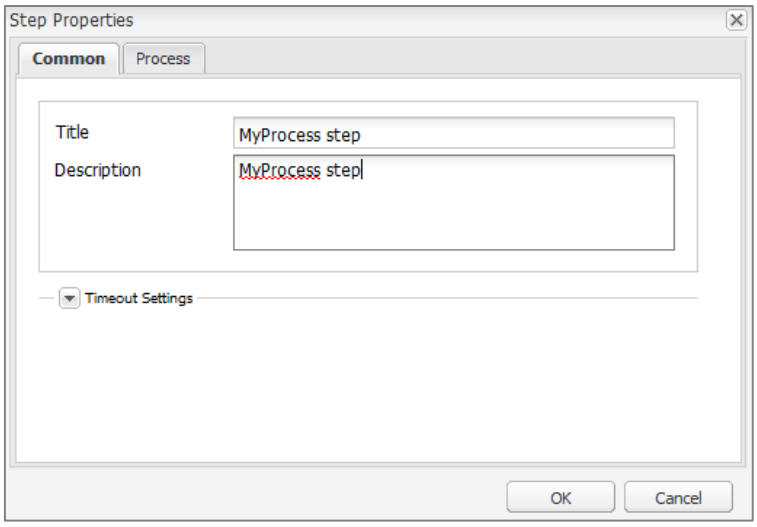


13. In the editor, the workflow steps will be shown as per the default configuration. Drag and drop **Process Step** from the **Workflow** section (make sure to click + next to Workflow; it is very easy to miss this – clicking the + will expand the workflows available to you) to the area with the dotted line "Drop steps or participants here"

14. Right-click and **Edit** the **Process Step**.

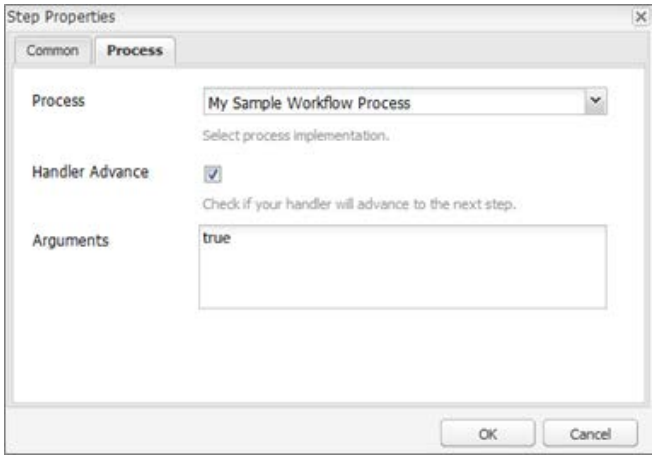


15. Provide the step properties: **Title** and **Description**.



16. Click the **Process** tab and select **My Sample Workflow Process** from the **Process** drop-down. Note that this process (OSGI Service) was created when you added the **MyProcess.java** class.

17. Select **Handler Advance** in order to tell the workflow to advance to the next step. Set the Argument value to **true**.



18. Click **OK.**

19. Click **Save** in the toolbar (underneath Request for Activation at the top of the page) to save the workflow.

20. Now return to the **Workflow Models** card view, and note that the workflow model **Request for Activation** has a **New** blue label in the upper-right corner of the card. Select this workflow model, and click **Start Workflow**.

21. Select a page for payload field then click **Run**. You will notice that a new node property is created by the workflow.

22. Click on **Notifications** (Bell icon) in the top right corner, and complete the participant steps for the workflow by selecting it and then clicking on Complete. Repeat this for both participant steps.

23. Use CRXDE Lite to check properties on the page used as payload. You will notice that a new node property **approved** is created by the workflow and set to **true** by the argument provided.

Scenario Conclusion

You have modified a workflow and added a new process step using OSGI service.

CHAPTER ELEVEN: BUILDING INTEGRATION POINTS

Overview

This chapter describes how you can import data from a third-party source by polling and using Workflow Launchers. You will also learn how to integrate your repository with other databases using JDBC.

Objectives

By the end of this chapter, you will be able to:

- Import data from external sources
- Use the Workflow Launcher to monitor polling events
- Integrate your repository with Databases using JDBC
- Describe the usage of OAuth Client Access
- implement periodic importers.

Ingesting Data from External Sources

The feed importer is a framework to repeatedly import content from external sources into your repository. The idea behind a feed importer is to poll a remote resource at a specified interval, parse it, and create nodes in the content repository that represent the content of the remote resource. In Adobe Experience Manager Sites (out-of-the-box), the feed importer is used for the following:

- In blogs to support the auto-blogging feature, which automatically creates blog posts from an external RSS or Atom feed.
- In calendars for iCalendar subscriptions, which automatically creates calendar events from an external ICS file or subscribes to/from other calendars.

The feed importer is found in the WCM Administrative interface under **Tools** > **Importers** > **Feed Importer**. Double-clicking the Feed Importer node opens a page that allows configuration of standard feed importers for RSS, Atom, Calendar, IMAP, and POP3.

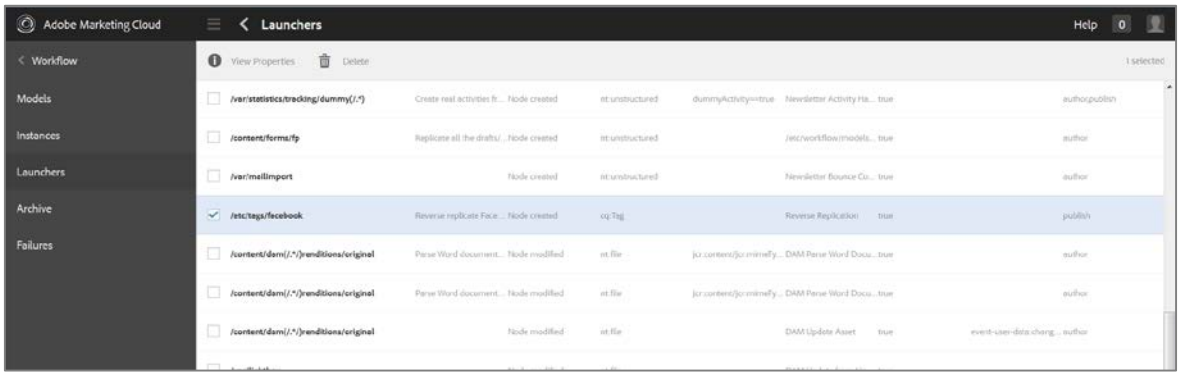
To implement your own feed importer, use the interface: `com.day.cq.pollingimporter.Importer`

Perform Task – Create a Polling Importer, from the Lab Activity section.

Using Workflow Launcher to Monitor Polling Events

The workflow launcher is used to define a workflow that can be launched on a specific event such as the creation, modification, or deletion of a node. You can access the workflow launchers at http://localhost:4502/libs/cq/workflow/admin/console/content/launchers.html

Look at the following highlighted launcher. The launcher is designed to run when a Facebook tag is created in the publish instance. The launcher would then execute steps to reverse replicate the tags to the author instance.



Using the same concept, you can create a launcher to monitor the changes in the ADBE stock (created in Task 1), and record events for specific conditions in the log file.

Steps to create the launcher:

Create and consume an OSGi service using Eclipse.

Create a workflow in Adobe Experience Manager.

Create a workflow launcher for the above workflow.

Once you complete these tasks, you can check the error.log file to check the recorded entry for the ADBE stock price.

Perform Task – Use the Workflow launcher to monitor polling events, from the Lab Activity section.

Integrating with Databases Using JDBC

Adobe Experience Manager lets you integrate your repository with external databases such as SQL, using connectors. The primary role of these connectors is to provide JCR access to existing data in an external database. The connector maps tables to nodes, with sub-nodes representing table's rows and properties representing the columns.

Adobe's JCR Connector family enables enterprises to access and manage all organizational content through one standardized API (JCR). This technology allows content access, synchronization and consolidation, leveraging the future-proof Content Repository API for Java Technology (JCR), even if the content resides in data stores that do not provide a JCR compliant API.

The basic steps to create a JDBC connection are:

Create or obtain an OSGi bundle that exports the JDBC driver package.

Configure a JDBC data source pool provider.

Obtain a data source object and create the connection in your code.

Sample code for JDBC

```
import java.sql.*;
......
Class.forName("org.h2.Driver");
Connection con = DriverManager.getConnection("jdbc:h2:tcp://" + host + "/~/" + db, user, pwd);
......
Statement stm = con.createStatement();
stm.execute(statement);
```

There are multiple methods to install the JDBC driver.

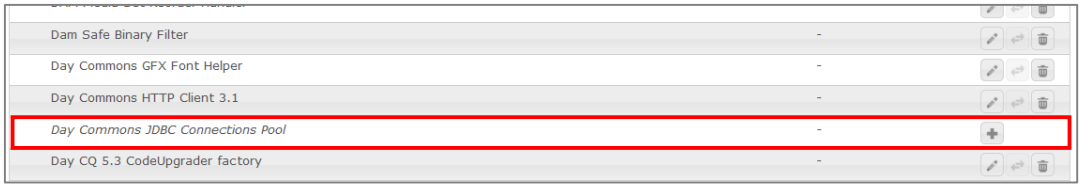
Install database OSGi bundle, if provided by the vendor

Include the DB driver JAR into bundle libs folder

Install the driver as an OSGi Extension bundle

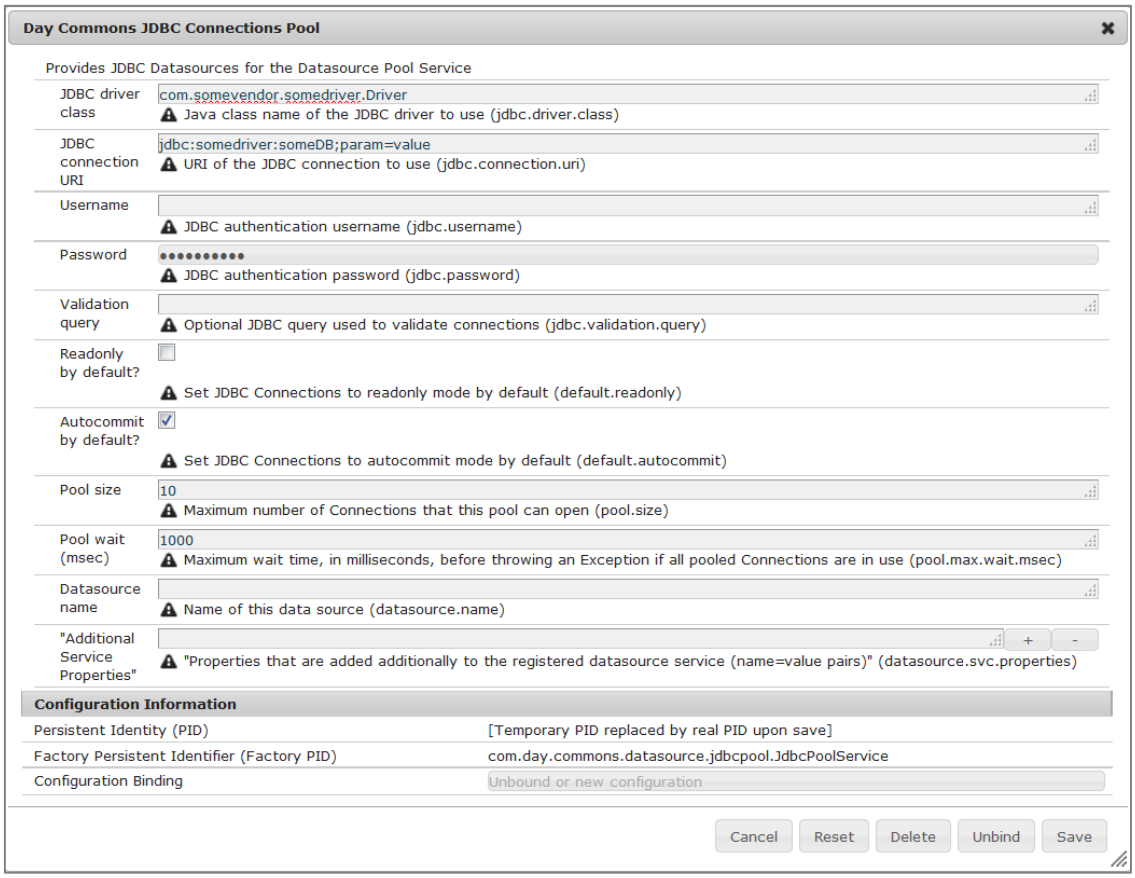
Add the JAR to the shared classpath of your application server and use `org.osgi.framework.bootdelegation` property in `sling.properties` to load the driver class.

You can configure the settings for the JDBC pool through the Web Console, under the **Configurations** section of OSGi tab. Search for **Day Commons JDBS Connections Pool**.



NOTE: Instead of using the Web Console, you can also create a configuration node in the repository using CRXDE Lite.

You can create a new connection by providing the necessary details such as credentials, data source settings, and so on.



NOTE:

- If the JDBC driver for your particular database is available as an OSGi bundle, you can deploy it directly using the Felix OSGi management console (Web Console).
- If the JDBC driver is only available as a standard jar file, then you must first convert it into an OSGi bundle before deploying it.

Best Practice:

- Include all database code into OSGi Bundle, and not as a JSP.
- Install JDBC driver as an OSGi bundle.
- Use Connection Pool from the Web Console.
- Set up OSGi configuration in Repository.

Working With OAuth Client Access

OAuth is an open standard for authorization that enables client applications to access server resources on behalf of a specific Resource Owner. OAuth enables you to notify a resource provider that the resource owner grants permission to a third-party access to their information. For example, if you want to import your contacts into Facebook from your Gmail account. Instead of manually creating your friends' list and typing in their email IDs, Facebook would use the OAuth protocol that requests Gmail to provide limited access. This would import all the contacts from Gmail without any breach in account security.

OAuth allows:

Different access levels: Read-only VS read-write. This allows you to grant access to your user list or a bi-directional access to automatically synchronize your new Facebook friends to your Gmail contacts.

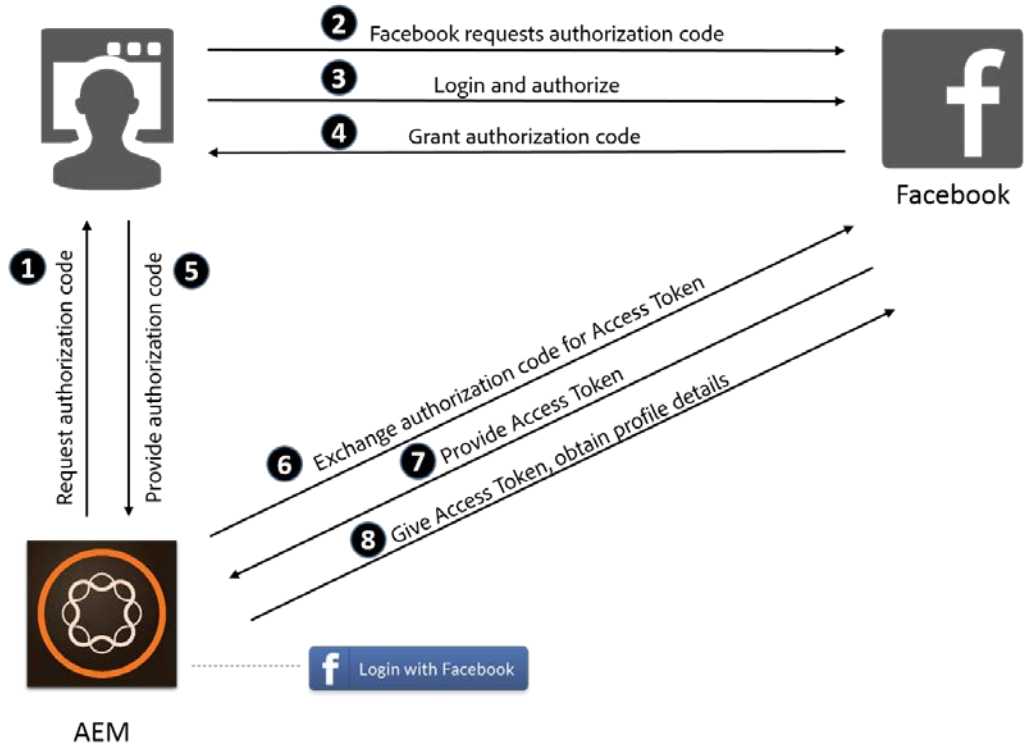
Access granularity: You can decide to grant access to only your contact information or to your entire list of friends, calendar, and what not.

Management of access from the resource provider's application. If the third-party application does not provide mechanism for cancelling access, you would be stuck with them having access to your information. With OAuth, there is provision for revoking access at any time.

Adobe Experience Manager provides two services—authentication service and client service. For this training, we will concentrate on the client service.

Using Adobe Experience Manager as a Client

Adobe Experience Manager provides OAuth 2 services wherein registered applications can connect to Adobe Experience Manager using third party login credentials. Let us see how this works. The following diagram shows how to log in to an Adobe Experience Manager site using a Facebook account.



When a user chooses to log in to Adobe Experience Manager using a Facebook account, a request is sent to Facebook for an authentication code. The user would then login to the Facebook account, credentials are authorized, and an authentication code is returned to the Adobe Experience Manager client. The Adobe Experience Manager client can then contact Facebook directly with the authentication code, exchange it for an access token and obtain the profile details. The access token obtained from Facebook would expire within a specified time frame.

Configuring the Client

The Adobe Granite OAuth Authentication Handler implements the `AuthenticationHandler` interface to delegate authentication to third parties supporting the OAuth protocol. Examples of OAuth Providers are Facebook and Twitter.

By default, the OAuth Authentication Handler is disabled. You need to configure it for it to operate. You can do so in the following two steps:

Configure the CRX repository to allow for trusted credentials authentication.

Configure the OAuth Authentication Handler.

Implementing a Periodic Importer

A feed importer enables you to import content from external sources into your repository at periodic intervals. The importer polls the remote resource, parses it, and then creates similar nodes in the content repository. You can use feed importers in the following situations:

- Blogging—to automatically create blog posts from an external RSS feed.
- Calendar—create calendar events from external ICS file, or other calendar subscriptions.

You can find feed importers in the WCM administrative interface under **Tools** > **Importers** > **Feed Importer**. To implement your own feed importer, use the interface: `com.day.cq.polling.importer.Importer`.

Chapter Eleven Lab Activity

Scenario

You need to obtain the Adobe (ADBE) share price and display it on your page.

Challenge

Selecting the right importer, and making use of the share price as per your requirement.

Overview

You have to create a stock ticker for Adobe system share price and display it on your page. You will achieve this by using polling importer to import configurable stock data. You will use the stock price from http://finance.yahoo.com/d/quotes.csv?s=ADBE&f=snd1l1yr (a csv file), and then save the latest stock data in the repository. The repository modification will trigger a workflow, and this will check for and report if the stock level reaches a certain value.

Pre-requisites

You need the project—training project from the USB contents, where you can write a new java class file to implement the custom importer. You also need an Adobe Experience Manager Author instance to test the importer.

Steps

1. Task - Create a Polling Importer

   Open Eclipse. Use the training project and create new **StockDataImporter.java** class file under **training.core** > **src/main/java** > **com.adobe.training.core** with the following code:

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


import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.MalformedURLException;

import java.net.URL;

import java.util.Arrays;

import java.util.Calendar;

import java.util.regex.Pattern;


import javax.jcr.Node;

import javax.jcr.RepositoryException;

import javax.jcr.Session;
```

```
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.jcr.api.SlingRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.day.cq.commons.jcr.JcrUtil;
import com.day.cq.polling.importer.ImportException;
import com.day.cq.polling.importer.Importer;

@Service(value = Importer.class)
@Component
@Property(name = "importer.scheme", value = "stock", propertyPrivate = false)
public class StockDataImporter implements Importer {

        private final String SOURCE_URL = "http://finance.yahoo.com/d/quotes.csv?f=snd1l1yr&s=";

        private final Logger LOGGER = LoggerFactory.getLogger(StockDataImporter.class);

        @Reference
        private SlingRepository repo;

        Session adminSession = null;

        @Override
        public void importData(final String scheme, final String dataSource, final Resource resource)
                throws ImportException {
```

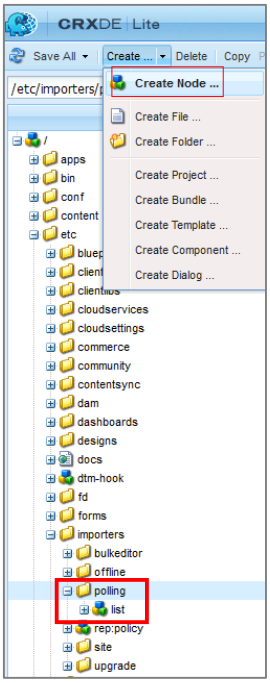
```
            try {
                // dataSource will be interpreted as the stock symbol
                URL sourceUrl       = new URL(SOURCE_URL + dataSource);
                BufferedReader in = new BufferedReader(new InputStreamReader(sourceUrl.openStream()));
                String readLine = in.readLine(); // expecting only one line
                String lastTrade = Arrays.asList(Pattern.compile(",").split(readLine)).get(3);
                LOGGER.info("Last trade for stock symbol {} was {}", dataSource, lastTrade);
                in.close();

                //persist
                writeToRepository(dataSource, lastTrade, resource);
            }
            catch (MalformedURLException e) {
                LOGGER.error("MalformedURLException", e);
            }
            catch (IOException e) {
                LOGGER.error("IOException", e);
            }
            catch (RepositoryException e) {
                LOGGER.error("RepositoryException", e);
            }

    }

    private void writeToRepository(final String stockSymbol, final String lastTrade, final Resource resource) throws
RepositoryException {
            adminSession= repo.loginService("training",null);
            Node parent = resource.adaptTo(Node.class);
            Node stockPageNode = JcrUtil.createPath(parent.getPath() + "/" + stockSymbol, "cq:Page",
                    adminSession);
            Node lastTradeNode = JcrUtil.createPath(stockPageNode.getPath() + "/lastTrade", "nt:unstructured",
```

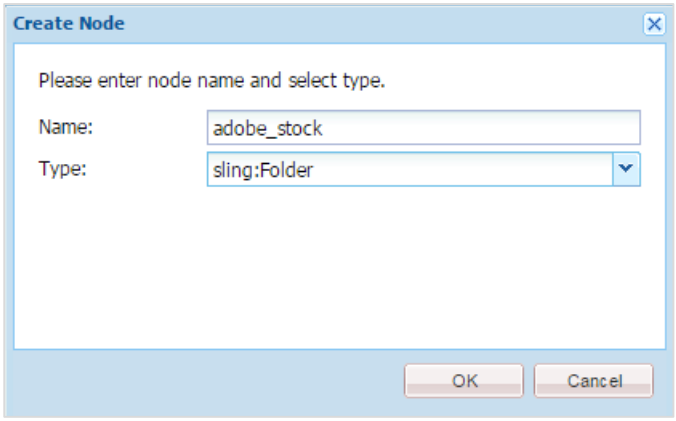
```
                                adminSession);
            lastTradeNode.setProperty("lastTrade", lastTrade);
            lastTradeNode.setProperty("lastUpdate", Calendar.getInstance());
            adminSession.save();
            adminSession.logout();
        }

        @Override
        public void importData(String scheme, String dataSource, Resource target,
                        String login, String password) throws ImportException {
            importData(scheme, dataSource, target);

        }
}
```

Note that we are implementing the interface **com.day.cq.polling.importer.Importer** in the code. You can explore the code and understand that we will be using many variables from JCR. You will create these variables in the JCR repository as a configuration node.

Deploy the project by selecting "**training.core**" project and clicking **Run** > **Run As** > **Maven install**.

Open CRXDE http://localhost:4502/crx/de/index.jsp and navigate to **etc > importers > polling** and create a new node with name **adobe_stock** of type **sling:folder.**





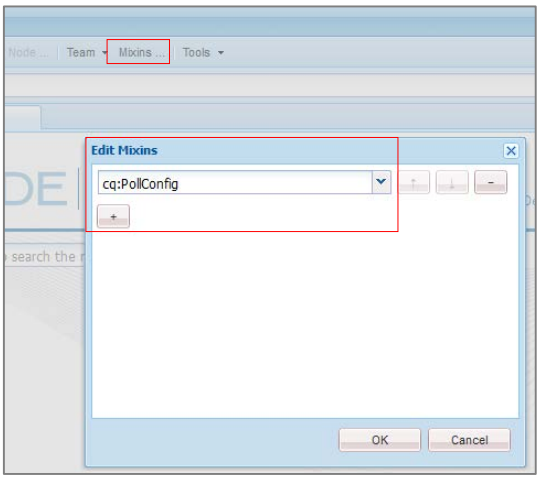
On the **adobe_stock** folder node, add the following properties:

i. **Name**: interval
   Type: Long
   Value: 300

j. **Name**: source
   Type: string
   Value: stock:ADBE

k. **Name**: target
   Type: string
   Value: /content

Click **Mixins** in the toolbar at the top, click **+,** and add **cq:PollConfig** then click **OK.**



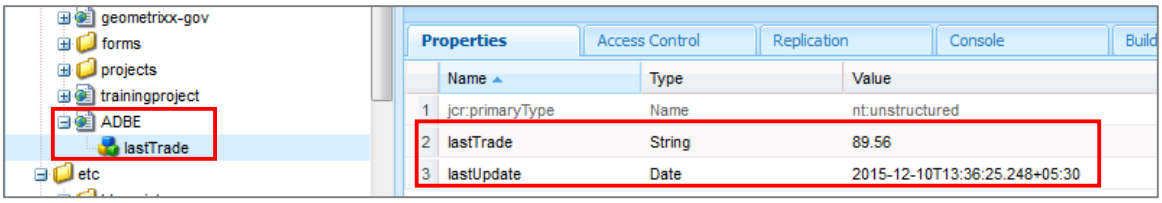
This configuration lets the JCR know this is a Mixins configuration.

Click **Save All** to save the properties created on the **adobe_stock** node.

The configurations we created will provide the information to **StockDataImporter.java** class. As a result, you should see a new node created at the target path **(/content**) provided in the node.

}jcr:content/metadata }jcr:content/metadata }jcr:content/metadata
The type of the ABDE node is **cq:page,** which is coming from the Java code written for the **StockDataImporter.java** class.

Click on **lastTrade** node and see the **lastTrade** and **lastUpdate** values for Adobe stock. This last trade value and time are coming from the source URL: http://finance.yahoo.com/d/quotes.csv?f=snd1l1yr&s



2. Task - Use the Workflow launcher to monitor polling events

Now, you will create a workflow that alerts authors when a stock value exceeds a certain threshold.

Open Eclipse. Use the training project and create new **StockAlertProcess.java** class file under **training.core** > **src/main/java** > **com.adobe.training.core** with following code:

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

import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.regex.Pattern;

import javax.jcr.Node;
import javax.jcr.RepositoryException;
import javax.jcr.Session;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Service;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.day.cq.workflow.WorkflowException;
import com.day.cq.workflow.WorkflowSession;
import com.day.cq.workflow.exec.WorkItem;
import com.day.cq.workflow.exec.WorkflowData;
import com.day.cq.workflow.exec.WorkflowProcess;
import com.day.cq.workflow.metadata.MetaDataMap;

@Service
@Component(metatype = false)
@Property(name = "process.label", value = "Stock Threshold Checker")
public class StockAlertProcess implements WorkflowProcess {

    private static final String PROPERTY_LAST_TRADE = "lastTrade";
    private static final String TYPE_JCR_PATH = "JCR_PATH";
    private static final String TYPE_JCR_UUID = "JCR_UUID";
    private static final Logger LOGGER = LoggerFactory.getLogger(StockAlertProcess.class);

    @Override
    public void execute(WorkItem workItem, WorkflowSession workflowSession, MetaDataMap args)
        throws WorkflowException {
        try {
            // get the node the workflow is acting on
            Session session = workflowSession.getSession();
```

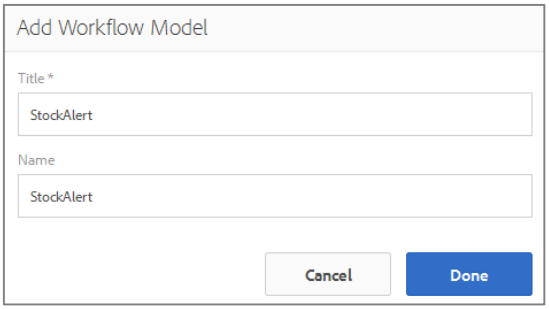
```
            WorkflowData data = workItem.getWorkflowData();
            Node node = null;
            String type = data.getPayloadType();
            if(type.equals(TYPE_JCR_PATH) && data.getPayload() != null) {
                String payloadData = (String) data.getPayload();
                if(session.itemExists(payloadData)) {
                    node = session.getNode(payloadData);
                }
            }
            else if (data.getPayload() != null && type.equals(TYPE_JCR_UUID)) {
                node = session.getNodeByIdentifier((String) data.getPayload());
            }
            LOGGER.info("********running with node {}", node.getPath());
            // parent's is expected to be stock symbol
            String symbol = node.getParent().getName();
            LOGGER.info("********found symbol {}", symbol);
            if (node.hasProperty(PROPERTY_LAST_TRADE)) {
                Double lastTrade = node.getProperty(PROPERTY_LAST_TRADE).getDouble();
                LOGGER.info("********last trade was {}", lastTrade);
                // reading the passed arguments
                Iterator<String> argumentsIterator = Arrays.asList(
                        Pattern.compile("\n").split(args.get("PROCESS_ARGS", ""))).iterator();
                while (argumentsIterator.hasNext()) {
                    List<String> currentArgumentLine = Arrays.asList(
                            Pattern.compile("=").split(argumentsIterator.next()));
                    String currentSymbol = currentArgumentLine.get(0);
                    Double currentLimit = new Double(currentArgumentLine.get(1));
                    if (currentSymbol.equalsIgnoreCase(symbol) && currentLimit < lastTrade) {
                        LOGGER.warn("Stock Alert! {} is over {}", symbol, currentLimit);
                    }
                }
            }
        }
        catch (RepositoryException e) {
            LOGGER.error("RepositoryException", e);
        }
    }
}
```

Deploy the project by selecting the **training.core** project and clicking **Run** > **Run As** > **Maven install**.

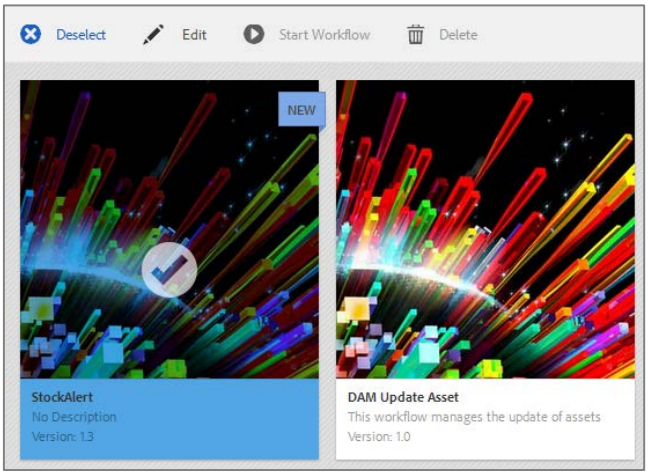
Navigate to **Tools** > **Workflow** > **Models** or open http://localhost:4502/libs/cq/workflow/admin/console/content/models.html/etc/workflow/models then click Create to create a new workflow model with the following details:

- **Title:** StockAlert
- **Name:** StockAlert

Click **Done**.

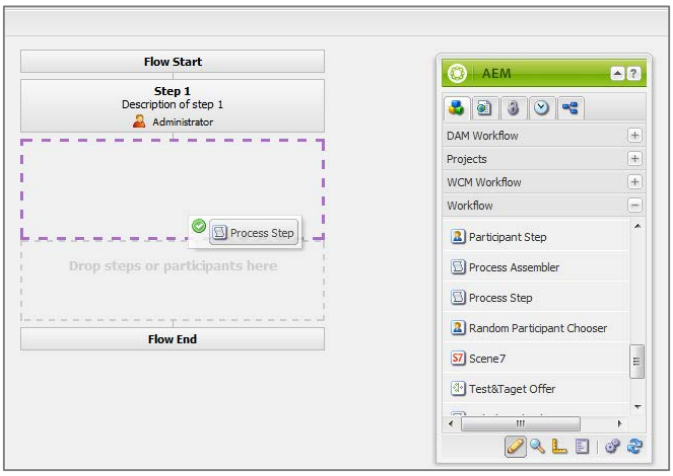


Select the newly created model **StockAlert**, click the checkmark, and then click **Edit.**



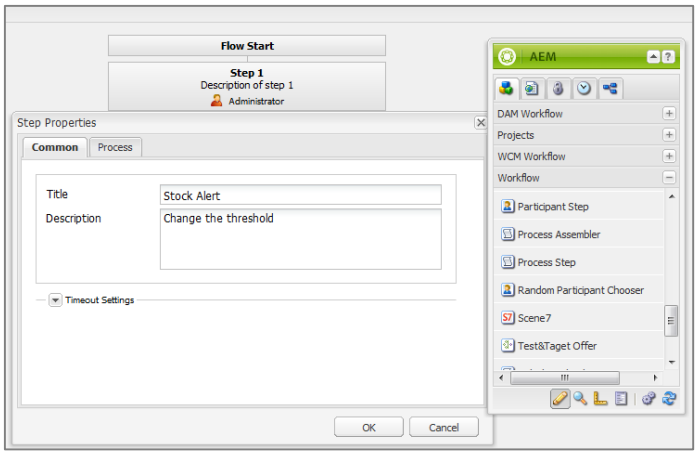
Delete the existing **Participant Step** from this workflow. This is not required as this workflow will be entirely automated.

Add a **Process Step** from **Workflow** section.



Double-click the **Process Step** and enter the following:

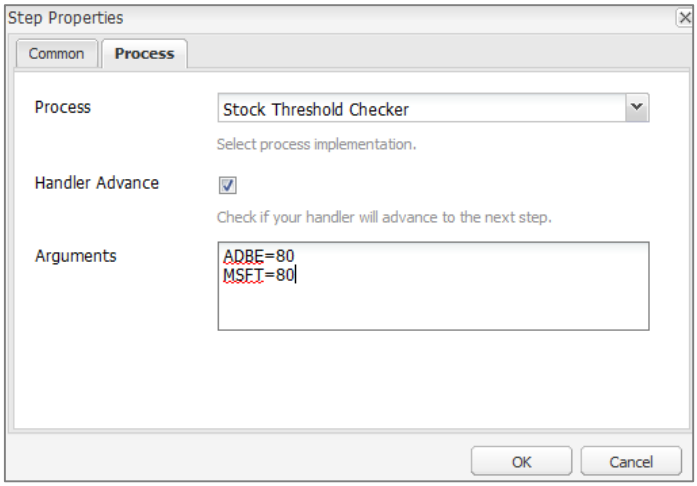
- **Title**: Stock Alert
- **Description**: Change the threshold



Click the **Process** tab and add:

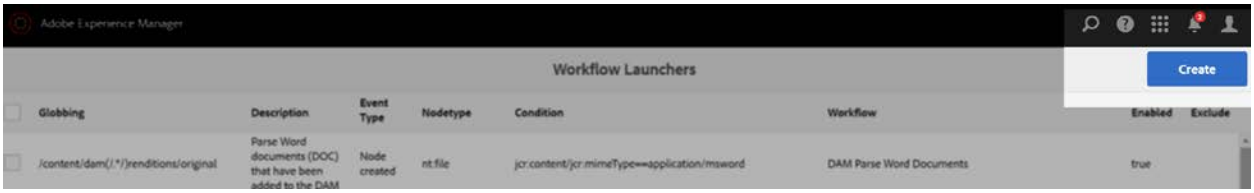
- **Process:** Stock Threshold Checker
- Handler Advance: Checked

- **Arguments:** ADBE=80
  MSFT=80



Click **OK** on the Step Properties dialog box, and then click **Save** (in the top left corner) to save the workflow Model.

Navigate to **Tools** > **Workflow** > **Launchers**, and click **Create** > **Add Launcher.**



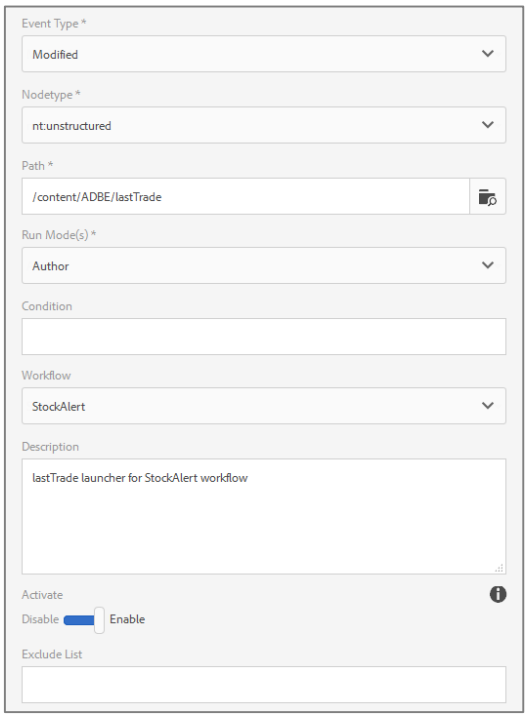
Note: This launcher will start the workflow whenever a **lastTrade** node is modified.

The Create Workflow Launcher page opens.

Enter the following values:

- **Event Type**: Modified
- **Nodetype:** nt:unstructured
- **Path:** /content/ADBE/lastTrade
- Run Mode(s): Author
- **Condition:** leave blank
- **Workflow:** StockAlert
- **Description:** lastTrade launcher for StockAlert workflow
- Activate: Enable

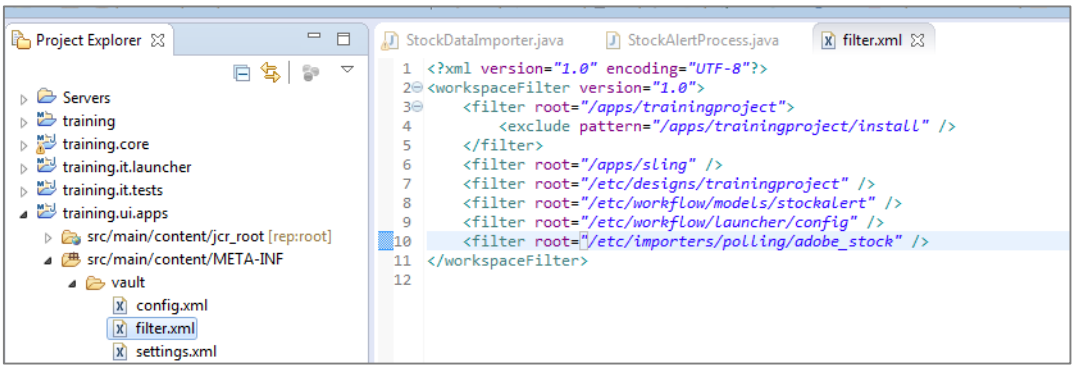
- **Exclude List:** Leave Blank



Click **Create**.

Open Eclipse and update **filter.xml** located at **training.ui.apps/src/main/content/META-INF/vault** with the following code:

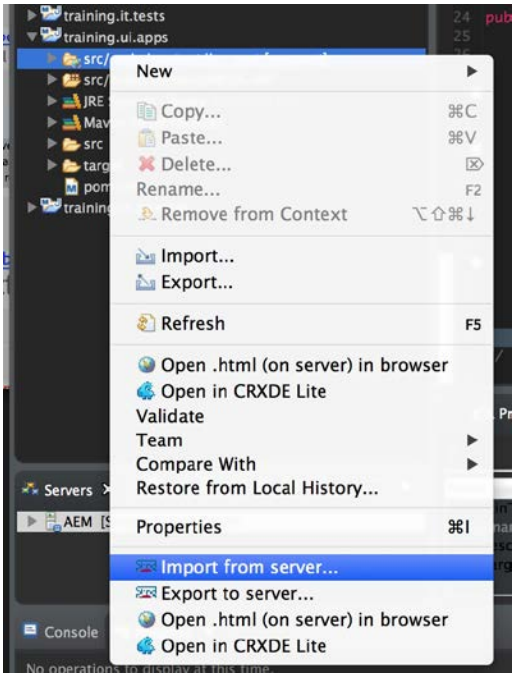
```
<?xml version="1.0" encoding="UTF-8"?>
<workspaceFilter version="1.0">
<filter root="/apps/trainingproject">
<exclude pattern="/apps/trainingproject/install" />
</filter>
<filter root="/apps/sling" />
<filter root="/etc/designs/trainingproject" />
<filter root="/etc/workflow/models/stockalert" />
<filter root="/etc/workflow/launcher/config" />
<filter root="/etc/importers/polling/adobe_stock" />
</workspaceFilter>
```



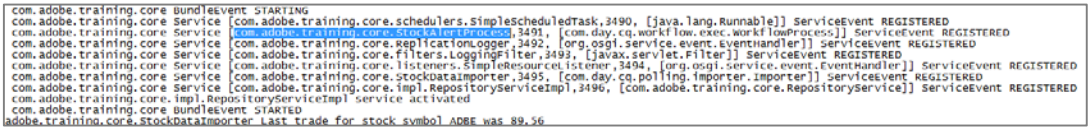
Navigate to **\workspace\training\ui.apps\src\main\content\jcr_root** and update the changes using VLT by running the command as:

```
vlt up
```

Alternatively, via eclipse, within the training.ui.apps submodule, right click on src/main/content/jcr_root, and select "Import From Server". Note instead if you update using the VLT command line tool, you require to update the eclipse project to see the changes reflected here.



You can check the project log file to verify if the workflow is running correctly.



Similarly, you can create more custom importers, launchers, and workflows to achieve custom requirement.

Scenario Conclusion

In this module, you have consumed data from an external source (Yahoo Finance) into the Adobe Experience Manager repository and created an alert whenever the price of the stock crosses the limit price given in the workflow.

CHAPTER TWELVE: DATA MIGRATION

Overview

This chapter explains the various methods available to migrate your data from an existing legacy system to Adobe Experience Manager. The Lab Activity section includes hands-on exercises on performing migration using VLT, Sling POST servlets, and JCR API. The module also shows you how to dynamically create pages and assets. This chapter also includes best practices for migration, and identifies the benefits of various types of migration.

Objectives

By the end of this chapter, you will be able to:

- migrate your data from an existing legacy system to Adobe Experience Manager
- identify the various storage elements in Adobe Experience Manager

Understanding Data Migration

Migration occurs when you have to transfer your content from an existing system (legacy system) to Adobe Experience Manager. A major challenge that you would face is when you have to migrate content from different systems. Combining content from various Content Management Systems (CMS) involves a thorough study of its architecture.

The Migration Process

The following are best practices for migrating data from legacy systems:

1. **Identify the source.**
   Most legacy systems are based on databases that relate to the data being stored. You need to extract this data from the source format into a neutral format. For example, you can extract the data into XML.

2. **Transform the existing content to the target format.**
   The extracted data must be converted into a format (xml format) that is recognizable by Adobe Experience Manager. That is, it should translate itself into nodes and properties as identified by the JCR. You need to use David's model to map data into the JCR repository.

3. **Import into CRX.**
   Import and store the content in CRX using tools such as FileVault or Package Manager.

There are two approaches to data migration:

- One-time migration. For example, when converting a Drupal site to an Adobe Experience Manager site.
- Continuous migration. For example, in ecommerce systems, products, vouchers, payments, and such need to move from the database into Adobe Experience Manager.

There are different types of migration:

- **Automated migration**
  This type of migration is completed automated, with no manual intervention during the migration.
- **Partially automated migration**
  Depending on the size and type of data, there would be instances where some parts of the migration is done manually, the rest done through automation.
- **Manual migration**
  This type of migration is done manually.
- **Migration on request**
  This type of migration is an ad-hoc migration and depends on the size and type of data.

As a final note, whenever you need to migrate content, consider the following:

Migration is not just copying and pasting data into CRX.

The migration plan differs with each project.

Make good use of the xml format.

Identifying the Challenges of Data Migration

A few of the challenges faced during data migration from legacy systems to Adobe Experience Manager are:

- Migrating data from different systems.
- Most legacy systems already have a CMS in place.
- Migration can be time-consuming as legacy systems can lack taxonomy and metadata.

Migrating Data from Legacy Systems

Adobe Experience Manager comes with a set of predefined sample sites that you can use to analyze for its data structure. You can then either make use of the same sites with modifications or recreate the site to suit your requirements.

Consider the following when planning your migration from legacy systems:

- Analyze the source system.
- Define the actual data architecture and all the special considerations for the migration.
- Pre-format data considerations, and make them presentable to the content migrator.

Implementation considerations:

- Define the data manipulation that needs to take place.
- Consider breaking the migration process into sub-migrations.
- Modify the data coming out of the source system based on the business requirements.
- Define the node structure for the new system.
- Perform a test on dummy data before testing on real data. You can use the Sling POST servlet to perform these tests.

There are three basic possibilities to migrate content from legacy systems into Adobe Experience Manager:

- Using VLT
- Using the Sling POST servlet
- Using the JCR API

Using VLT

VLT-based migration is efficient for large quantities of content, but it may not be easy to debug if you get the format wrong.

Basic steps to migrate content using VLT:

Export the content into a VLT serialization. This will create `.content.xml` files containing an xml representation of the content at each node.

Use VLT to import the content into Adobe Experience Manager.

An alternative approach to importing content is to create a content package and import it into Adobe Experience Manager using the Package Manager.

Perform Task – Migrate content from a legacy system using VLT from the Lab Activity section.

Using Sling POST Servlet

The main purpose of using this method is to test the migration and ensure the content structure is accurate before doing the actual migration. To use this method, you must first verify you have curl installed in your system. curl is used to issue an http request, and call the Sling POST servlet to post new content into the repository.

1.1.3 Installing and Configuring Curl

curl is provided on the USB memory stick. You can also download it from the following site: http://curl.haxx.se/download.html. Ensure that you select the correct version of curl for your operating system.

Extract the file to a suitable location. For example, `/usr/bin` on the Mac, and `C:\curl` on Windows.

Ensure that the PATH environment variable is configured to include the path to curl.

Type `curl --version` to test that the installation is working.

1.1.4 Migrating content

Follow these steps to migrate content using the Sling POST servlet:

Export the content from the legacy system to your file system.

Use a shell script to transform the content into curl commands to the Sling POST servlet.

Perform Task – Use Sling POST servlet to test XML, from the Lab Activity section.

Using the JCR API

JCR APIs are used to automate the migration of content from legacy systems to Adobe Experience Manager. This method is efficient and allows the use of Adobe Experience Manager/JCR methods. It is also useful for scripted fixes to the content structure.

Follow these steps to migrate content using the JCR API:

Export the content in an arbitrary structure, into your file system.

Run a script (JSP) that accesses the file system and creates corresponding content in the repository.

Perform Task – Use the JCR API to migrate data, from the Lab Activity section.

Using Packages for Data Migration

The Package Manager is a packaging tool that is available with Adobe Experience Manager, which you can use to create packages to export content into external systems, or to import content from other systems. In this module, you will use the Package Manager to migrate data from your legacy systems into the JCR.

When to Use Packages for Migration

You can use content packages to migrate large amounts of data (Gigabytes or Terabytes) in a short period of time. However, a major requirement for this process is that the structure of the data being imported must be in the right format. It must match the JCR repository, in its nodes and properties.

Using the Package Manager for Migration

The Package Manager is a utility that runs on FileVault. When a package is uploaded, the install function of the package manager unzips the file and uses FileVault to write the content into the repository. When a package is created, the package manager uses FileVault to export the content, and then packages the resulting file structure into a zipped folder.

Content packages can contain content or project-related data. It is a zipped file that resembles the Vault serialization. The `.content.xml` file represents the content from the repository, in the form of files and folders. It contains vault meta-information, filter definitions, and import-configuration information.

The package must have the following structure:

- **jcr_root**: This folder represents the root node of the repository, and contains the actual content of the package.
- **META-INF**: This folder contains metadata regarding node definitions, and also contains the filter.xml, which gives directions to FileVault about which paths to include.

Applying Data Migration Best Practices

The following sections are best practices used in data migration.

Creating Pages Dynamically

Adobe Experience Manager has some very powerful high-level APIs that you can use to create pages based on the underlying data structures. You can create paragraph or text nodes, tag a page, or activate a page.

A page consists of nodes, and properties along with a `sling:resourceType`. You can use the `com.day.cq.wcm.api.PageManager` class to point to the underlying resources in xml and automate the process of data migration.

Perform Task – Create a page dynamically, from the Lab Activity section.

Creating Assets Dynamically

Assets consist of nodes, and properties along with a `sling:resourceType`. You can use the `com.day.cq.dam.api.AssetManager` to create the asset, and the `com.day.cq.tagging.TagManager` APIs to tag the assets. Using these APIs, you can create assets, tag them, as well as add metadata to the assets.

You can use this process for migration by pointing to the existing assets and creating the asset in a Digital Asset Manager, with all the asset information that is required.

Perform Task – Create an asset dynamically, from the Lab Activity section.

Identifying Cost Benefit

Based on the complexity and size of the migrating system, you can use any of the following three methods for migration:

Manual migration (human entry)
Hire someone to enter the data into a format that is compatible with Adobe Experience Manager.

Automated migration (using APIs)
Export data as XML from the legacy system, then write a script to convert the data into a JCR friendly XML, and finally import that package into JCR.

Hybrid migration (combination of manual and automated)
Import the bulk of the data through XML, and hire someone to update the data manually; for example, for adding metadata to modernize the data.

Storage Elements in Adobe Experience Manager

With Adobe Experience Manager 6.0, one of the most important changes is with the repository. There are two node storage implementations—Tar storage and MongoDB storage.

Tar Storage

Tar files store content as various types of records with larger segments. Journals are used to track the latest state of the repository.

Key features of Tar storage:

- Immutable Segments
- Locality
- Compactness

By default, Adobe Experience Manager 6.0 uses the Tar storage to store nodes and binaries, using the default configuration options. To manually configure the storage settings, you can perform the following steps:

1. Download the Adobe Experience Manager quickstart JAR and place it in a new folder.
2. Unpack the file by executing the following command in the prompt:

```
java –jar cq-quickstart-6.jar -unpack
```

3. Create a folder named **crx-quickstart\install** in the installation directory.

4. Create a file called org.apache.jackrabbit.oak.plugins.segment.SegmentNodeStoreService.cfg in the newly created folder.
5. Edit the file and set the configuration options.
    l. **repository.home**: Path to the repository home under which various repository related data is stored.
    m. **tarmk.size**: Maximum size of a segment in MB.
6. Start Adobe Experience Manager.

MongoDB Storage

The MongoDB storage leverages MongoDB for sharding and clustering. The repository tree is kept in one MongoDB database where each node is a separate document.

MongoDB storage features:

- Revisions
- Branches
- Previous documents
- Cluster node metadata

You can configure Adobe Experience Manager to run with MongoDB storage by following these steps:

1. Download the Adobe Experience Manager quickstart JAR and place it in a new folder.
2. Unpack the file by executing the following command in the prompt:

```
java –jar cq-quickstart-6.jar -unpack
```

3. Ensure that MongoDB is installed and an instance of mongod is running.
4. Create a folder named **crx-quickstart\install** in the installation directory.
5. Configure the node store by creating a configuration file with the name of the configuration you want to use in the crx-quickstart\install directory.
6. Edit the file and set your configuration options. The following options are available:
    a. **mongouri**: The MongoURI required to connect to Mongo Database. The default is mongodb://localhost:27017
    b. **db**: Name of the Mongo database. By default new Adobe Experience Manager 6.x installations use **aem-author** as the database name.
    c. **cache**: The cache size in MB. This is distributed among various caches used in DocumentNodeStore.
    d. **changesSize**: Size in MB of capped collection used in Mongo for caching the diff output.
    e. **customBlobStore**: Boolean value indicating a custom data store will be used. The default is false.
7. Create a configuration file with the PID of the data store you want to use and edit the file in order to set the configuration options. For more information, see Configuring Node Stores and Data Stores.

8. Start the Adobe Experience Manager 6.x jar with a MongoDB storage backend by running:

```
java –jar cq-quickstart-6.jar –r crx3,crx3mongo
```

Compacting Tar Files

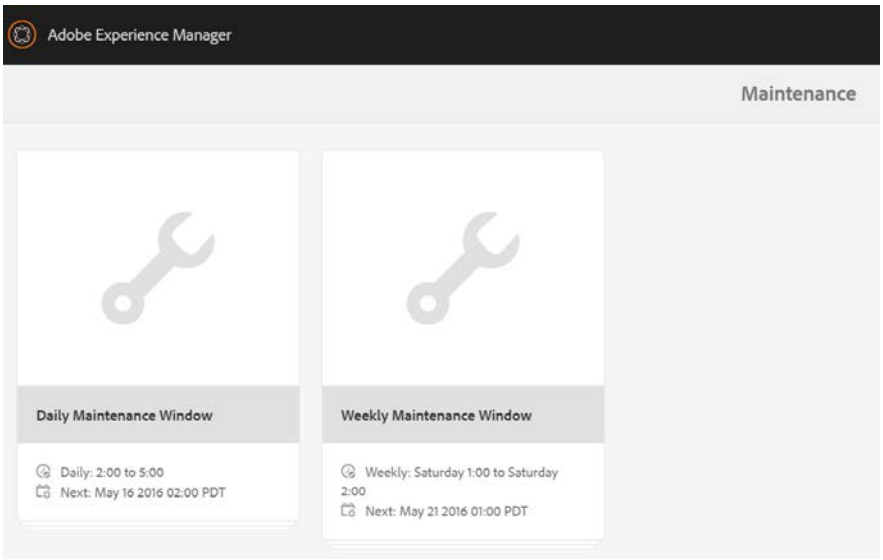
When you update existing data, the existing data is never overwritten, which causes the size of the repository to increase. To deal with this increase, Adobe Experience Manager has a garbage collection mechanism known as Tar Compaction. This mechanism reclaims disk space by removing obsolete data from the repository.

There are various methods to perform compaction:

- Revision cleanup using Operations Dashboard
- Revision cleanup using JMX Console
- Offline Compaction
- Online Compaction

1.1.5 Revision cleanup using Operations Dashboard

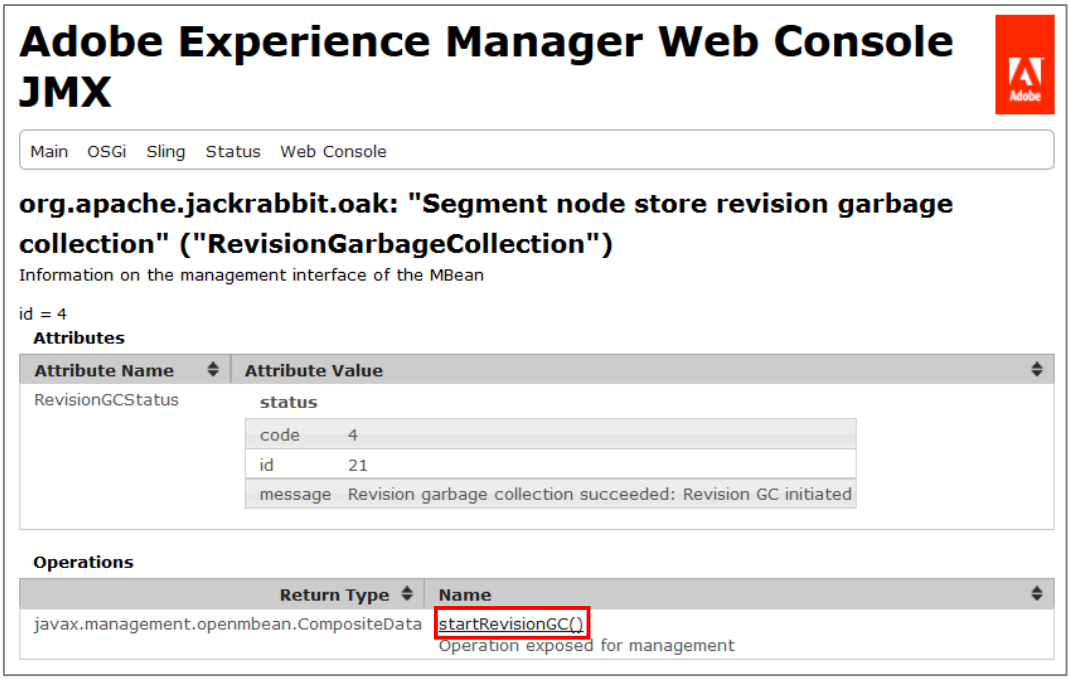
Revision Cleanup is a daily maintenance tool available in the Operations Dashboard (http://localhost:4502/libs/granite/operations/content/maintenance.html > **Daily Maintenance Window** > **Revision Clean Up**). You can hover over the tool, and click the start icon to manually start the cleanup.



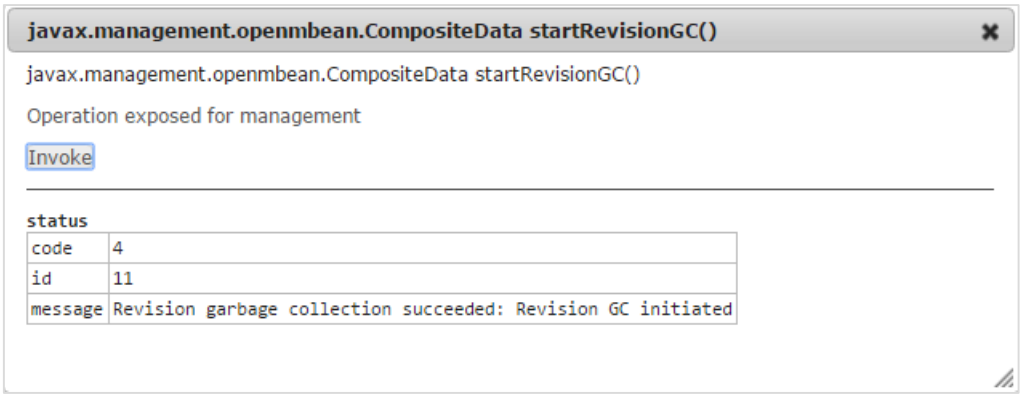
1.1.6 Revision cleanup using JMX Console

You can use the JMX console at http://localhost:4502/system/console/jmx to invoke garbage collection.

1. Search for and click the row that contains the RevisionGarbageCollection MBean. The Adobe Experience Manager Web Console JMX page opens.
2. In the Operations area, click **startRevisionGC().**



3. Click **Invoke** on the javax.managemetn.openmbean.CompositeData startRevisionGC() dialog box. The dialog box will expand and include a status area, which includes code, id, and message.



1.1.7 Offline compaction

For faster compaction of the Tar files and situations where normal garbage collection does not work, Adobe provides a manual Tar compaction tool called Oak-run. It can be downloaded at the following location: http://mvnrepository.com/artifact/org.apache.jackrabbit/oak-run/.

This tool is a runnable JAR file that can be manually run to compact the repository. To run the tool, perform the following steps:

1. Shut down Adobe Experience Manager.

```
java -jar oak-run.jar checkpoints install-folder/crx-quickstart/repository/segmentstore
```

2. Use the downloaded Oak-run tool to find old checkpoints.

3. Delete the unreferenced checkpoints.

```
java -jar oak-run.jar checkpoints install-folder/crx-quickstart/repository/segmentstore rm-
    unreferenced
```

```
java -jar oak-run.jar compact install-folder/crx-quickstart/repository/segmentstore
```

4. Run the compaction and wait for it to complete.

1.1.8 Online compaction

Online compaction is performed when Adobe Experience Manager cannot be shut down, and needs to be maintained while running.

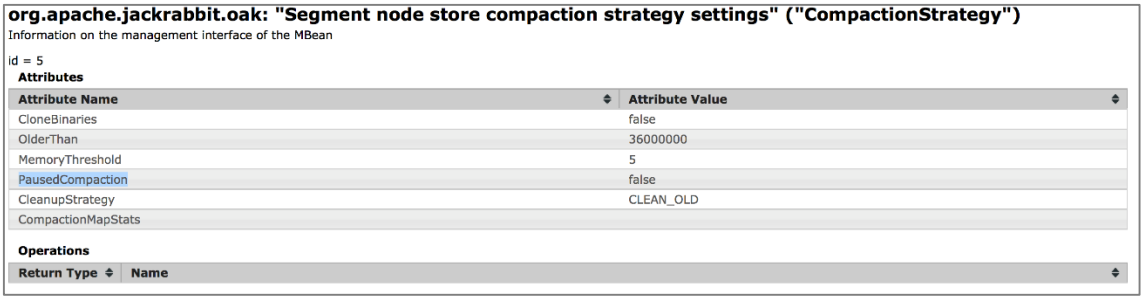
To perform online compaction, perform these steps:

1. Go to the folder where Adobe Experience Manager is installed, then browse to `crx-quickstart\install`.
2. Open the **org.apache.jackrabbit.oak.plugins.segment.SegmentNodeStoreService.config** file.

```
pauseCompaction=false
```

3. Add the following line to the configuration file:

4. Restart Adobe Experience Manager.
5. Go to the JMX console by pointing your browser to: http://server:port/system/console/jmx
6. Search for **CompactionStrategy** and click the MBean that shows up in the search.
7. Verify the value for **PausedCompaction** is set to false. This confirms that online compaction is set to run:



8. Verify if Online Compaction is running properly. You can do this by first going to the Operations Dashboard and checking what is the time interval configured for the **Daily Maintenance Window.** By default, it is scheduled to run between 2 AM and 5 AM.

9. Inspect the **error.log** file for events logged during the time of the daily maintenance window to see if online compaction ran correctly.

Chapter 12 Lab Activity

Scenario

Your company plans to migrate from an existing Content Management System (CMS) to Adobe Experience Manager. Using existing data, you need to perform a series of steps to move all content into the new repository.

Challenge

There are many ways to migrate content from an existing CMS. Selecting the right method is essential for a quick and accurate migration. These methods differ based on the type and bulk of the existing data.

Overview

You will use three methods to import data from the existing system:

Using VLT

Using Sling POST Servlet

Using the JCR API

You will also dynamically create pages and assets.

Pre-requisites

You need to have the project—trainingproject from the USB Contents. Before you can perform the tasks in this section, you must have the following:

- An AEM Author instance running in your system
- Curl installed in your system, with the path set in the Environment Variables

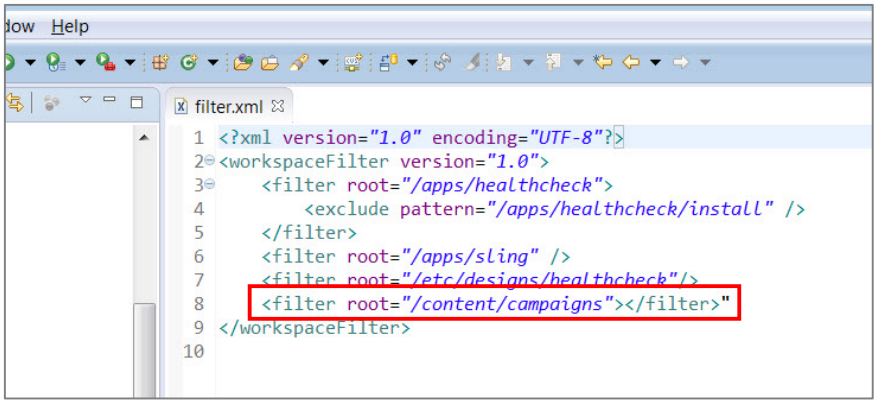
Steps

1. Task – Migrate content from legacy system using VLT

In this task, you will simulate the import of new content by adding a new campaign using VLT. All existing campaigns are first exported into the file system so that you have the required structure of nodes and properties represented in the file system and can easily replicate this. Then, by copying an existing campaign structure and modifying the copy, you simulate the import of new content in the required structure. Finally, you will use VLT to update the repository with the new content.

In **training.ui.apps/src/main/content/META-INF/vault**, open the **filter.xml** file, and add a filter for **/content/campaigns** so that VLT also serializes these nodes to the file system. That is, add the following line to the end of the file just before the **</workspaceFilter>** tag:

```
<filter root="/content/campaigns"></filter>
```



(Windows) On the command prompt, using the **cd** command, change the directory to **training.ui.apps \src\main\content\jcr_root** and execute the following command:

```
vlt up
```

You should see the campaigns nodes and the.content.xml files being copied into the file system.

```
cp -R content/campaigns/geometrixx/scott-recommends
   content/campaigns/geometrixx/shantanu-recommends
```

Copy the scott-recommends campaign to a new campaign called shantanu-recommends, by executing the following commands:

```
MAC: xcopy content\campaigns\geometrixx\scott-recommends content\
WINDOWS: campaigns\geometrixx\shantanu-recommends /S /E /I /H
```

Ensure that you remove the **.vlt** file below by using the following commands:

```
MAC: rm content/campaigns/geometrixx/shantanu-recommends/.vlt
WINDOWS: del content\campaigns\geometrixx\shantanu-recommends\.vlt
```

In your file system, go to your current workspace by navigating to: **training\ui.apps\src\main\content\jcr_root\content\campaigns\geometrixx\shantanu-recommends**, and open the .content.xml file in an editor.

Replace all instances of **Scott** or **Scott Reynolds** with **Shantanu**, except for the .jpg image.

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq=
    jcr:primaryType="cq:Page">
    <jcr:content
        cq:lastModified="{Date}2011-02-02T16:46:52.932+01:00"
        cq:lastModifiedBy="admin"
        cq:segments="/etc/segmentation/geometrixx/male"
        cq:template="/libs/cq/personalization/templates/teaser"
        jcr:primaryType="cq:PageContent"
        jcr:title="Scott Recommends"
        sling:resourceType="cq/personalization/components/teaserpage">
        <par
            jcr:primaryType="nt:unstructured"
            sling:resourceType="foundation/components/parsys">
            <image
                jcr:created="{Date}2011-02-02T11:46:42.526+01:00"
                jcr:createdBy="admin"
                jcr:lastModified="{Date}2011-02-02T16:37:18.093+01:00"
                jcr:lastModifiedBy="admin"
                jcr:primaryType="nt:unstructured"
                sling:resourceType="foundation/components/image"
                fileReference="/content/dam/geometrixx/portraits/scott_
                imageCrop="156,124,718,557"
                imageRotate="0"
                width="220"/>
            <text
                jcr:created="{Date}2011-02-02T11:47:07.053+01:00"
                jcr:createdBy="admin"
                jcr:lastModified="{Date}2011-02-02T16:46:52.929+01:00"
                jcr:lastModifiedBy="admin"
                jcr:primaryType="nt:unstructured"
                sling:resourceType="foundation/components/text"
                text="&lt;p&gt;&lt;b&gt;Scott Reynolds recommends &lt;/b&gt; &am
                href=&quot;/content/geometrixx/en/products/circle.html&
                textIsRich="true"/>
```

Add the new content under version control by executing the following command:

```
vlt add content/campaigns/geometrixx/shantanu-recommends
```

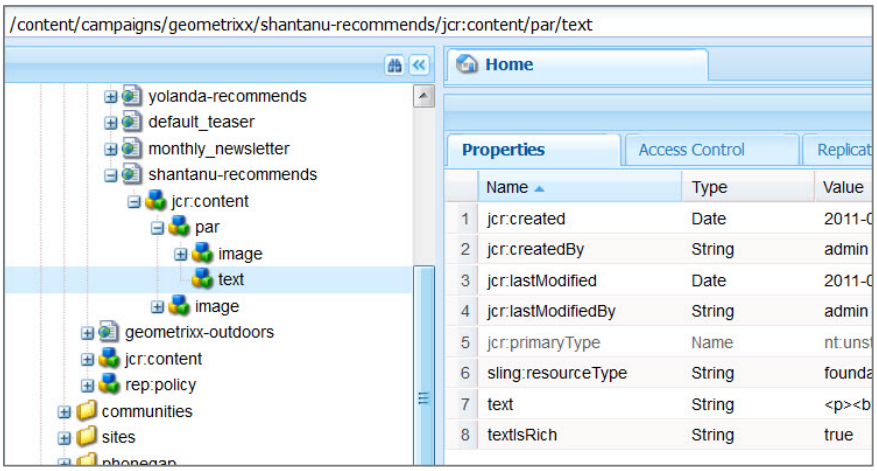
Check in (ci) the new content by executing the following command:

```
vlt ci content/campaigns/geometrixx/shantanu-recommends
```

Inspect the new campaign in the Site admin.

2. Task – Use Sling POST servlet to test XML

Inspect the campaigns in CRXDE Lite. You need to add a text node to Shantanu's campaign.

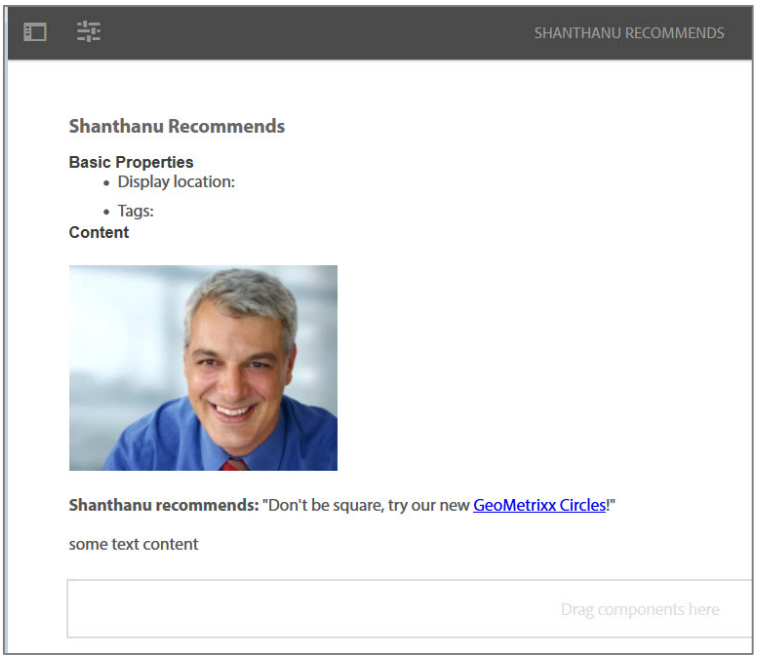


On the command line, execute the following command (it is a one-line command, with no spaces in the URI):

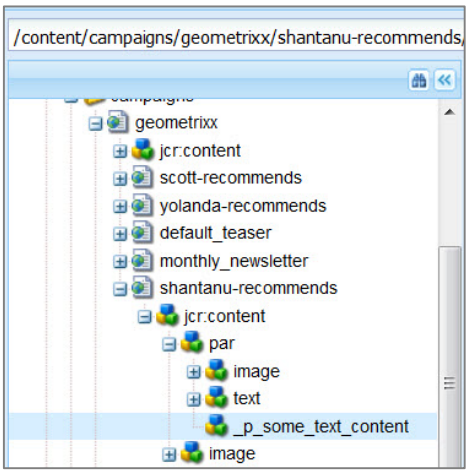
```
curl -u admin:admin -X POST -d "sling:resourceType=foundation/components/text&text=<p>
some text content</p>&textIsRich=true"
   http://localhost:4502/content/campaigns/geometrixx/shantanu-recommends/jcr:content/par/*
```

This is POSTing a rich text string to the par node under the shantanu-recommends campaign.

Double-click the campaign in the Site admin. You should see that there is a new text component.



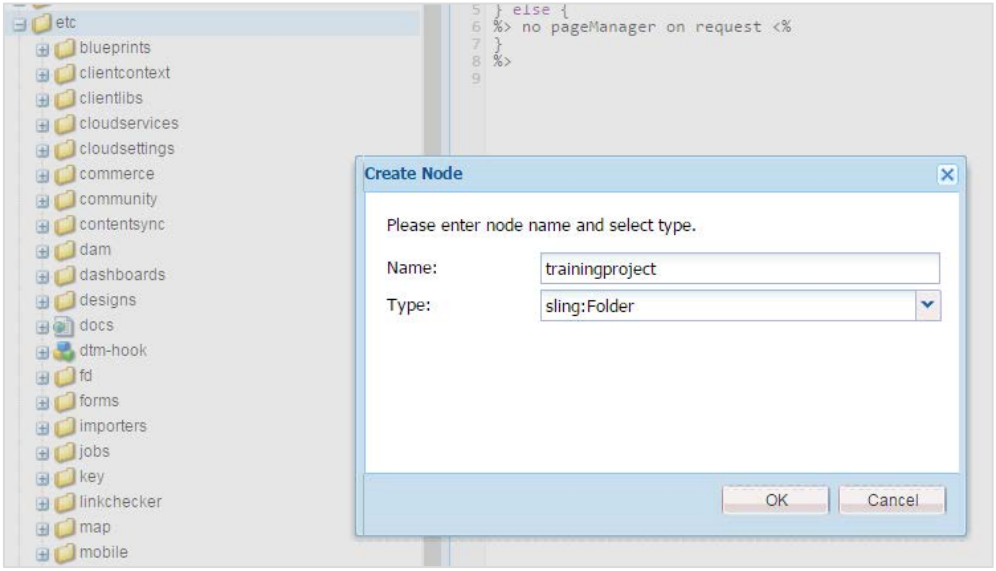
If you check the nodes in CRXDE Lite, you can see the new node added.



3. Task – Use the JCR API to migrate data

In this task, you will use a JSP script that makes use of the JCR API to search for a specific content, and add new content wherever it is found. You can use this technique to update content from a legacy system, by adding the required nodes or properties to make the content usable in Adobe Experience Manager.

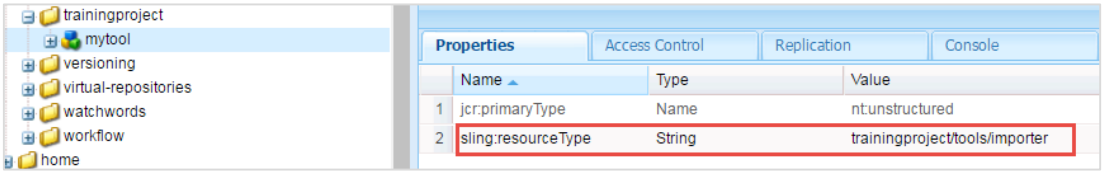
In CRXDE Lite, navigate to the **/etc** node, and create a new node called **trainingproject**, of type **sling:Folder**.



Under the new **trainingproject** node, add another node called **mytool**, of type **nt:unstructured**.

Add the following property to the mytool node. This will be the resource that is addressed with the browser to execute the JSP script.

Name	Type	Value
sling:resourceType	String	trainingproject/tools/importer



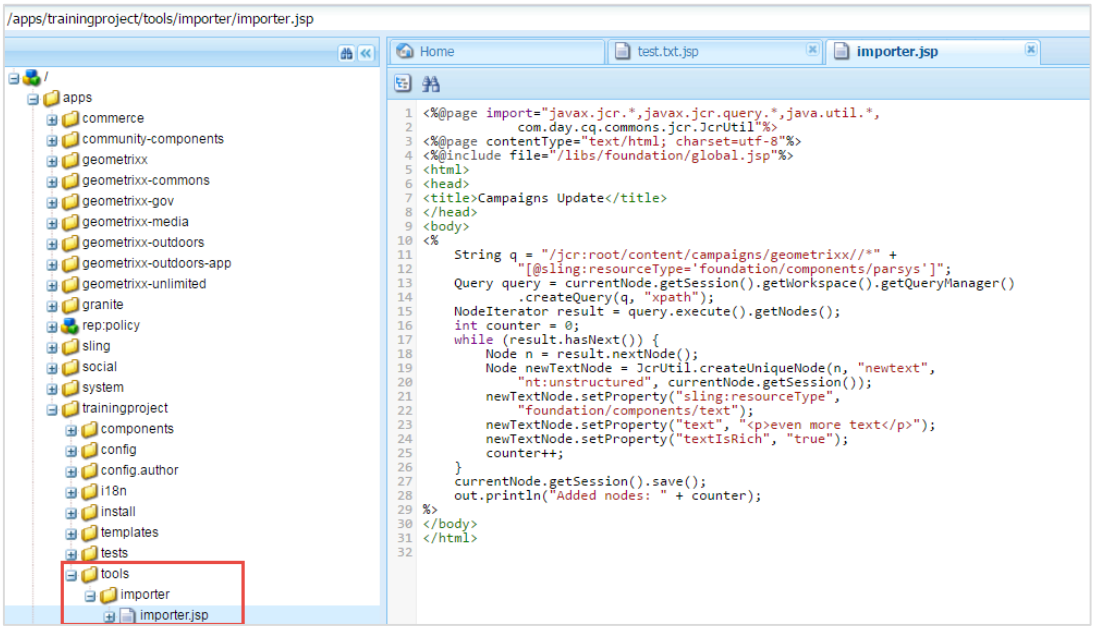
Save the changes.

Navigate to **/apps/trainingproject**, and add a new folder named **tools**.

Under **tools**, add another folder named **importer**.

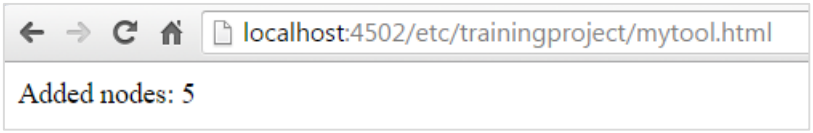
In the **importer** folder, add a new file named **importer.jsp**.

Include the following script in the importer.jsp file. This script will search for paragraph nodes in the Geometrixx campaigns by checking for **sling:resourceType='foundation/components/parsys'**, and add child nodes with text to all the nodes found.



```
<%@page import="javax.jcr.*,javax.jcr.query.*,java.util.*,
com.day.cq.commons.jcr.JcrUtil"%>
<%@page contentType="text/html; charset=utf-8"%>
<%@include file="/libs/foundation/global.jsp"%>
<html>
<head>
<title>Campaigns Update</title>
</head>
<body>
<%
String q = "/jcr:root/content/campaigns/geometrixx//*" +
       "[@sling:resourceType='foundation/components/parsys']";
Query query = currentNode.getSession().getWorkspace().getQueryManager()
.createQuery(q, "xpath");
NodeIterator result = query.execute().getNodes();
int counter = 0;
while (result.hasNext()) {
    Node n = result.nextNode();
    Node newTextNode = JcrUtil.createUniqueNode(n, "newtext",
           "nt:unstructured", currentNode.getSession());
    newTextNode.setProperty("sling:resourceType",
           "foundation/components/text");
    newTextNode.setProperty("text", "<p>even more text</p>");
    newTextNode.setProperty("textIsRich", "true");
    counter++;
}
currentNode.getSession().save();
out.println("Added nodes: " + counter);
%>
</body>
</html>
```

Open the link http://localhost:4502/etc/trainingproject/mytool.html to execute the code.



Open the Shantanu Recommends campaign from the Site admin. It should have an additional text node.

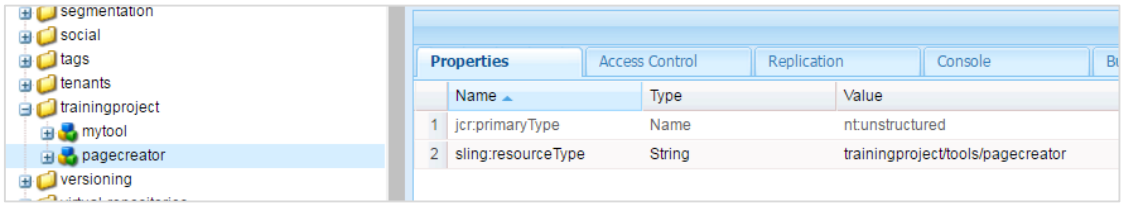
4. Task – Create a page dynamically

In this task, you will use a JSP script that makes use of **com.day.cq.wcm.api.PageManager** and create pages dynamically. This will be very useful in a scenario where you have to migrate existing pages from another repository to Adobe Experience Manager. You need to make sure the page content is defined and mapped correctly as per your requirement for migration. We will be creating the page dynamically by providing the page component and other details in our implementation.

In CRXDE Lite, navigate to the **/etc/trainingproject** node, which we created in an earlier task.

Create a new **nt:unstructured** node named **pagecreator**.

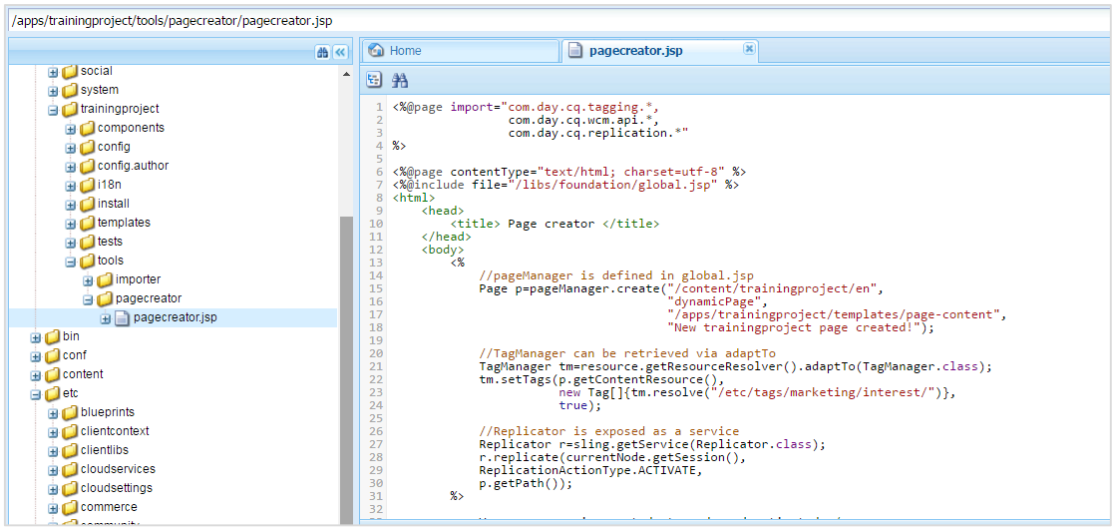
Add the sling:resourceType property to the node with value as trainingproject/tools/pagecreator.



Click **Save All** in the upper-left corner.

Navigate to /apps/trainingproject/tools and create a new nt:folder node named pagecreator.

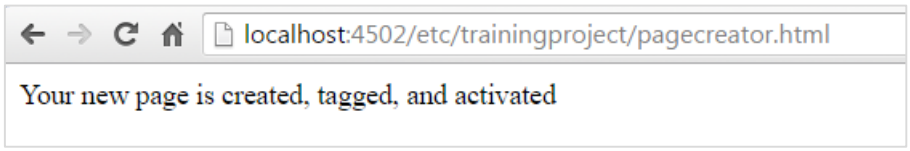
In **pagecreator** folder, create a new file **pagecreator.jsp** with the following code:



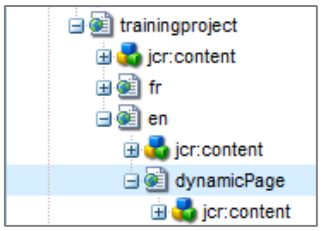
```
<%@page import="com.day.cq.tagging.*,
com.day.cq.wcm.api.*,
com.day.cq.replication.*"
%>

<%@page contentType="text/html; charset=utf-8" %>
<%@include file="/libs/foundation/global.jsp" %>
<html>
<head>
<title> Page creator </title>
</head>
<body>
<%
//pageManager is defined in global.jsp
Page p=pageManager.create("/content/trainingproject/en",
                          "dynamicPage",
                          "/apps/trainingproject/templates/page-content",
                          "New trainingproject page created!");
//TagManager can be retrieved via adaptTo
TagManager tm=resource.getResourceResolver().adaptTo(TagManager.class);
tm.setTags(p.getContentResource(),
       new Tag[]{tm.resolve("/etc/tags/marketing/interest/")},true);

//Replicator is exposed as a service
Replicator r=sling.getService(Replicator.class);
r.replicate(currentNode.getSession(),
       ReplicationActionType.ACTIVATE,
       p.getPath());
%>
<p> Page created, tagged, and activated </p>
</body>
</html>
```

Click **Save All** in the upper-left corner.

Open http://localhost:4502/etc/trainingproject/pagecreator.html and you will see the following message:

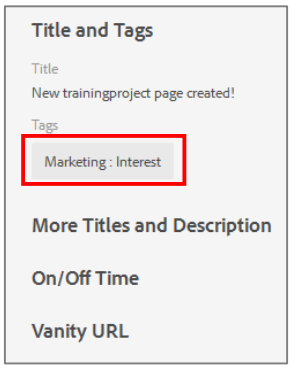


In CRXDE, navigate to **/content/trainingproject/en** and check if the page is created successfully.



Open the newly created page: http://localhost:4502/content/trainingproject/en/dynamicPage.html

Open the newly created page in site admin and notice the tag added during the page creation.



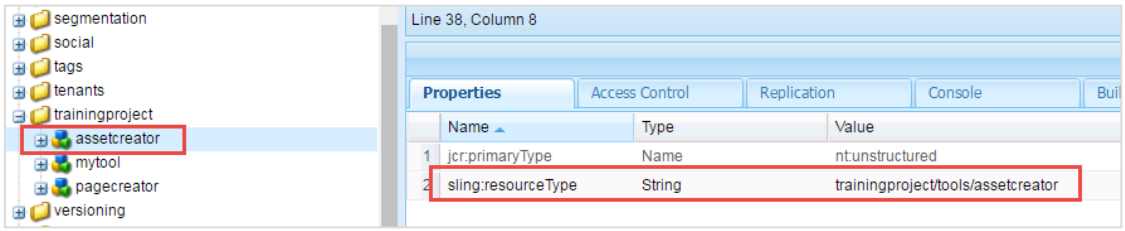
5. Task – Create an asset dynamically

In this task, you will use a JSP script that makes use of com.day.cq.dam.api.AssetManager and create assets dynamically. This will be very useful in a scenario where you have to migrate existing assets from another repository to Adobe Experience Manager. You need to make sure that the assets properties are defined and mapped correctly as per your requirement during migration. You will be creating an asset dynamically by providing the asset details in your implementation.

In CRXDE Lite, navigate to the **/etc/trainingproject** node, which we created in an earlier task.

Create a new nt:unstructured node named assetcreator.

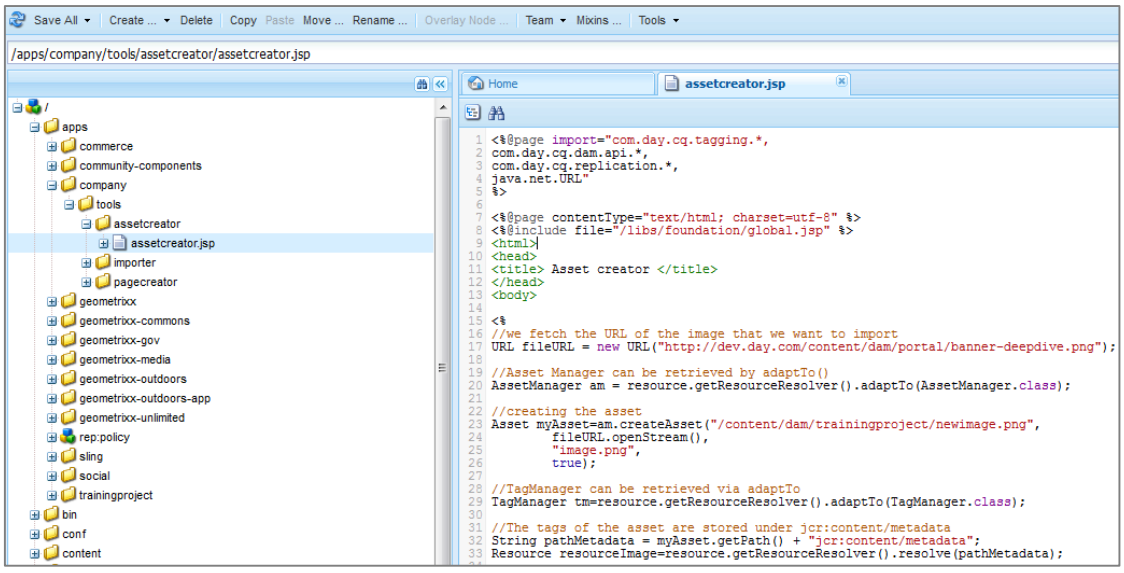
Add the sling:resourceType property to the node with value: trainingproject/tools/assetcreator



Click **Save all** in the upper-left corner.

Navigate to /apps/trainingproject/tools and create a new nt:folder named assetcreator.

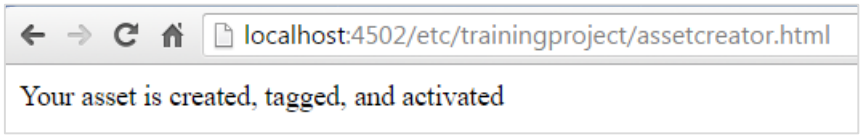
In **assetcreator** folder, create a new file **assetcreator.jsp** with the following code:



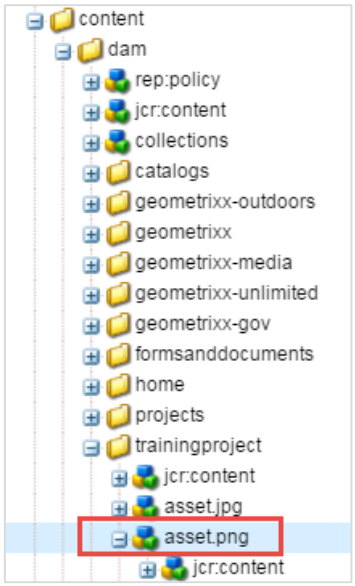
```
<%@page import="com.day.cq.tagging.*,
com.day.cq.dam.api.*,
com.day.cq.replication.*,
java.net.URL"
%>

<%@page contentType="text/html; charset=utf-8" %>
<%@include file="/libs/foundation/global.jsp" %>
<html>
<head>
<title> Asset creator </title>
</head>
<body>

<%
//we fetch the URL of the image that we want to import
URL fileURL = new URL("http://sample.scene7.com/is/image/s7u/aem_training2RB");

//Asset Manager can be retrieved by adaptTo()
AssetManager am = resource.getResourceResolver().adaptTo(AssetManager.class);

//creating the asset
Asset myAsset=am.createAsset("/content/dam/trainingproject/newimage.png",
        fileURL.openStream(),
        "image.png",
        true);

//TagManager can be retrieved via adaptTo
TagManager tm=resource.getResourceResolver().adaptTo(TagManager.class);

//The tags of the asset are stored under jcr:content/metadata
String pathMetadata = myAsset.getPath() + "/jcr:content/metadata";
Resource resourceImage=resource.getResourceResolver().resolve(pathMetadata);

tm.setTags(resourceImage,
        new Tag[]{tm.resolve("/etc/tags/stockphotography/technology/")},
        true);

//Replicator is exposed as a service
Replicator r=sling.getService(Replicator.class);
r.replicate(currentNode.getSession(),
        ReplicationActionType.ACTIVATE,
        myAsset.getPath());
%>
<p> Asset created, tagged, and activated </p>
</body>
</html>
```

Click **Save All** in the upper-left corner.

Open http://localhost:4502/etc/trainingproject/assetcreator.html and you will see the following message:



In CRXDE, navigate to **/content/dam/trainingproject** and check if the **asset.png** file was created.



Open the newly created asset and check the tag created by code.

Remember to run **vlt up** or an Import from Server in Eclipse to retrieve the changes made to the repository.

You can modify your code to process multiple pages and assets so that they can be migrated to Adobe Experience Manager in batches.

Scenario Conclusion

Using the various methods, you have successfully migrated content from legacy systems to Adobe Experience Manager. You have also dynamically created pages and assets.

CHAPTER THIRTEEN: USERS, GROUPS AND PERMISSIONS

Overview

This chapter includes content and hands-on activities to determine what actions a user or group can take and where it can perform those actions. Adobe Experience Manager uses Access Control Lists (ACLs) to perform these tasks.

Objectives

By the end of this chapter, you will be able to:

- create and assign users, groups, and permissions

Permissions and ACLs

Permissions define who is allowed to perform which actions on a resource. The permissions are the result of access control evaluations. You can change the permissions granted/denied to a given user by selecting or clearing the check boxes for the individual AEM actions. A check mark indicates that an action is allowed. No check mark indicates that an action is denied.

Where the check mark is located in the grid also indicates what permissions users have in what locations within AEM (that is, which paths).

Action	Description
Allow (check mark)	CQ WCM allows the user to perform the action on this page or any child pages.
Deny (no check mark)	CQ WCM does not allow the user to perform the action on this page or on any child pages.

The permissions are also applied to any child pages. If a permission is not inherited from the parent node but has at least one local entry for it, the following symbols are appended to the check box. A local entry is one that is created in the CRX 2.3 interface. A wildcard in the path of ACLs currently can only be created in CRX.

For an action at a given path:

* (asterisk)	There is at least one local entry (effective or ineffective). These wildcard ACLs are defined in CRX.
! (exclamation mark)	There is at least one entry that currently has no effect.

When you move your cursor over the asterisk or exclamation mark, a tooltip provides more details about the declared entries. The tooltip is split into two parts:

Upper part	Lists the effective entries
Lower part	Lists the non-effective entries that may have an effect somewhere else in the tree (as indicated by a special attribute present with the corresponding ACE limiting the scope of the entry). Alternatively, this is an entry whose effect has been revoked by another entry defined at the given path or at an ancestor node.

Actions

You can perform actions on a page (resource). For each page in the hierarchy, you can specify which action the user is allowed to take on that page. Permissions enable you to allow or deny an action.

Action	Description
Read	The user is allowed to read the page and any child pages.
Modify	The user can modify existing content on the page and on any child pages. At the JCR level, users can modify a resource by its properties, locking, versioning, and nt-modifications, and they have complete write permission on nodes defining a jcr:content child node, for example, cq:page, nt:file, cq:Asset.
Create	The user can: Create a new paragraph on the page or on any child page. Create a new page or child page. If modify is denied, the sub-trees below jcr:content are specifically excluded because the creation of jcr:content and its child nodes are considered a page modification. This only applies to nodes defining a jcr:content child node.
Delete	The user can: Delete existing paragraphs from the page or any child page. Delete a page or child page. If modify is denied, any sub-trees below jcr:content are specifically excluded because removing jcr:content and its child nodes is considered a page modification. This only applies to nodes defining a jcr:content child node.
Read ACL	The user can read the access control list of the page or child pages.
Edit ACL	The user can modify the access control list of the page or any child pages.
Replicate	The user can replicate content to another environment (for example, the publish instance). The privilege is also applied to any child pages.

Access Control Lists and How They Are Evaluated

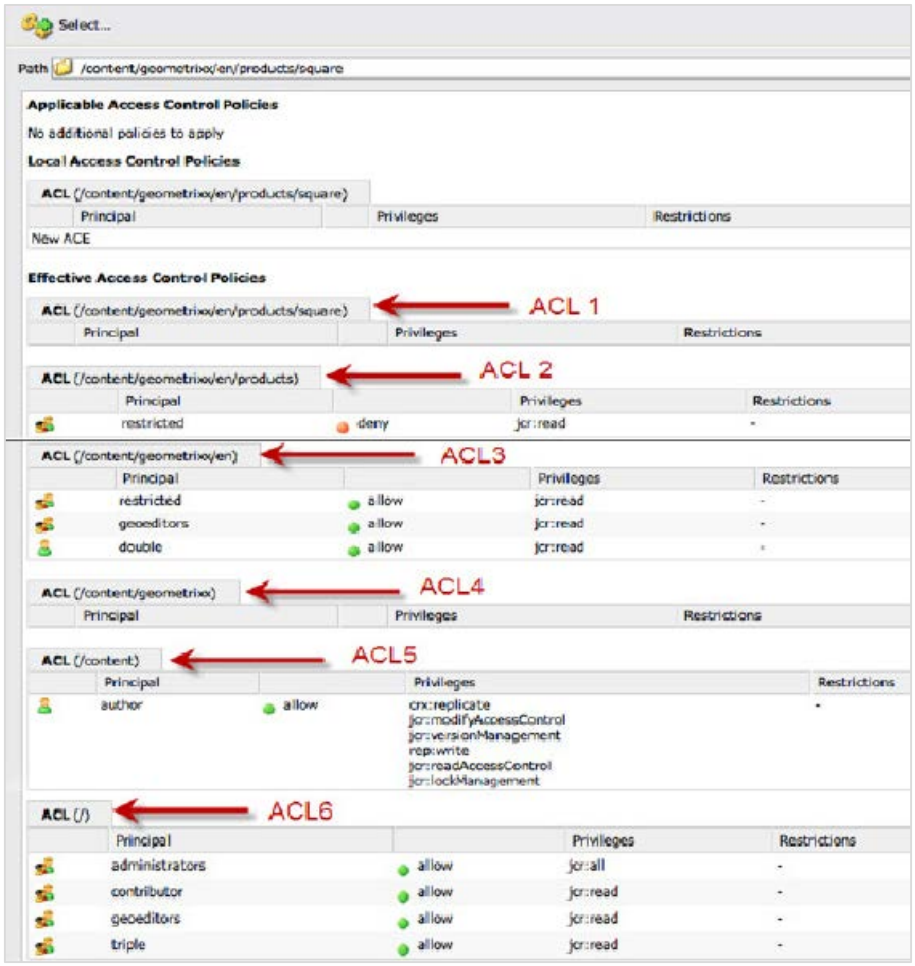
AEM sites use Access Control Lists (ACLs) to organize the permissions being applied to the various pages.

ACLs are made up of individual permissions and are used to determine the order in which these permissions are actually applied. The list is formed according to the hierarchy of the pages under consideration. This list is then scanned bottom-up until the first appropriate permission to apply to a page is found.

Assume that a user wants to access to the following page:

```
/content/geometrixx/en/products/square
```

... and the ACL list for that page is the following one:



The ACL (or permission) that will be applied to the page is ACL 1, related to /content/geometrixx/en/products/square. In our case, the ACL associated with this page is empty, so AEM will go one level up to ACL 2.

ACL 2 will be applied if the user belongs to the restricted group and in this case, the user will be denied access for this page. If the user is not part of the restricted group, AEM will go up another level to ACL 3.

ACL 3 will be applied if the user belongs to the group geoeditors or double only. In this case, the user will have access granted to the page. If the user is part of the restricted group, as we saw, the ACL applied is ACL 2. If the user is not part of the three mentioned groups, AEM will go up one level to ACL 4.

With ACL 4 being empty, if AEM reaches this level, it will go up again one level to ACL 5.

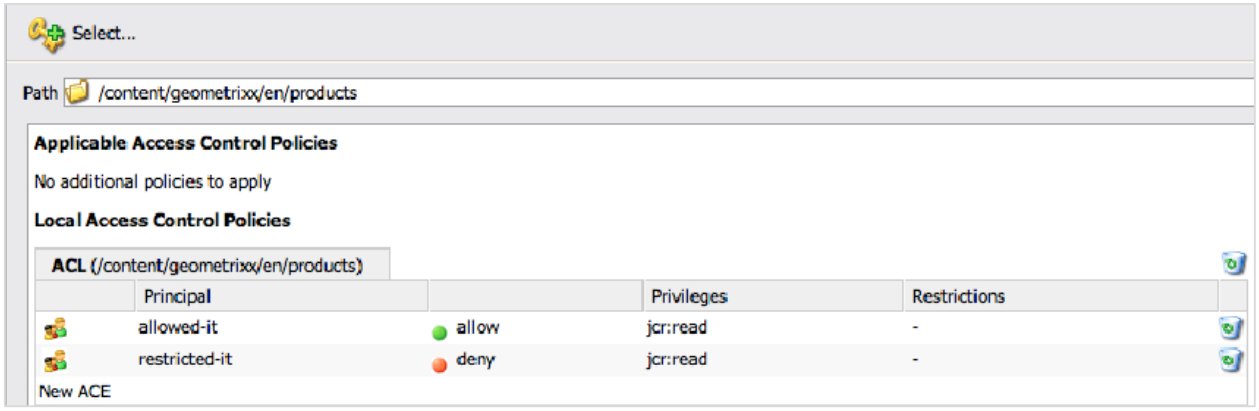
ACL 5 will be applied if the user belongs to the author group, and then the user will have access granted to the page. If the user is not part of the author group, AEM will go up one level to ACL 6.

ACL 6 will be applied if the user belongs to the administrators, contributor, or triple group. If this is the case, the user will have access granted to the page. If the user is part of the geoeditors group, as we saw, AEM would have already applied ACL 3 and granted access to the page. If the user is not part of any group, access to the page is denied.

Concurrent Permission on ACLs

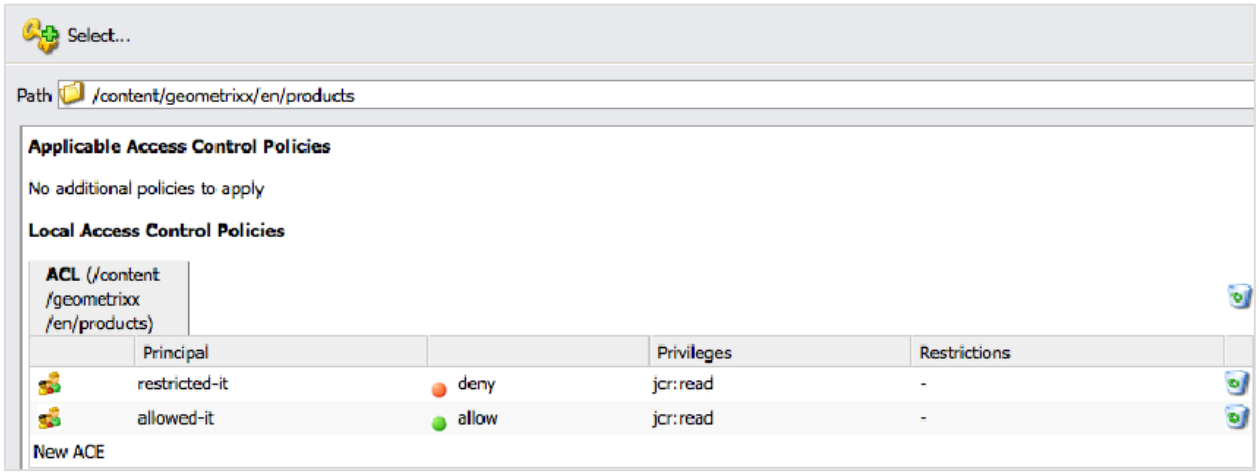
When two concurrent (and opposing) permissions are listed on the same ACL for the same resource, the ACL that is applied for such resource is the one at the bottom.

Suppose we have the following ACL for the same resource under /content/geometrixx/en/products:



If users are part of the two groups allowed-it and restricted-it, they will see the access to the page products denied because the ACL deny in read access is the rule at the bottom.

Now, if the order of the ACL is the opposite:



This time, when users are part of the two groups allowed-it and restricted-it, they will see the access granted to the page products because the ACL allow in read access is the rule at the bottom.

Chapter 13 Lab Activity

Scenario

You are in charge of ensuring that the right users are given access to the right pages/tools.

Challenge

Identifying which users fall under which group, as well as what rights to give them.

Overview

Use Access Control Lists (ACLs) to manage access to users.

Pre-requisites

- Existing project—**trainingproject** from the USB Contents
- AEM Author instance

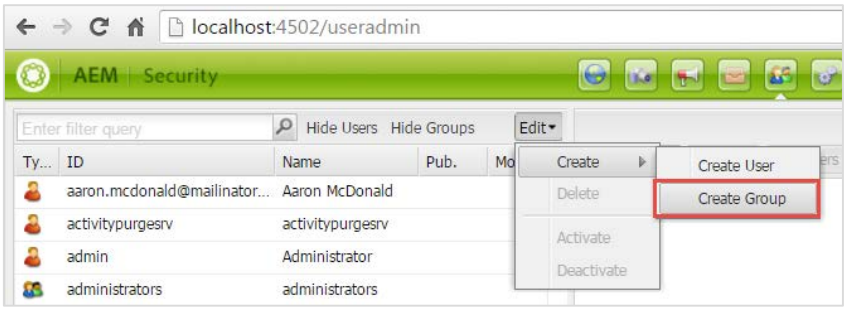
Steps

1. Task – Work with ACLs

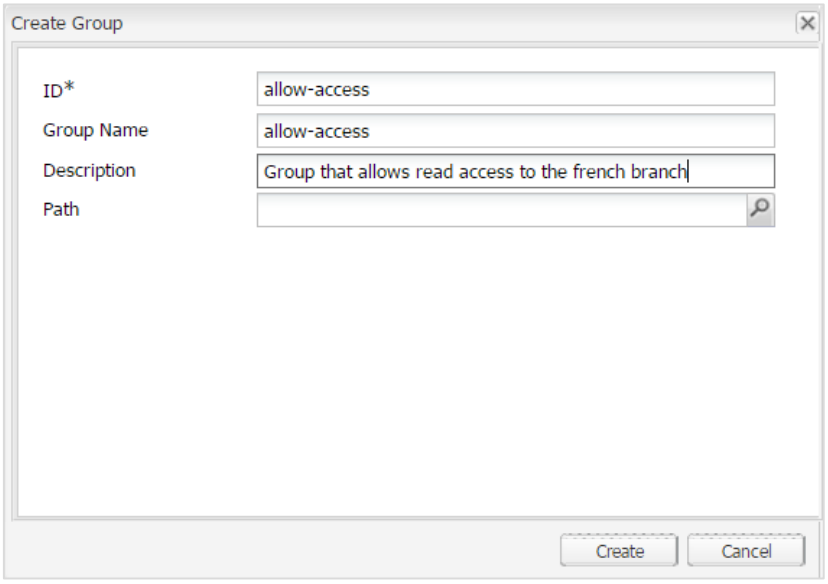
You will create a user who will be part of two groups with two different ACLs and you will modify them programmatically.

Navigate to http://localhost:4502/useradmin.

Click **Edit**, select **Create**, and then select **Create Group**. The group will have read access to **/content/geometrixx/fr**.



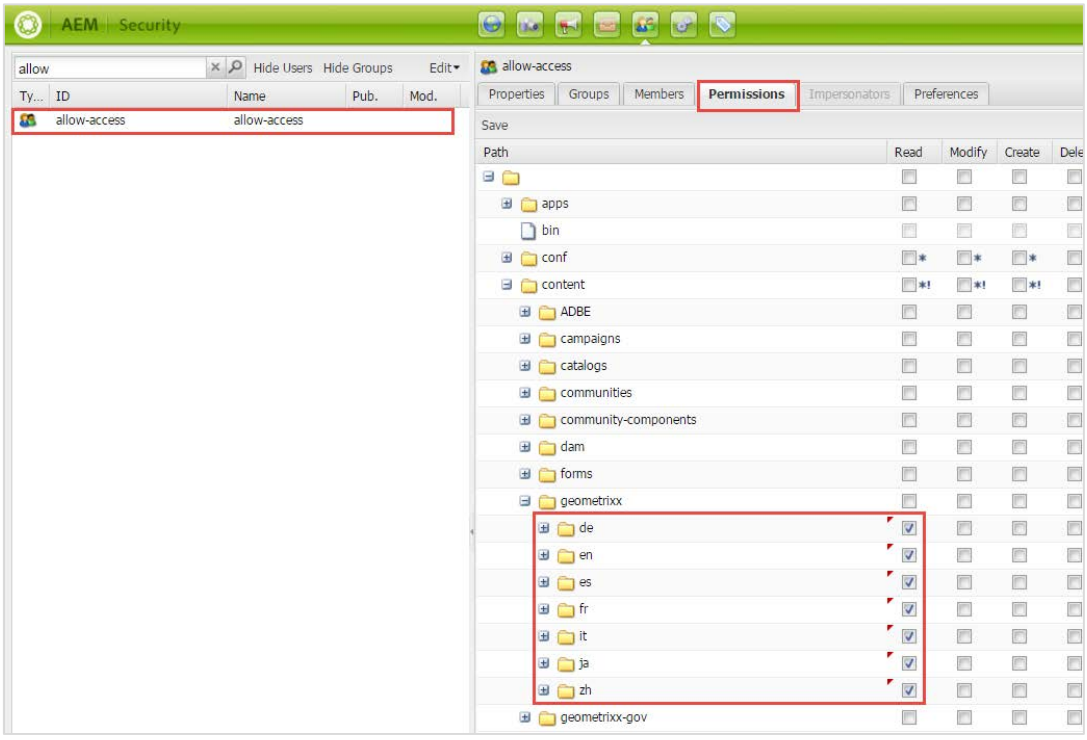
Create the group as follows:



From the left pane, double-click the newly created **allow-access** group. If you do not see the group, refresh your page.

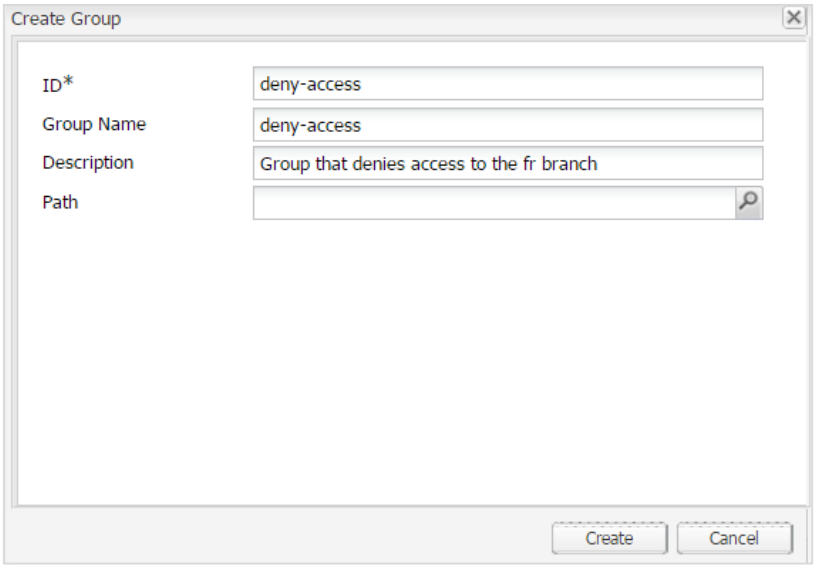
In the allow-access (group) window, click the **Permissions** tab.

Assign read rights to all the language branches as shown below (by expanding the content folder: content > geometrixx):



Click **Save**.

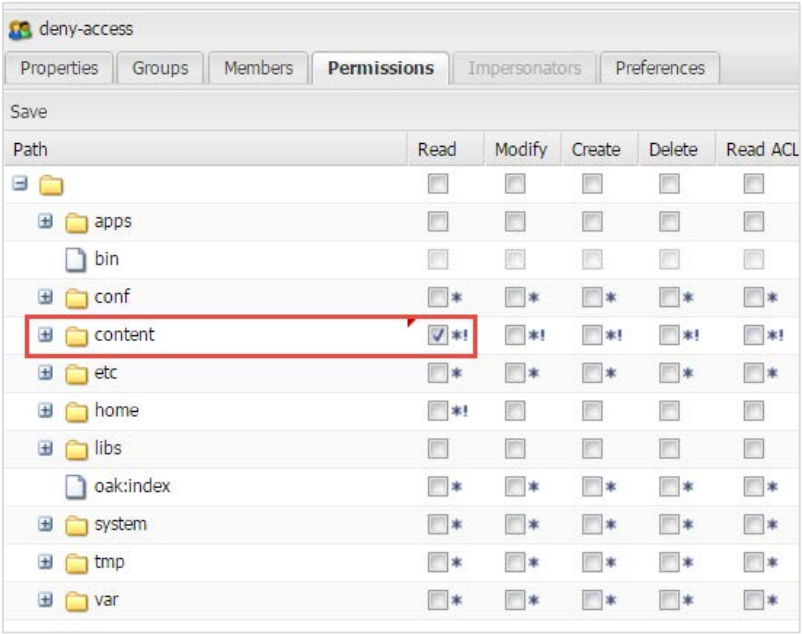
Create another group named **deny-access**, which will not have read access to **/content/geometrixx/fr**.



Open the newly created group by either searching for the group in the search box, or by using the **Back/Next** buttons.

Assign read rights to all the language branches, except for the French one as follows:

Provide read access to the **/content** node.

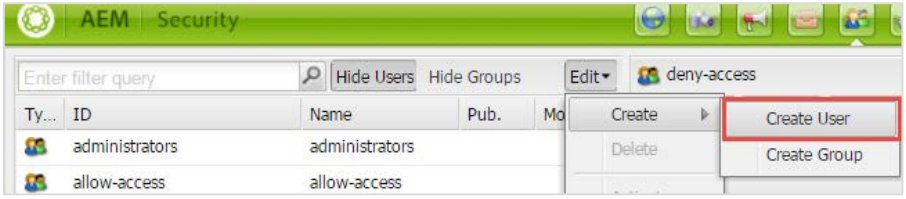


Click **Save** to save the changes.

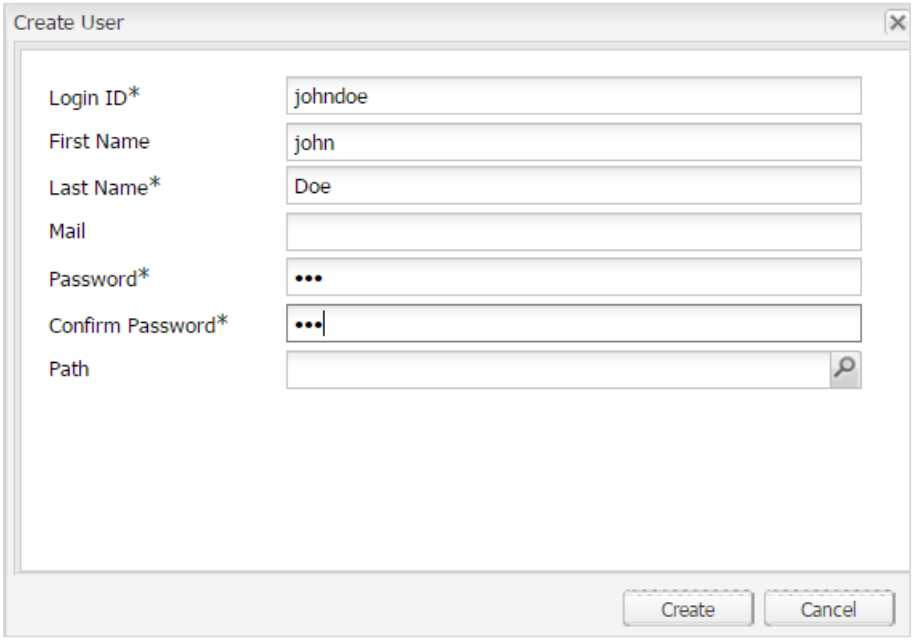
Refresh the deny-access group, expand the /content/geometrixx node, and deselect the **fr** node.

Save your changes once more.

Now, you need to create a new user named John Doe. On the menu, click **Edit** > **Create** > **Create User**.



Give the following details to the user:



NOTE: You can give any password.

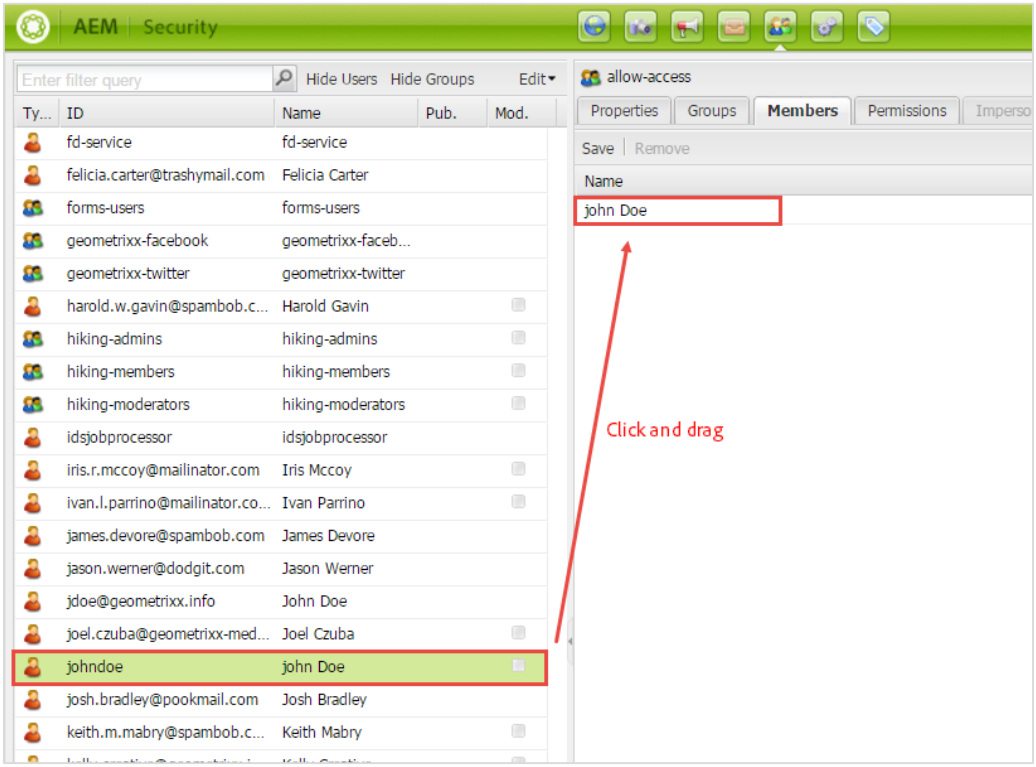
Add the John Doe user to the allow-access and deny-access groups. To achieve this, perform these steps:

Open the **allow-access** group.

Select the **Members** tab.

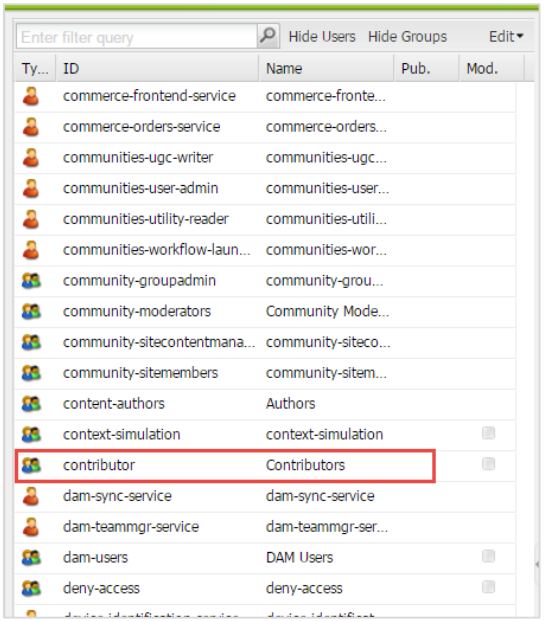
From the right-pane, click and drag the **John Doe** user to the member list.

Click **Save**.



Repeat steps 'a' through 'd' for the deny-access group.

Following the above method, add the user to the **contributor** group so that elements liks CSS-JS libraries and the design associated with the page can be accessed.



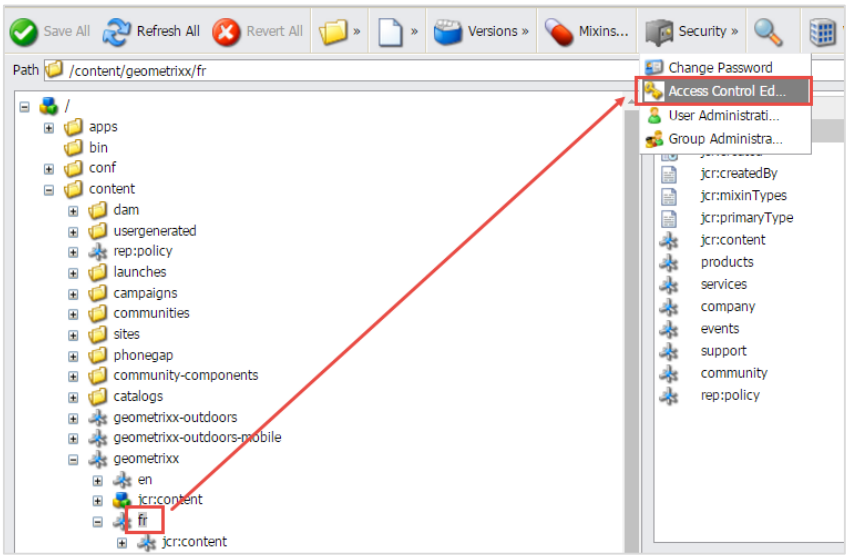
Open CRX Explorer (http://localhost:4502/crx/explorer/index.jsp)

Select Content Explorer.

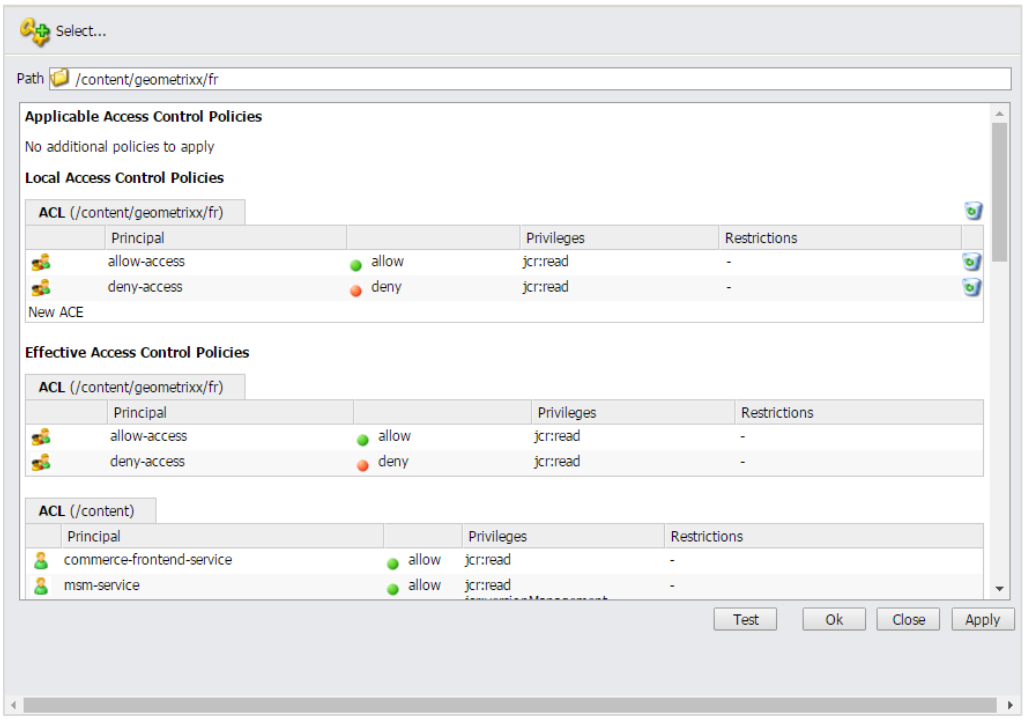
In the left pane, navigate to **content** > **geometrixx**.

Select the **fr** node.

From the top menu, select **Security** > **Access Control Ed...**



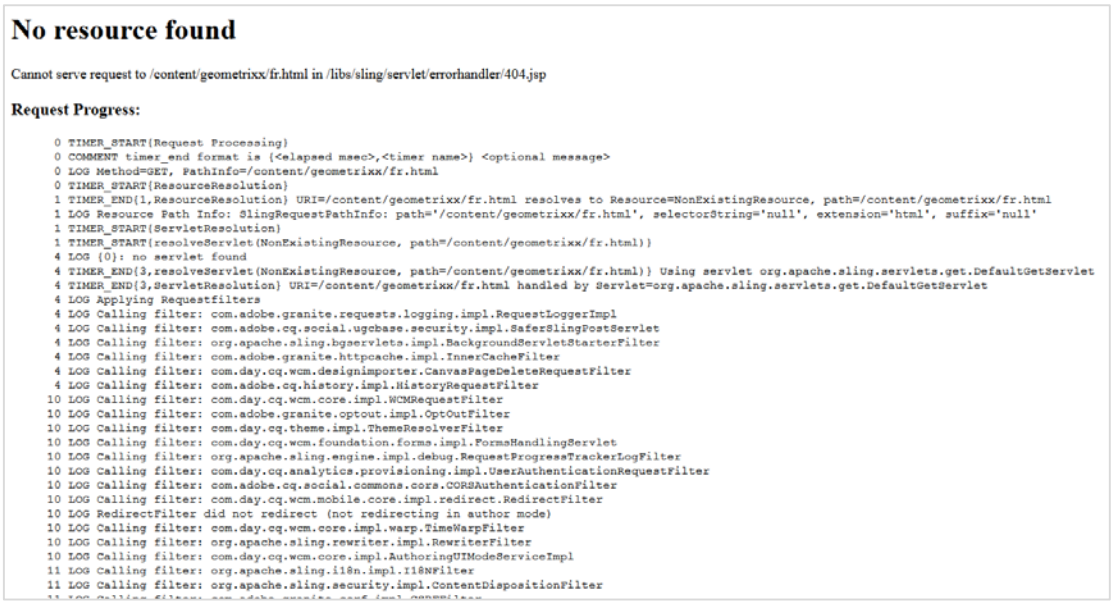
You should see the following information:



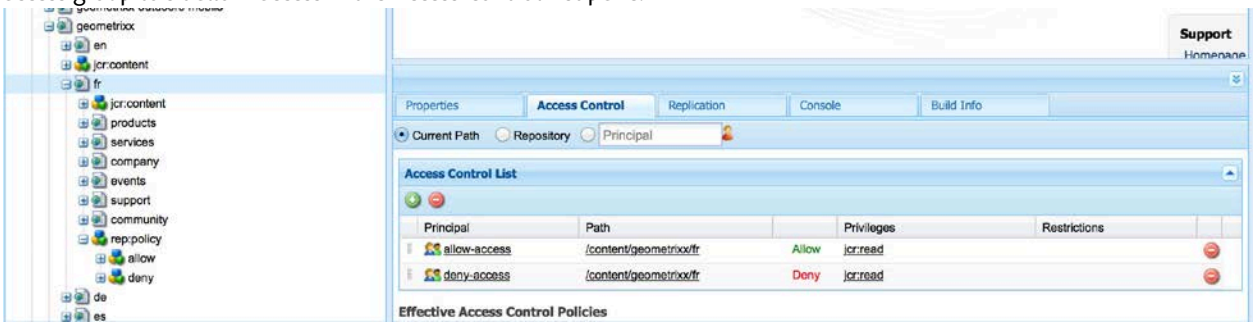
As you can see, the user john Doe does not have access to the French page because the deny rule is the one at the bottom of the ACL rules applied to the resource **/content/geometrixx/fr** and therefore takes precedence.

Navigate to http://localhost:4502, and impersonate as **johndoe**.
NOTE: Use another browser to test this, so that you don't have to clear your cache eveytime you make changes.

Open the page http://localhost:4502/content/geometrixx/fr.html in your browser. As the user john Doe does not have access to the fr page, you will see the following screen:



Navigate to CRXDE Lite and select the Access Control tab. Just as you were able to view the access control via the AC editor earlier, here you can change their order- granting the **allow-access** group priotity. With the **Current Path** checkmark selected, while /content/geometrixx/fr is selected in the heriarchy view, reposition the deny-access group to sit allow-access in the Access Control List pane.



Click **Save in CRXDDE Lite**. John Doe should now have access to the French page because the ACL rule on the bottom is the one granting him read acces to the node.

Refresh the session, and reopen the French page http://localhost:4502/content/geometrixx/fr.html. This time you are able to see the French page. Voila!

Now re-order the ACLs again so that the user does not have access to the French page.

2. Task – Automate ACLs

Create a class file named **ModifyPermissions** under the the training.core package at **training.core** > **src/main/java** > **com.adobe.training.core**.

Copy the following code into the class:

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

import java.util.NoSuchElementException;

import javax.jcr.RepositoryException;
import javax.jcr.Session;
import javax.jcr.security.AccessControlList;
import javax.jcr.security.AccessControlManager;
import javax.jcr.security.AccessControlPolicyIterator;
import javax.jcr.security.Privilege;

import org.apache.sling.jcr.api.SlingRepository;
import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.jackrabbit.api.security.JackrabbitAccessControlList;
import org.apache.jackrabbit.api.security.user.Authorizable;
import org.apache.jackrabbit.api.security.user.UserManager;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component
public class ModifyPermissions {
private static final String CONTENT_GEOMETRIXX_FR = "/content/geometrixx/fr";
private static final Logger LOGGER=  LoggerFactory.getLogger(ModifyPermissions.class);

@Reference
private SlingRepository repo;

@Activate
protected void activate(){
        LOGGER.info("ModifyPermissions activated");
        modifyPermissions();
}
```

```
private void modifyPermissions() {
        Session adminSession = null;
        try{
        adminSession= repo.loginService(“training”,null);
        UserManager userMgr=
        ((org.apache.jackrabbit.api.JackrabbitSession)adminSession).getUserManager();
        AccessControlManager accessControlManager = adminSession.getAccessControlManager();

        Authorizable denyAccess = userMgr.getAuthorizable("deny-access");

        AccessControlPolicyIterator policyIterator =
                accessControlManager.getApplicablePolicies(CONTENT_GEOMETRIXX_FR);
        AccessControlList acl;
        try{
                acl=(JackrabbitAccessControlList) policyIterator.nextAccessControlPolicy();

        }catch(NoSuchElementException nse){
                acl=(JackrabbitAccessControlList)
                accessControlManager.getPolicies(CONTENT_GEOMETRIXX_FR)[0];

        }

        Privilege[] privileges = {accessControlManager.privilegeFromName(Privilege.JCR_READ)};
        acl.addAccessControlEntry(denyAccess.getPrincipal(), privileges);
        accessControlManager.setPolicy(CONTENT_GEOMETRIXX_FR, acl);
        adminSession.save();
        }catch (RepositoryException e){
        LOGGER.error("************************Repo Exception", e);
        }finally{
        if (adminSession != null)
                adminSession.logout();
        }
}
}
```

This service grants read access to **/content/geometrixx/fr**, to the group named **deny-access**.

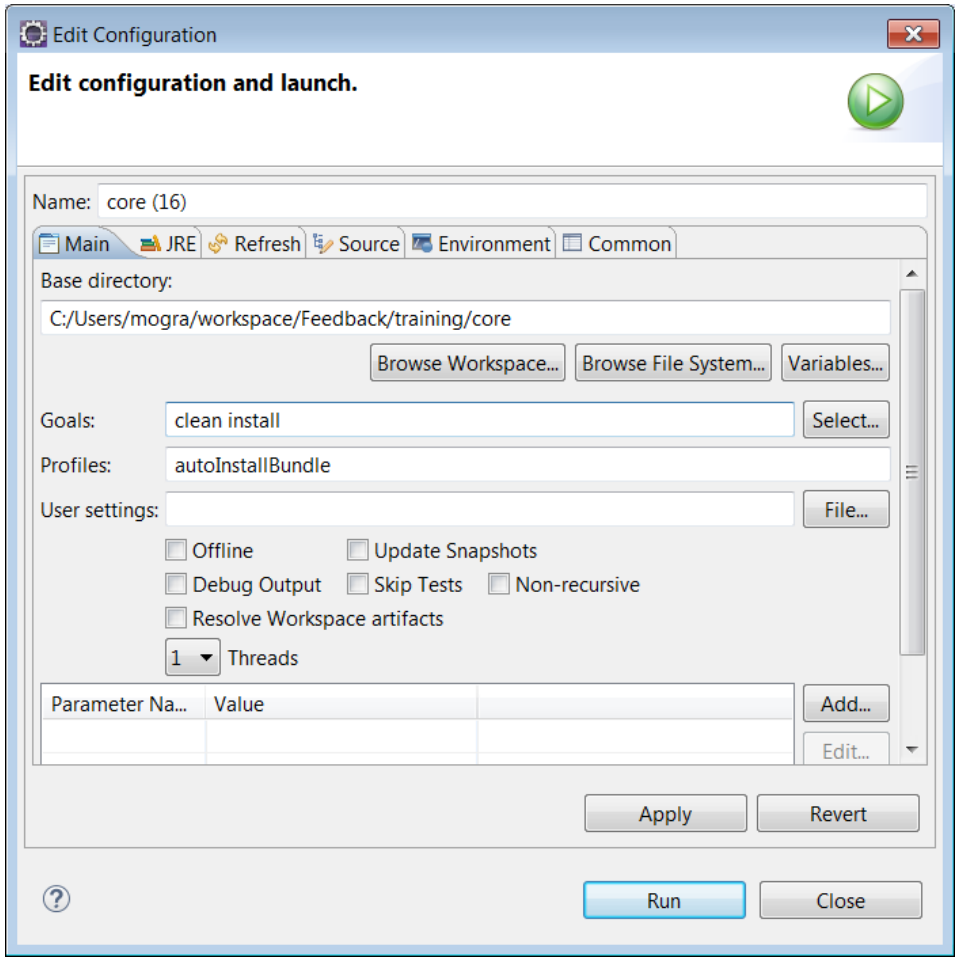
Your pom.xml already has the following dependency:

```
<dependency>
<groupId>org.apache.jackrabbit</groupId>
<artifactId>jackrabbit-api</artifactId>
<version>2.2.0</version>
<scope>provided</scope>
</dependency>
```

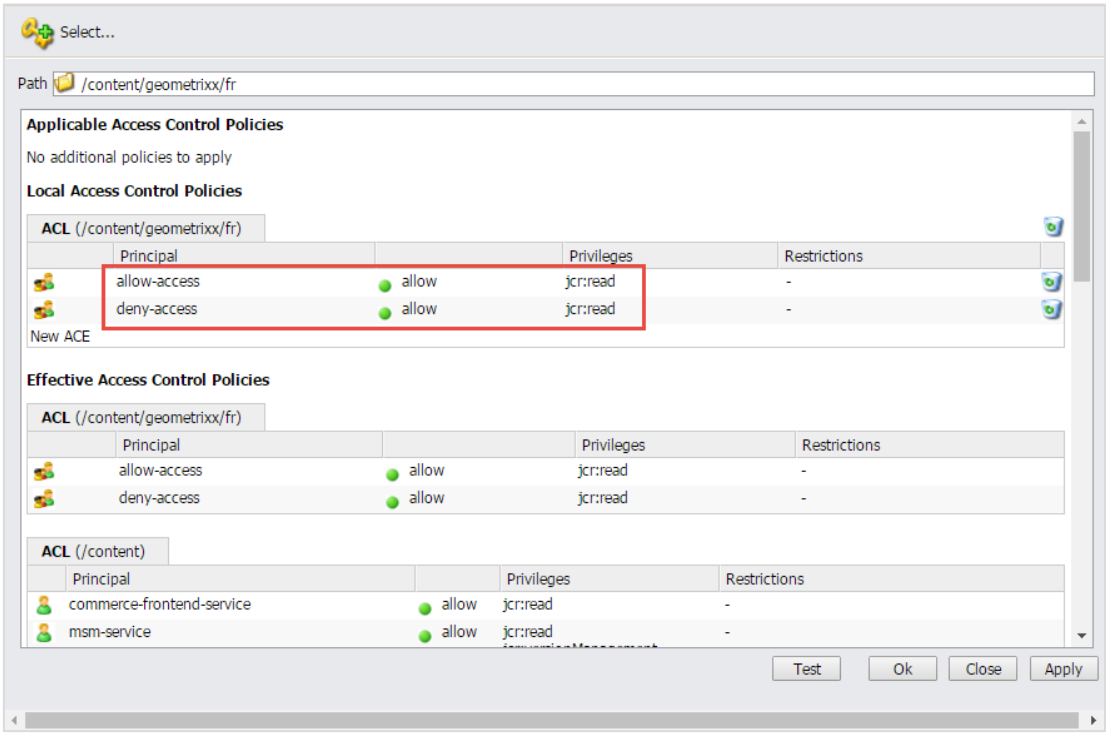
You cannot easily find this bundle in the Web Console. This bundle is exported by the system bundle (bundle 0). The service you implement will add the required permissions to the specified location on activation of the OSGi component.

Save your changes.

Deploy the bundle using **Run As** > **Maven build** and enter **clean install** for Goals.
NOTE: The autoInstallBundle is already set for this package.



Reopen the ACL editor, and check the permissions for the page: **/content/geometrixx/fr**.



Refresh and reopen the French page as john Doe. You are now able to access the page.

Scenario Conclusion

You have provided read access rights to a user to the French page of the Geometrixx site.

CHAPTER FOURTEEN: WRITING TESTS

Overview

This chapter looks at the available testing frameworks such as JUnit, Maven, Mockito, and PowerMock. You will also learn how to perform a Hobbes tests on your project. The chapter covers Jenkins—a tool that supports continuous integration, and is used widely in real time projects to build and test a project.

Objectives

By the end of this chapter, you will be able to:

- Run unit tests and functional tests on your project

Understanding Testing Frameworks

Software testing is done to ensure that the system performs as expected. The results of a test are compared with the expected outcomes to validate the functionalities of the system. Some of these tests can be extensive and repetitive. In such cases, you can use testing frameworks to automate the testing process.

A testing framework is a system with a set of rules for automation of software testing. This system makes use of function libraries, test data sources, object details, and various reusable modules.

NOTE:

- As a **mandatory** practice, you must place your Junit and Mockito testing code in the `src/test` package, or any package below.
- In the exercise, we will place our code in the `/java/com.adobe.training.core.models` package.

The JUnit Framework

JUnit is a testing framework for Java that is used to implement unit testing. It is an instance of the xUnit architecture for unit testing frameworks, and can be integrated with Eclipse or Maven. With JUnit testing, multiple tests can be run at the same time, which means you do not need human intervention to interpret test results.

Here is how you can use the JUnit framework to run tests:

Annotate a method with `@org.junit.Test`.

Whenever you want to check a value, import `org.junit.Assert.*` statically, call `assertTrue()`, and pass a Boolean that is true if the test succeeds.

To use Maven to run these tests, you can use the following commands and force Maven to recompile your project:
`mvn test` or `mvn clean test`

If you want to run only specific test cases, then you can use the following command:
`mvn –Dtest=MyTestsCase, MyOtherTestCase test`

The Mockito Framework

Mockito is an open source testing framework that allows the creation of mock objects in automated unit tests. Mock testing frameworks effectively fake some external dependencies so that the object being tested has a consistent interaction with its outside dependencies. Mockito intends to streamline the delivery of these external dependencies that are not subjects of the test.

Features of Mockito:

- Mocks concrete classes as well as interfaces
- Verification errors are clean—click on stack trace to see failed verification in test; click on exception's cause to navigate to actual interaction in code. Stack trace is always clean.
- Allows flexible verification in order
- Supports exact-number-of-times and at-least-once verification
- Flexible verification or stubbing using argument matchers

- Allows creating custom argument matchers or using existing hamcrest matchers

The PowerMock Framework

PowerMock uses a custom classloader and byte code manipulation to enable mocking of static methods, constructors, final classes and methods, private methods, removal of static initializers and more. By using a custom classloader, no changes need to be done to the IDE or continuous integration servers which simplifies adoption. PowerMock extends the existing APIs with a small number of methods and annotations to enable additional features.

To summarize, look at the primary differences between Mockito and PowerMock:

Mockito	PowerMock
Mockito does not include specific language characteristics like constructors or static methods for mocking.	PowerMock offers constructors and static methods to Mockito and other frameworks, through its individual class loader and bytecode management.
Mockito contains a JAR file in the classpath for supporting mocking APIs.	PowerMock is a Mockito API.
Mockito does not require any code to be executed before a test.	PowerMock includes code for preparation of tests.
Mockito does not support mocking of enum data types.	PowerMock supports mocking of enum data types.

Performing Unit Tests

In this chapter, you will look at the various unit tests that can be performed with Adobe Experience Manager. Unit testing is the process where the application is divided into units, and each unit is individually tested for its successful operation.

Writing Sling Tests

You can use Sling to perform a number of tests, including:

- JUnit tests using OSGi bundles in an OSGi system.
- Scriptable tests in a Sling instance, using any supported scripting language.
- Integration tests against a Sling instance that is started during the Maven build cycle or independently.

This section of the chapter has hands-on exercises for performing Sling server tests and scriptable server-side tests.

Performing Sling-based Tests on the Server

To perform a Sling test, you need the `org.apache.sling.junit.core` bundle. This bundle provides a service that allows bundles to register JUnit tests, and these tests are executed on the server by the `JUnitServlet` registered by default at `/system/sling/junit`. You should also note that this bundle is independent of Sling, and can work in other OSGi environments as well.

Performing Sling Scriptable Tests

To perform a Sling scriptable test, in addition to the above mentioned bundle, you also need the `org.apache.sling.junit.scriptable` bundle. While executing these tests, you need to make note of the following:

- A node with the `sling:Test` mixin is a scriptable test node.
- Scriptable test nodes are executed only if they belong to the Sling's `ResourceResolver` search path. For example, `/libs` or `/apps`.

Perform Task – Unit testing using JUnit and Maven, from the Lab Activity section.

Perform Task – Perform tests using Mockito, from the Lab Activity section.

Perform Task – Run Sling-based tests on the server, from the Lab Activity section.

Perform Task – Run scriptable server-side tests, from the Lab Activity section.

NOTE: To execute a test, the scriptable tests provider makes an HTTP request to the test node's path, with a `.test.txt` selector and extension, and expects the output to contain only the string TEST_PASSED. Empty lines and comment lines starting with a hash sign (#) are ignored in the output, and other lines are reported as failures.

Performing Functional Tests

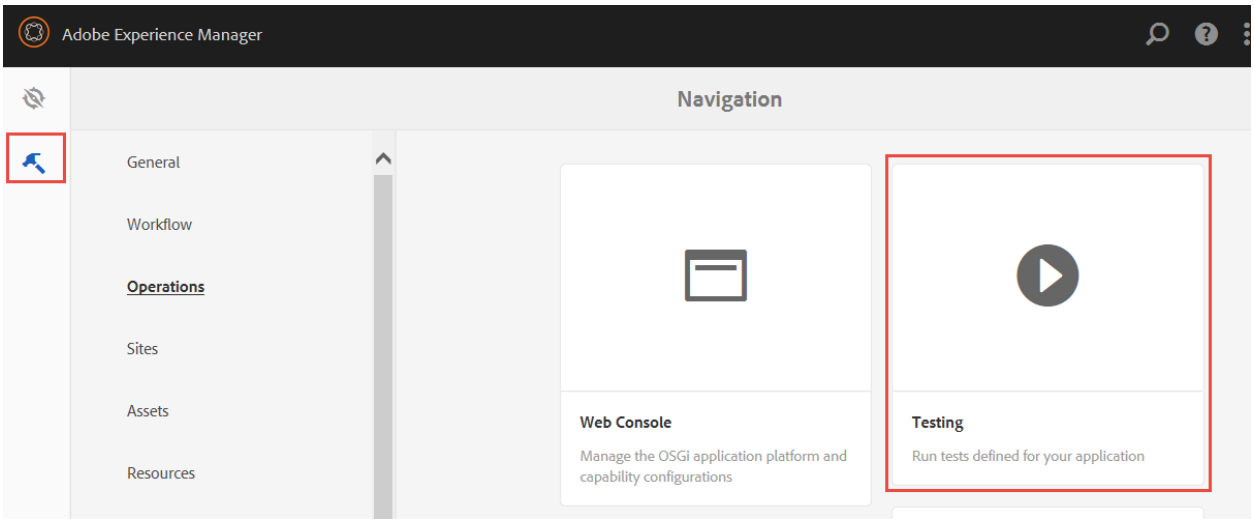
While unit tests check each small unit of an application, functional tests are more involved in evaluating whether the web application performs according to the business requirements. These requirements may be in the form of functional specifications or design specifications.

Functional testing does not involve testing of back-end code, but rather the front-end is tested to check if it reacts properly to the user input. Using Test-Driven Development (TDD), you will write a test first, then write the simplest code possible to implement the desired functionality. You will then refactor the code to meet production standards.

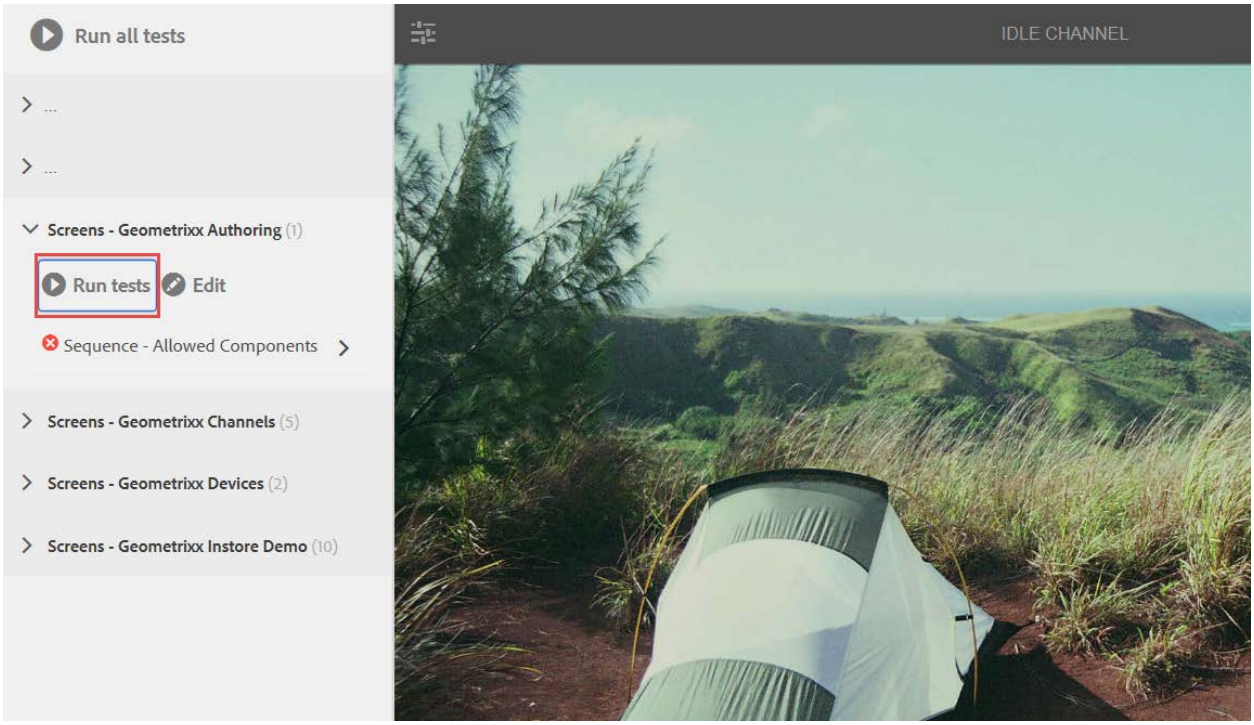
Performing Hobbes Tests

Functional testing is a form of testing that is dedicated to the user experience. Before you can publish your site, you need to ensure that every component in it is fixed and in full working order. Adobe Experience Manager has an out-of-the-box functional testing framework embedded in its UI, and it uses the Hobbes testing framework as its base. This framework provides a streamlined experience for ensuring the usability of interactive elements on your site.

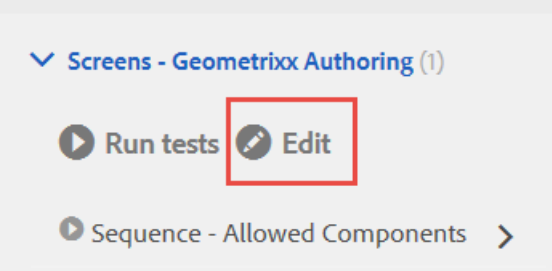
You can access this framework through Tools > Testing in AEM:



In a new browser tab, you can perform Hobbes testing on a particular screen (site) or run all test suites. You can select a test suite and run a test case within it:



You can also edit the test cases in a new browser tab by clicking the pencil icon in the toolbar below a test suite. This opens the corresponding code file in CRXDE Lite.



Now that you know how to navigate through the test suites and test scripts, let us see how to create and configure a new test suite.

Perform Task – Create a test suite in Adobe Experience Manager in the Lab Activity section.

Using Jenkins for Continuous Integration

Continuous integration is a process where all development work is integrated to a system at a predefined time, and is automatically tested and built. The purpose of using continuous integration is to identify and catch errors early in the process.

Jenkins is an open source continuous integration tool used for build automation. Its main functionality is to execute a predefined set of steps based on a trigger such as a change in version control system, or a time based trigger such as creating a build every 30 minutes.

The steps to be executed could be any of the following:

- Perform a software build with Apache Maven.
- Run a shell script.
- Archive the build result.
- Start the integration tests.

With Jenkins, you can:

- Monitor the execution of steps.
- Stop the process if any of the steps fail.
- Notify respective users of the status of the build, whether the build is a success or failure.

If you are working on an Adobe Experience Manager project, here is how you can use Jenkins:

Work on the feature or bug fix.

Test it on your local instance.

Commit or push the changes to the central repository; for example, SVN or GIT.

Have Jenkins configured to kick start the build production, and deploy packages to the Adobe Experience Manager Integration Server.

Move the feature/bug-fix to the Quality Assurance (QA) team.

QA team will pick up the feature/bug-fix, and test code changes on the Integration Server.

NOTE: If the build is not successful, Jenkins can identify the faulty source code that was checked in to the repository, and then notify the developer of the error. This needs to be fixed before the next build process can take place.

Chapter 14 Lab Activity

Scenario

You are at a stage when the work that you have done in your project needs to be tested before you can push them to the central repository; both in terms of unit tests, as well as functional tests.

Challenge

Identifying the appropriate test framework.

Overview

Create tests using Junit, Mockito, and sling automated test cases.

Pre-requisites

- Existing project—**trainingproject** from the USB Contents
- AEM Author instance

Steps

1. Task - Unit testing using JUnit and Maven

   In Eclipse, open the training project.

Create a class named **ProcessContent** in **com.adobe.training.core**:

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

import javax.jcr.Node;
import javax.jcr.RepositoryException;
import javax.jcr.Session;


// This class does not contain any test methods.

public class ProcessContent {


    private static final String CONTENT_NODENAME = "content";

    public String stripNonLettersOrNumbers(String in) {
        if(in != null) {
            return in.replaceAll( "[^\\p{L}\\p{N}]", "");
        } else {
            return null;
        }

    }


    public String getContentPath(Session session)  throws RepositoryException {

           Node rootNode = session.getRootNode();
        if (rootNode.hasNode(getContentNodename()))  {

           Node contentNode = rootNode.getNode(getContentNodename());
            return contentNode.getPath();

        } else {
            return rootNode.getPath();
        }
    }


    // the getter
           public static String getContentNodename() {
                      return CONTENT_NODENAME;
           }

}
```

This code contains three methods:

- **stripNonLettersOrNumbers**: Does not depend on JCR
- **getContentPath**: Depends on JCR environment specifics
- **getContentNodename:** (getter) Depends on JCR environment specifics

In the **com.adobe.training.core.models** package, create a new java class **MavenProcessContentTest .java** with the following code:

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

import static org.junit.Assert.assertEquals;
import org.junit.Test;
import com.adobe.training.core.ProcessContent;

public class MavenProcessContentTest {
    @Test
    public void testStripNonLettersOrNumbers() {
        ProcessContent pc = new ProcessContent();
        assertEquals("abc1", pc.stripNonLettersOrNumbers("a_b!c.1"));
    }
}
```

Note that you have added a method to test **ProcessContent.stripNonLettersOrNumbers**. In this first example, you are using pure JUnit. Notice the **@Test** annotation that informs JUnit the procedure to test, as well as the **assertEquals** method, which will tell you if your class behaved as expected.

Run **mvn clean test** to execute the test. You can also just do a **Maven install** on the training.core package. You should see the following result:

```
-------------------------------------------------------
TESTS
-------------------------------------------------------
Running com.adobe.training.core.MavenProcessContentTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.049 sec
Results :
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

NOTE: If the test does not run, perform this step—select the **MavenProcessContentTest.java** file, and run the JUnit test. To do this, right-click the java file, go to **Run As**, and select **JUnit Test**. A JUnit console pops up with the results of the test. When you re-run the mvn clean test, it should work.

Change the testing part to force an error in the test. Change the code as follows:

```
public class MavenProcessContentTest {
       @Test
       public void testStripNonLettersOrNumbers() {
       ProcessContent pc = new ProcessContent();
       assertEquals("abbc1", pc.stripNonLettersOrNumbers("a_b!c.1"));
       }
}
```

Test again. Now you should see an error:

```
-------------------------------------------------------
TESTS
-------------------------------------------------------
Running com.adobe.training.core.MavenProcessContentTest
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.058 sec <<< FAILURE!
Results :
Failed tests: testStripNonLettersOrNumbers(com.adobe.training.core.MavenTestSamplesTest):
expected:<ab[b]c1> but was:<ab[]c1>
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
```

And the project refuses to build:

```
[INFO] ------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------
```

2. Task - Perform tests using Mockito

In Eclipse, open the training project. We already created a class, **ProcessContent.java**, in **com.adobe.training.core** with the following code:

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

import javax.jcr.Node;
import javax.jcr.RepositoryException;
import javax.jcr.Session;

public class ProcessContent {

    public static final String CONTENT_NODENAME = "content";

    public String stripNonLettersOrNumbers(String in) {
        if(in != null) {
            return in.replaceAll( "[^\\p{L}\\p{N}]", "");
        } else {
            return null;
        }

    }
    public String getContentPath(Session session)  throws RepositoryException {
        Node rootNode = session.getRootNode();
        if (rootNode.hasNode(CONTENT_NODENAME))  {
            Node contentNode = rootNode.getNode(CONTENT_NODENAME);
            return contentNode.getPath();

        } else {
            return rootNode.getPath();
        }
    }

}
```

In the **com.adobe.training.core.models** package, create a new java class **testProcessContent** with the following code:

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

import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

import com.adobe.training.core.ProcessContent;

import org.junit.Test;

import javax.jcr.Node;
import javax.jcr.RepositoryException;
import javax.jcr.Session;

public class testProcessContent {

    @Test
    public void testGetContentPath()
        throws RepositoryException {
        //create a mock repository session and nodes
        Session JCR_SESSION_MOCK = mock(Session.class);
        Node ROOT_NODE_MOCK = mock(Node.class);
        Node CONTENT_NODE_MOCK = mock(Node.class);

        //prepare the expected method calls
        when(JCR_SESSION_MOCK.getRootNode()).thenReturn(ROOT_NODE_MOCK);

when(ROOT_NODE_MOCK.hasNode(ProcessContent.getContentNodename())).thenReturn(true);
when(ROOT_NODE_MOCK.getNode(ProcessContent.getContentNodename())).thenReturn(CONTENT_N
ODE_MOCK);
        when(CONTENT_NODE_MOCK.getPath()).thenReturn("/content");

        //create an instance of the process content object
        ProcessContent pc = new ProcessContent();

        //run method test
        assertEquals("/content", pc.getContentPath(JCR_SESSION_MOCK));
    }
}
```

Note that we have added a method to test **testGetContentPath**. In this example, we are using Mockito. Notice the **@Test** annotation that tells JUnit the procedure to test, as well as the **assertEquals** method, which will tell us if our class behaved as expected.

Run **mvn clean test** to execute the test. You should see the following result:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.adobe.training.core.models.TestHelloWorldModel
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.356 sec - in
com.adobe.training.core.models.TestHelloWorldModel
Running testProcessContent.testProcessContent
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.076 sec - in
testProcessContent.testProcessContent

Results :
```

```
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
```

3. Task - Run scriptable server-side tests

Scriptable server-side tests are usually made by creating a node with the **mixin** type **sling:Test**. This node has to have a property **sling:resourceType** pointing to the script used for testing purposes. To execute the test, ask for this resource (script) in Adobe Experience Manager.

Open CRXDE Lite, and navigate to **apps** > **trainingproject**. Add a new node under the trainingproject folder named **tests** of type **sling:Folder**.

NOTE: If the node already exists, ignore this step.

Under the **tests** folder, create a node named **testpagemanager** of type **nt:unstructured**.

Add the following property to the **testpagemanager** node:

Name	Type	Value
sling:resourceType	String	trainingproject/tests

Select the **testpagemanager** node, and click **Mixins**...

Click the **+**, and select **sling:Test** from the drop-down list.

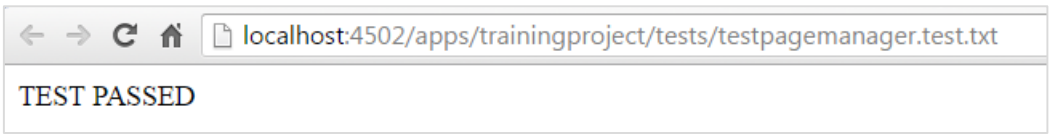
NOTE: You may have to refresh the browser screen before the **sling:Test** mixin appears in the drop-down list. If it is still not available, go to the Web Console and install the **org.apache.sling.junit.core-1.0.10.jar** and **org.apache.sling.junit.scriptable-1.0.10.jar** file, provided with the lab files.

Save your changes.

The test script will test if **/libs/foundation/global.jsp** puts the **PageManager** object on the request. Create a JSP called **test.txt.jsp** (notice the selector and the extension used) under the tests folder, and add the following code:

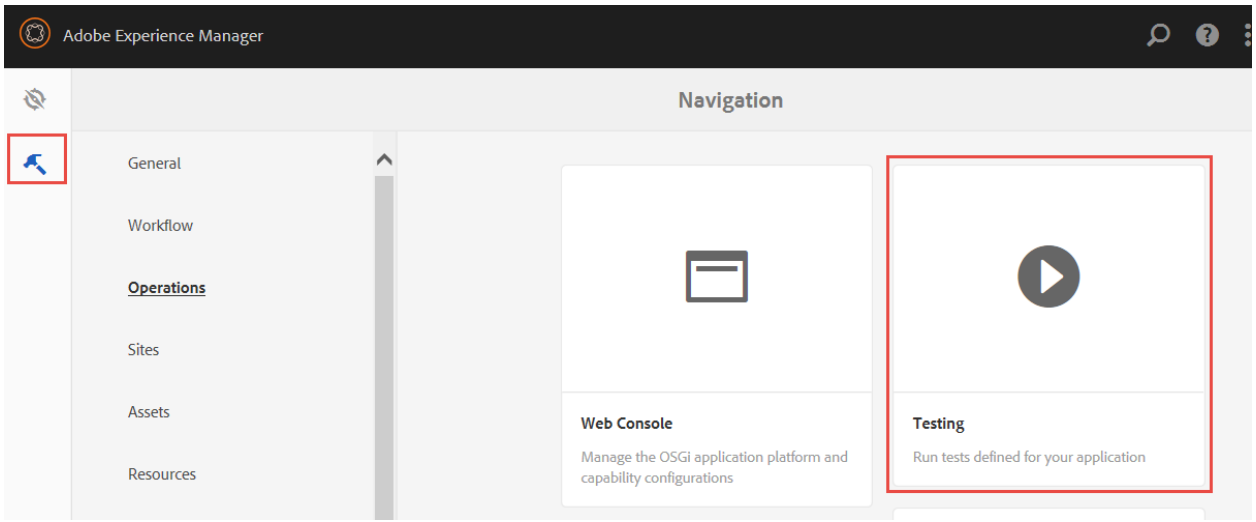
```
<%@include file = "/libs/foundation/global.jsp" %>
<%
if (pageManager != null) {
%> TEST PASSED <%
} else {
%> no pageManager on request <%
}
%>
```

Execute the test by browsing to: http://localhost:4502/apps/trainingproject/tests/testpagemanager.test.txt

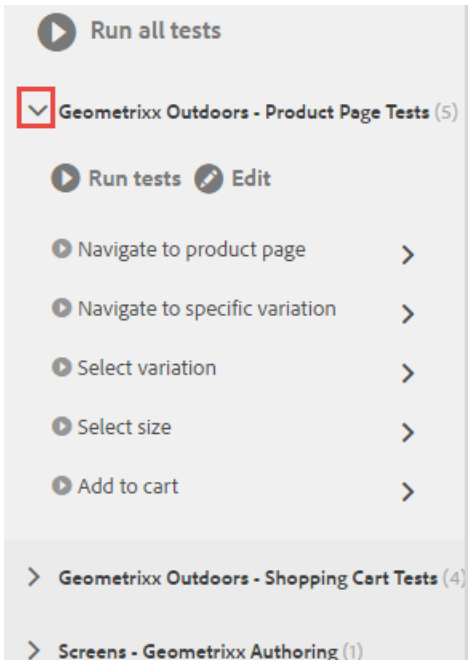


4. Task - Create a test suite in Adobe Experience Manager

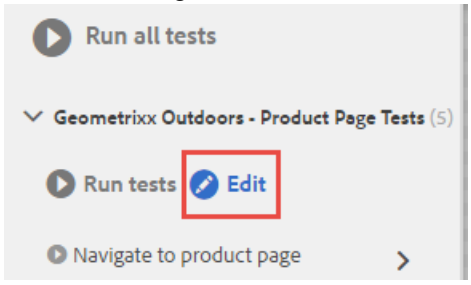
Navigate to Tools > Operations > Testing in AEM:



In the new browser tab that opens, expand the **Geometrixx Outdoors – Product Page tests (5)** node:



Click **Edit** to navigate in a new tab to the test scripts in CRXDE Lite:



In the tests folder that automatically opens in CRXDE Lite, create a new file named **TrainingPageTests.js**.

Create a new test suite by adding the following code at the top of the **TrainingPageTests.js** file. A test suite has many test cases. In other words, a test suite is a group of related test cases.

```
new hobs.TestSuite("Training Page tests", {path:"/apps/geometrixx-
outdoors/tests/ProductPageTests.js", register: true})
```

Add a test case under the test suite as follows on the next line:

```
.addTestCase(new hobs.TestCase("First training test case")
.navigateTo("/content/geometrixx-outdoors/en/men/shorts/jola-summer.html")
.asserts.location("/content/geometrixx-outdoors/en/men/shorts/jola-summer.html",
true)
.asserts.visible("article.product[data-sku='mnapjs.1-S']", true)
.asserts.visible("article.product[data-sku='mnapjs.2-S']", false)
)
```

Analyze the test case. NOTE: This is what the code is doing:

Instruct the framework to navigate to the **jola-summer.html** page.

Check if the location changes to **jola-summer.html**. The test case is marked as fail if it does not.

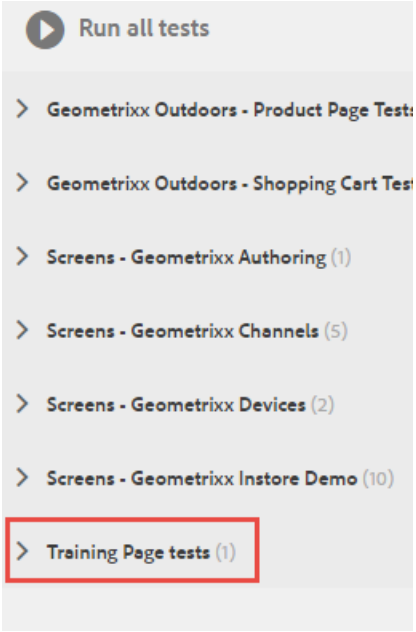
Check if the SKU attribute, 1-S, is present. If yes, mark as Pass.

Check if the SKU attribute, 2-S, is present. If yes, mark as Fail. You just want to see how a test case can fail.

Save the file in CRXDE Lite by clicking **Save All.**

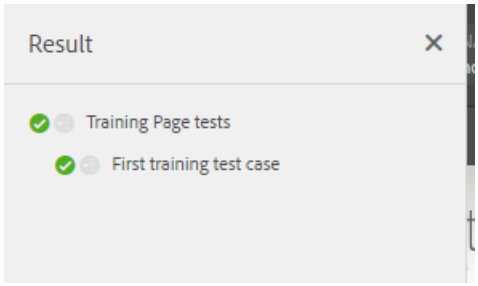
Open **js.txt**, and add the following entry**: TrainingPageTests.js** (on line 3). **Save all.**

Return to the product page, and refresh your browser.
Observe that the Test Suite you added is displayed there.



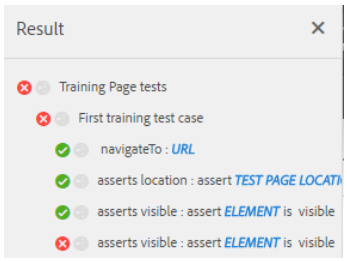
Expand the test suite and click **Run tests**. Note that the page is refreshing with **jola-summer.html**.

Click on the title **First training test case**. This opens the Result pane. Now click on **First training test case** in the result pane, to see the individual test results. Observe the test results (both passed):



Using CRXDE Lite, change the last line of code in **TrainingPageTests.js** to assert that the product IS visible: .asserts.visible("article.product[data-sku='mnapjs.2-S']", true).

Save the changes (Save all) in CRXDE Lite. Refresh the testing page, then run the tests again. Note the errors generated:



Use vlt, or Import from server in Eclipse to update your file system with the new test code in the repository, so that it is under version control.

Scenario Conclusion

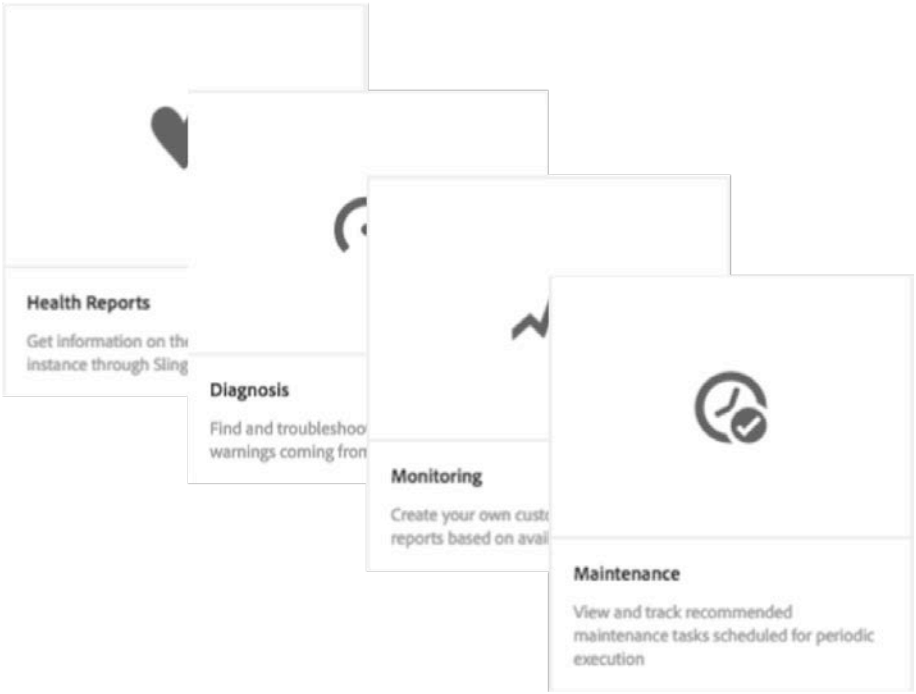
You have performed the following tests on your code:

- JUnit tests
- Maven tests
- Tests using Mockito
- Sling-based tests
- Scriptable server-side tests
- You have also created a test suite in Adobe Experience Manager

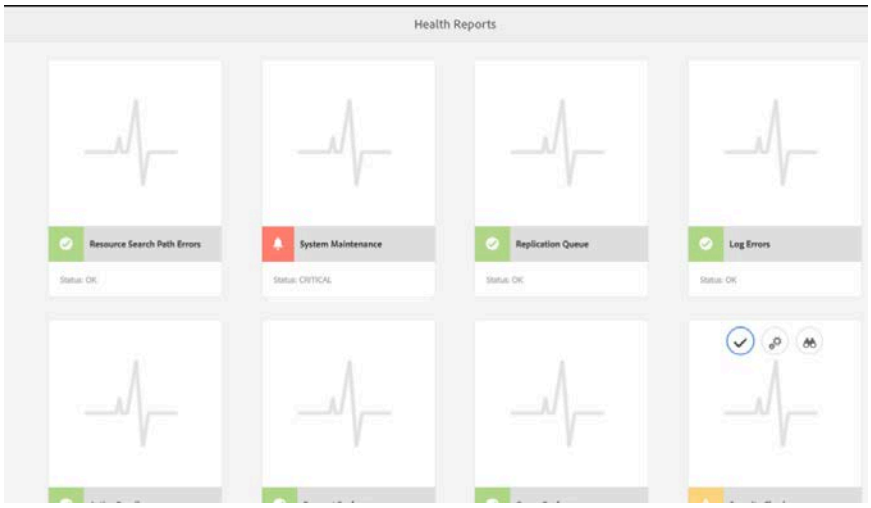
APPENDIX: OPERATIONS DASHBOARD

Overview

The Operations Dashboard in Adobe Experience Manager 6.2 helps system operators to monitor the Adobe Experience Manager system health at a glance. The Dashboard also provides auto-generated diagnosis information on relevant aspects of Adobe Experience Manager and allows configuring and running of self-contained maintenance automation to reduce project operations and support cases significantly. The Operations Dashboard can be extended with custom health checks and maintenance tasks. Further, Operations Dashboard data can be accessed from external monitoring tools via JMX. You can access the Dashboard from AEM Home > Tools > Operations or by using OmniSearch by pressing "/" and entering "Operations".



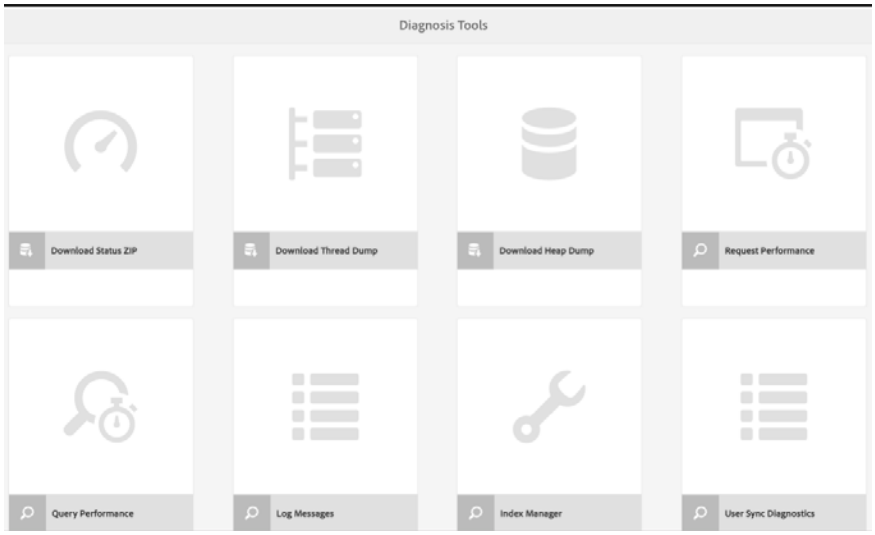
Health Reports
The Health Report system provides information on the health of an AEM instance through Sling Health Checks. This can be done via either OSGI, JMX or HTTP requests (via JSON). It offers measurements and threshold of certain configurable counters and in some cases, will offer information on how to resolve the issue.



Diagnosis Tools
Diagnosis Tools that can help finding and troubleshooting root causes of the warnings coming from the Health Check Dashboard, as well as providing important debug information for system operators.

Amongst the most important features are:

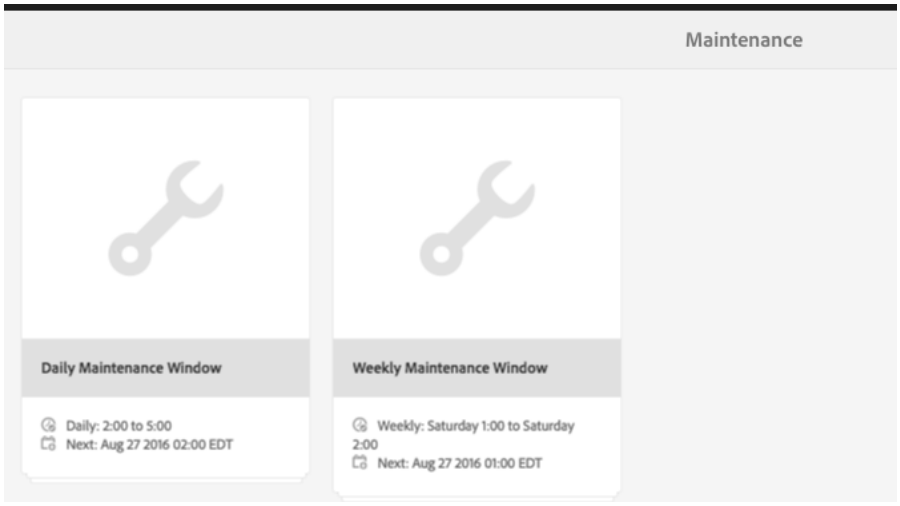
- A log message analyzer
- The ability to access heap and thread dumps
- Requests and query performance analyzers



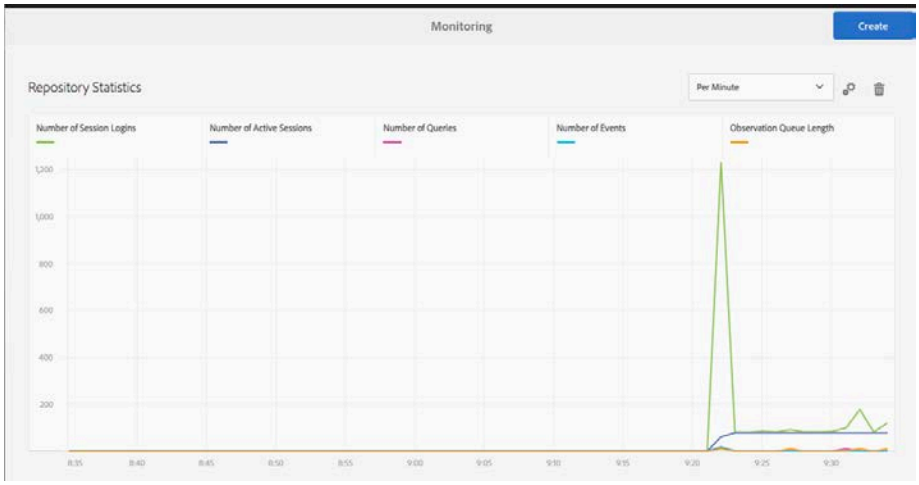
Automated Maintenance Tasks
The Automated Maintenance Tasks page is a place where you can view and track recommended maintenance tasks scheduled for periodic execution. The tasks are integrated with the Health Check system and their execution has minimal impact on system performance. The tasks can also be manually executed from the interface.

AEM 6 ships with a default set of automated maintenance tasks, including Version Purge, Workflow Purge, and Revision Clean Up among others,

Custom maintenance tasks can be implemented as OSGi services. The maintenance task infrastructure is based on Apache Sling's job handling.



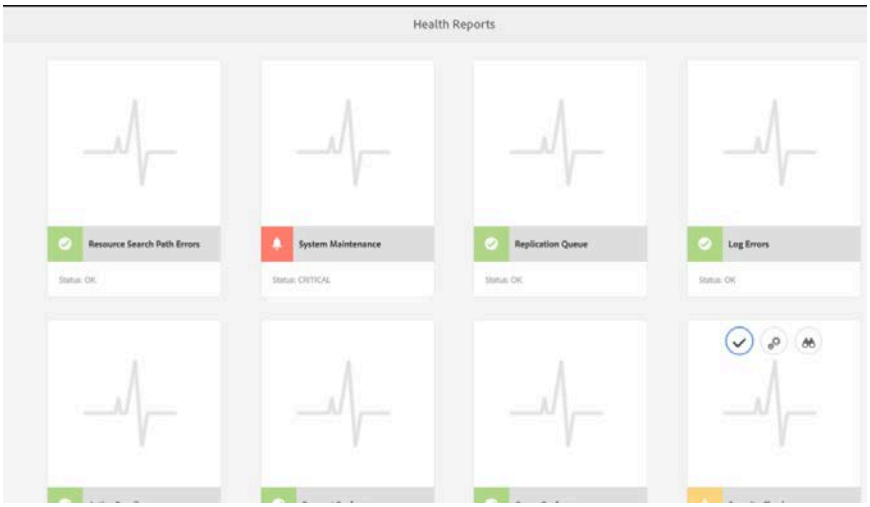
Monitoring
Health Check Dashboard can integrate with Nagios via the Granite JMX Mbeans.



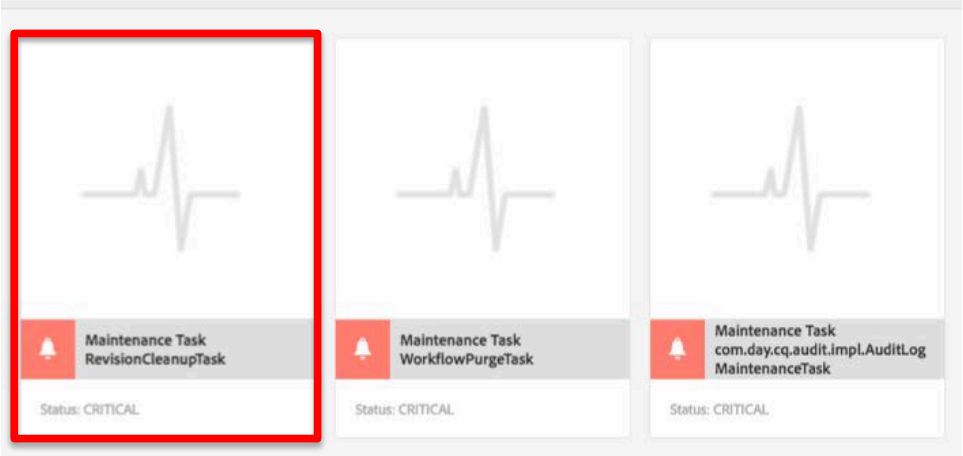
For more information about the Operations Dashboard, see https://docs.adobe.com/docs/en/aem/6-2/administer/operations/operations-dashboard.html.

Exercise 1: Diagnosing an issue using the Operations Dashboard

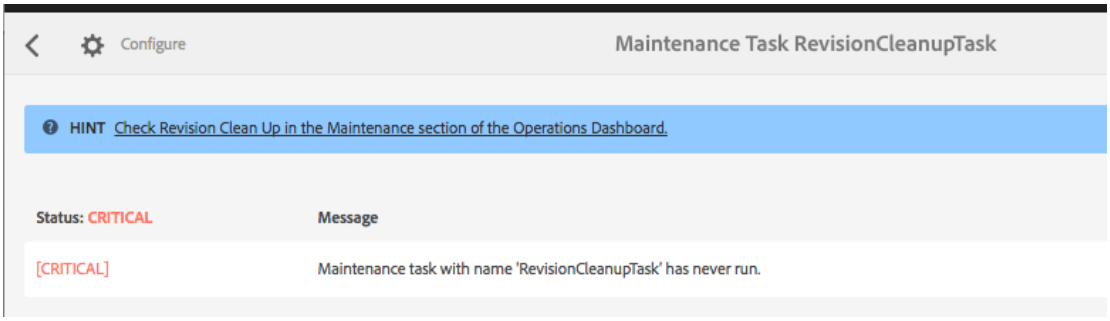
1. Navigate to **AEM Home** > **Tools** > **Operations**. Select **Health Reports**.



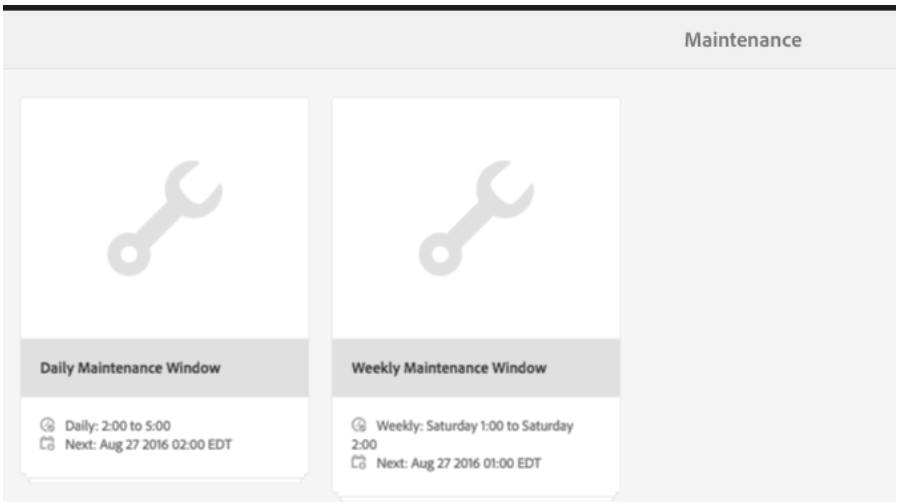
2. Notice that the System Maintenance card is in a CRITICAL state. You can use the Dashboard functionality to diagnose and treat the problem.
3. Double-click on the System Maintenance card to display the next level of information. Notice that all the cards are in a CRITICAL state.



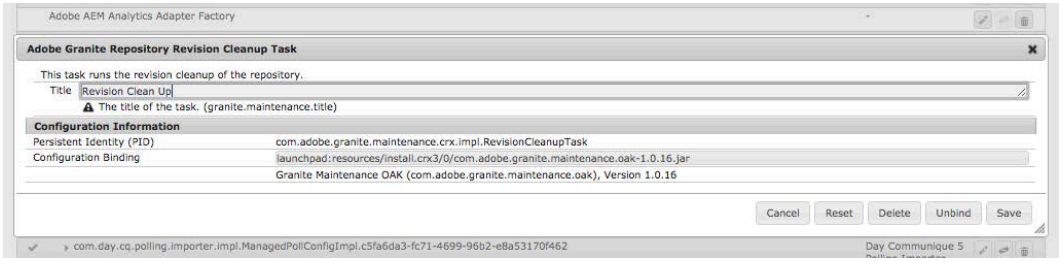
4. Double-click on the RevisionCleanupTask to display details of the problem. Notice that the system gives you a diagnosis and a Hint for solving the problem.



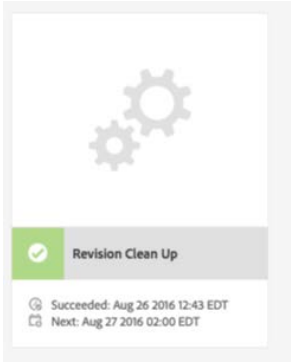
5. Click on the Hint, which takes you to the Automated Maintenance Tasks page. Notice there are daily and weekly maintenance tasks.



6. Click in the Daily Maintenance Window to see the list of scheduled Daily tasks.
7. Hover on the RevisionCleanup Task. Notice that you can Run the task and configure the task.



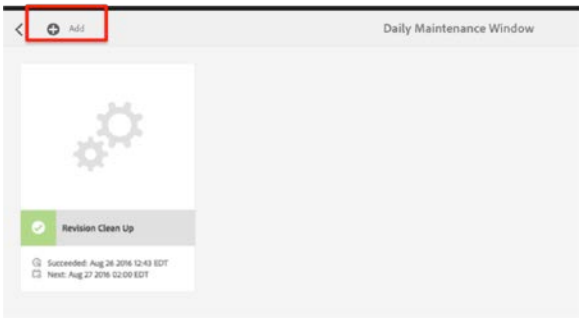
8. To launch the task, click the **play** button. While the task is running, there will be a yellow indicator and then when it is complete, a green (success) or red (failure) indicator.



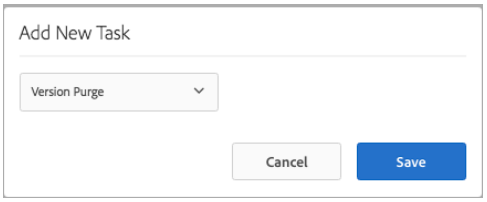
Congratulations! You now know how to use the Operations Dashboard to diagnose and fix an issue.

Exercise 2: Add a New Task to the Maintenance Schedule

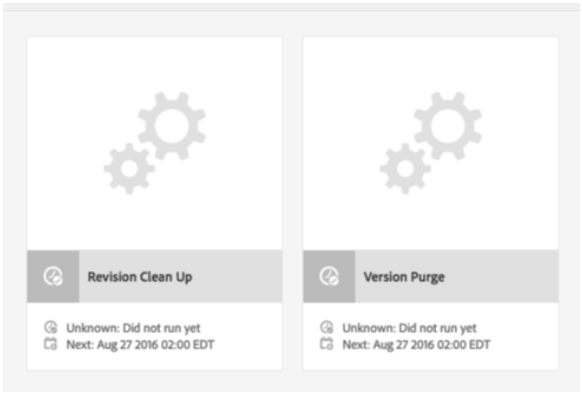
1. Navigate to **AEM Hone** > **Tools** > **Operations** > **Maintenance** > **Daily Maintenance**.



2. Click on the **"+"** icon to add a new task.

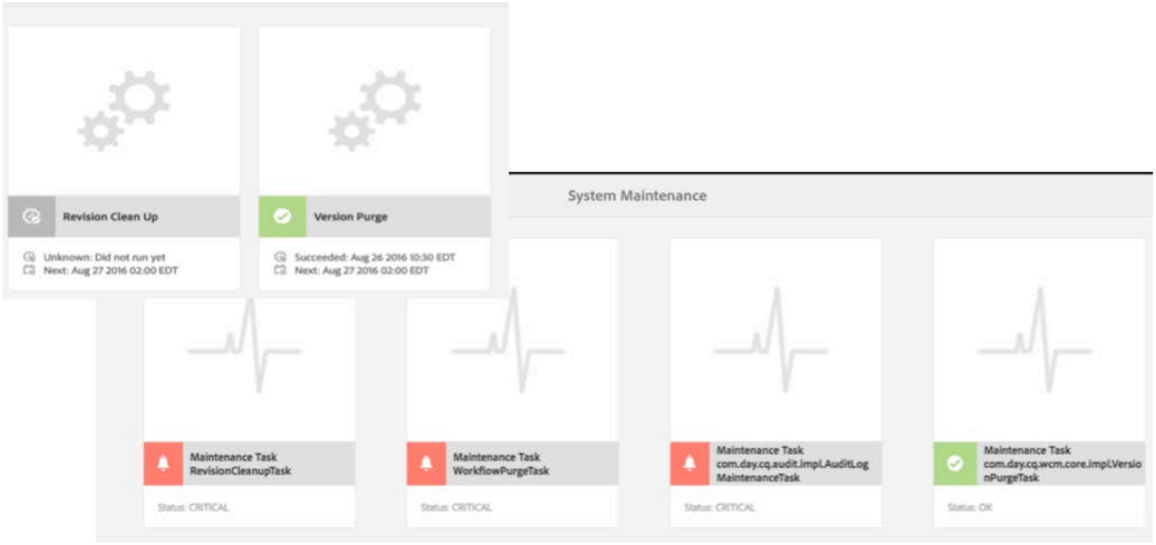


3. Select the **Version Purge** task and click **Done**.
4. A new card will appear in the daily maintenance page.



5. Before you run the Version Purge task, return to the System Maintenance page, AEM Home > Tools > Operations > Health Reports > System Maintenance. Notice that a new CRITICAL task has shown up.

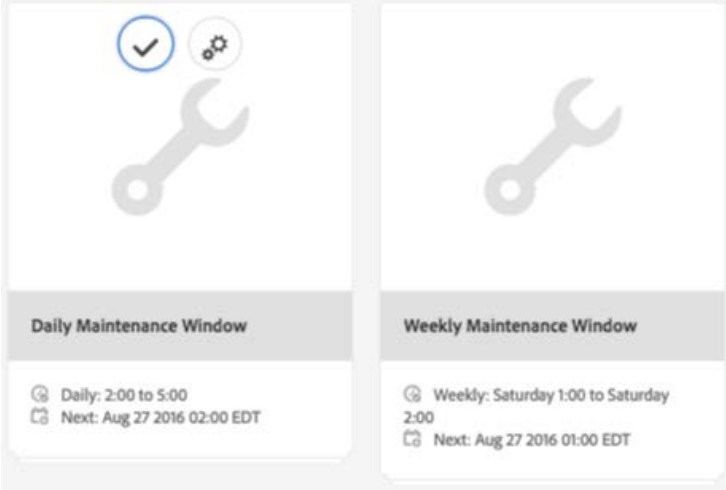
6. Return to the daily maintenance page and run the Version Purge task by clicking on the **play** button.



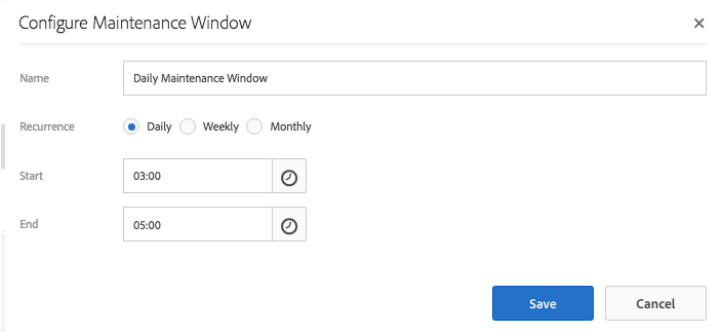
Congratulations! You now know how to add a new task to the automated maintenance schedule.

Exercise 3 Modify the Maintenance Schedule

1. Navigate to **AEM Home** > **Tools** > **Operations** > **Maintenance**.



2. Hover over the **Daily Maintenance Window** and click the **Configure** button.

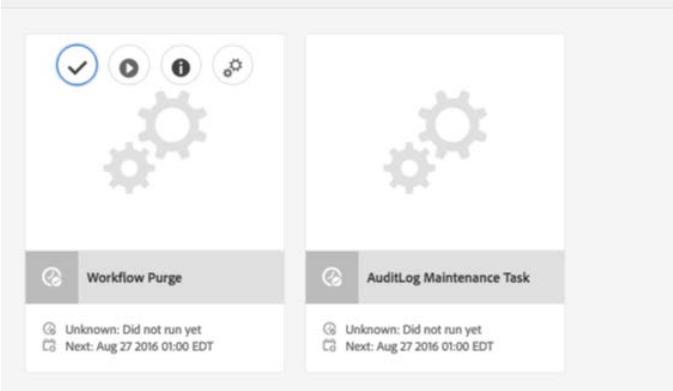


3. Modify the **Start** time to 3:00 and Click Save.

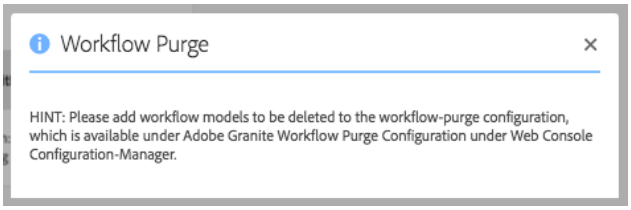
Congratulations! You now know how to modify the Automated Maintenance Schedules.

Exercise 4 Configure the Workflow Purge Task.

1. Navigate to and open the **Weekly Maintenance Window (Tools > Operations > Maintenance > Weekly Maintenance window**)

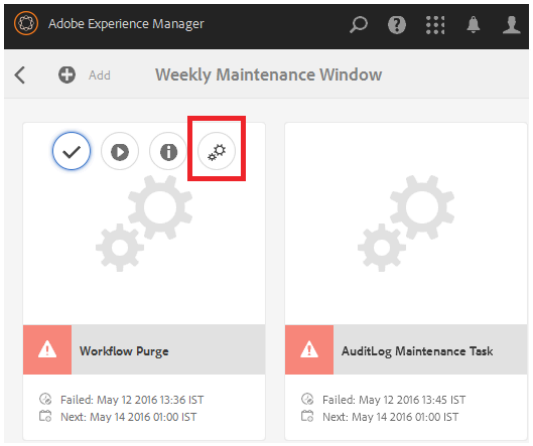


2. Hover over the Workflow Purge task and click **Info.**

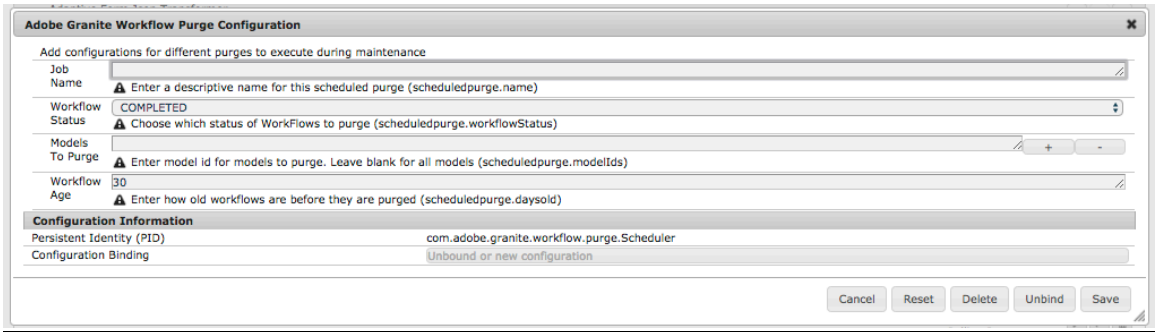


3. Close the Info window.

To configure the Workflow Purge, you must add the workflow models to be purged to the Workflow Purge Scheduler. If you run the task before specifying the workflow models to be purged, the task will fail.



4. Navigate to **AEM Home > Tools > Web Console > Configuration Manager**. Find the Workflow Purge Scheduler: com.adobe.granite.workflow.purge.Scheduler. http://localhost:4502/system/console/ configMgr/com.adobe.granite.workflow.purge.Scheduler.



Notice that the configuration is empty. To configure the Workflow Purge Scheduler, you must create configuration nodes in the repository.

5. Navigate to **AEM Home > Tools > CRXDE Lite** (http://localhost:4502/crx/de/)

6. If the */apps/geometrixx/config.author* folder does not already exist, do the following (otherwise, go to Step 7):
    a. Right click the */apps/geometrixx* folder and choose **Create... Create Node**.
    b. Enter the following values in the **Create Node** dialog:

       Name: config.author

       Type: sling:Folder

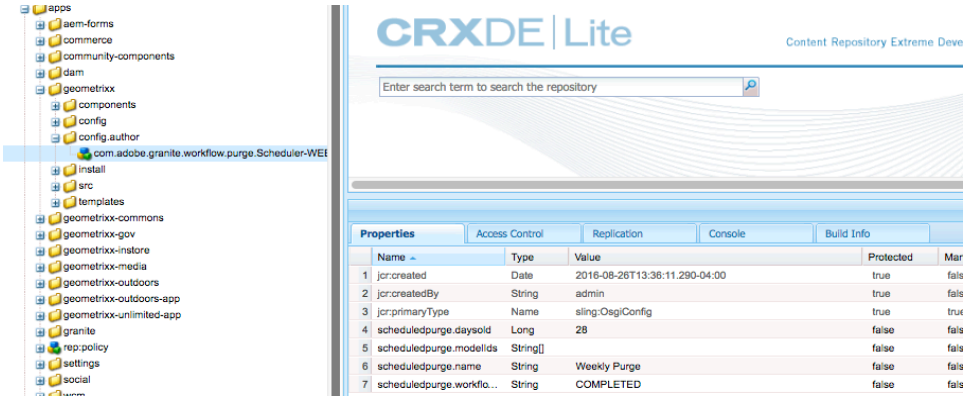
    c. Click **Save All**.
7. Right-click */apps/geometrixx/config.author* and select **Create > Create Node**.
8. Enter the following values in the **Create Node** dialog and click **OK**:

   Name: com.adobe.granite.workflow.purge.Scheduler-WEEKLY

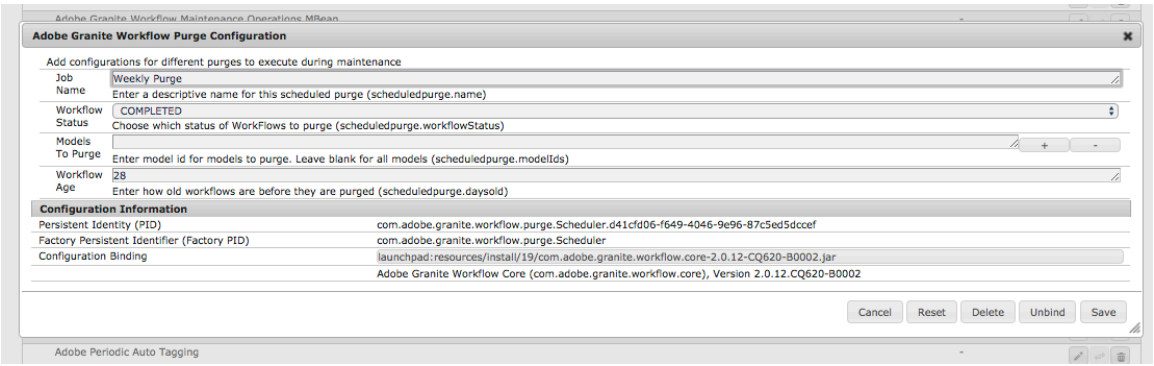
   Type: sling:OsgiConfig

9. **Save**.
10. Add the following properties on the Properties tab. As you add each property, click **Save All**. Refer to the following table for the four properties you will add:

Name	Type	VALUE
scheduledpurge.name	String	Weekly Purge
scheduledpurge.workflowStatus	String	COMPLETED
scheduledpurge.modelIds	String[]	""(leave this blank)
scheduledpurge.daysold	Long	28

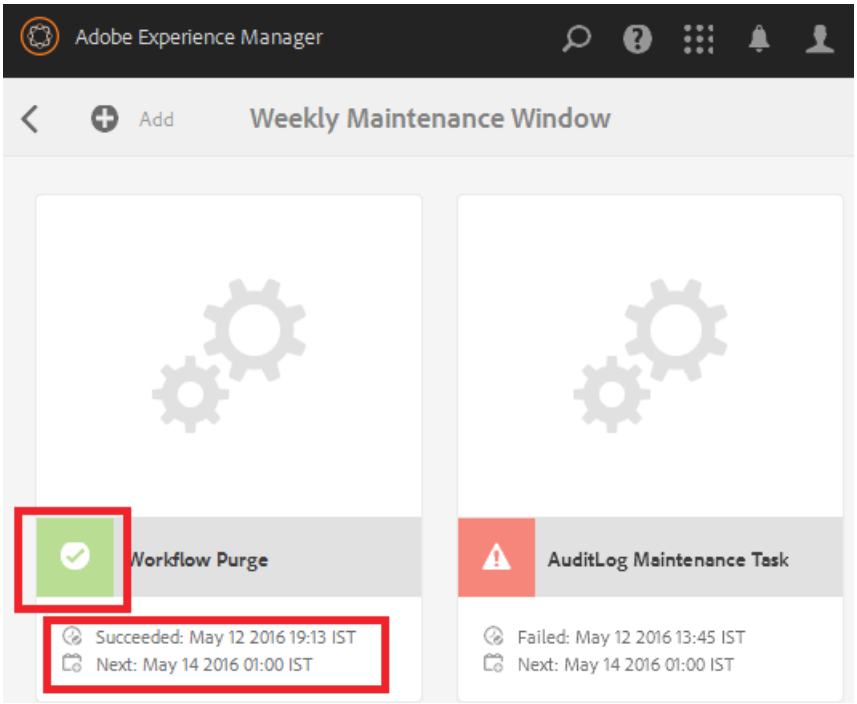


11. Navigate to **AEM Home > Tools > Web Console > Configuration Manager** http://localhost:4502/system/console/ configMgr/com.adobe.granite.workflow.purge.Scheduler again.



Notice that all values, except the workflow models are filled in. The explanation next to the ModelIds to Purge notes that leaving the ModelIds blank will purge all models.

12. Return to the Workflow Purge task and run the task. You will notice that the card has a green indicator, meaning Success.



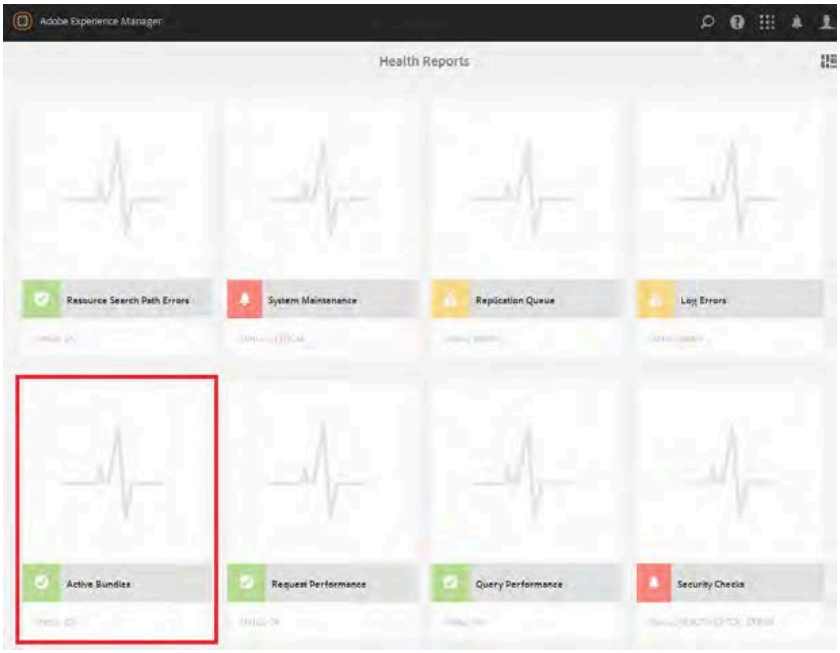
Congratulations, you now know how to configure a purge task.

Extra credit exercise: Successfully run the Audit log Maintenance Task.

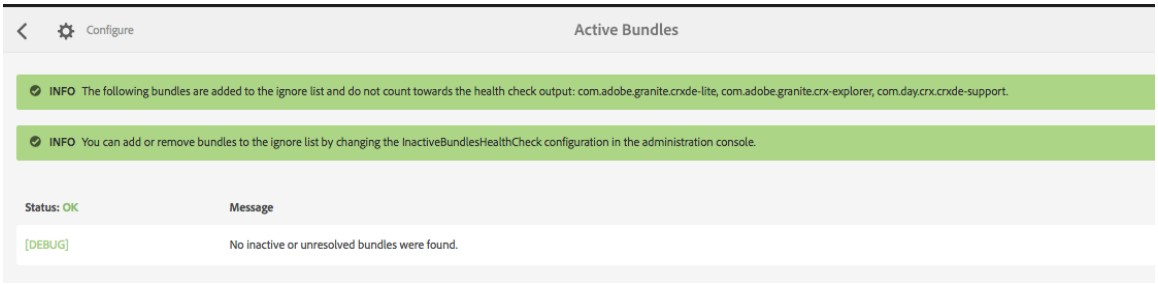
1. Using the same methodology as Exercise 8.4 and the com.adobe.cq.audit.purge.Scheduler. Configure the Audit Log Maintenance task to run successfully.

Exercise Extra Credit: Configure Health Reports to check Bundle status.

1. Navigate to AEM Home > Tools > Operations > Health Reports.
2. Check the **Active Bundles** card in the **Health Reports.**

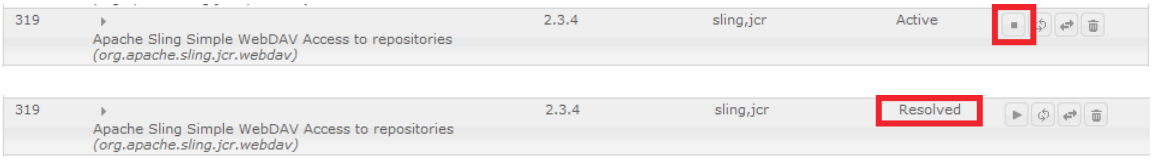


3. After the vanilla installation of AEM with the out-of-the-box settings, the status of this card should be **OK** (Green card). Click the card to ensure that there are no inactive or unresolved bundles.

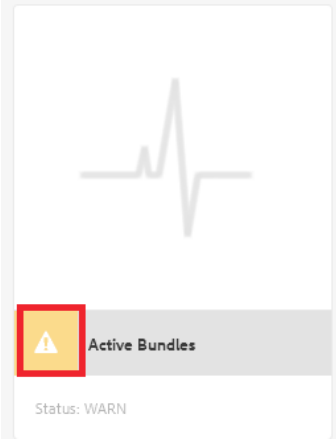


To **test the Health Check**, you will stop a bundle.

4. Navigate to **AEM Home > Tools > Web Console**.
5. Click the **Stop** button for the **Apache Sling Simple WebDAV Access to Repositories** bundle.

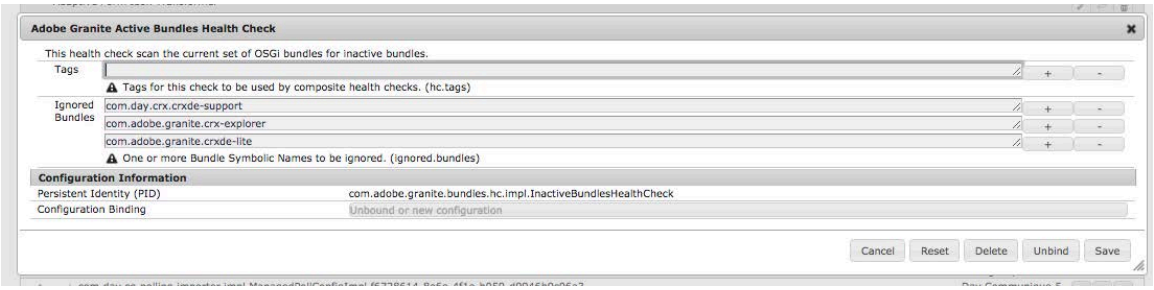


6. Return to the Health Reports and refresh the page. You will notice that the Active Bundles card has changed to yellow (WARN).

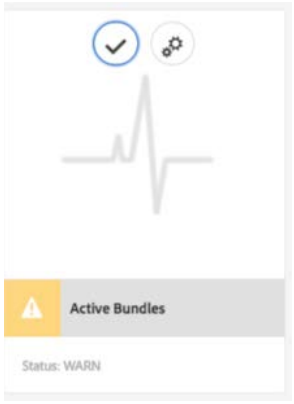


You can also configure the Active Bundles Health Check to monitor the status a specific set of bundles.

7. Navigate to **AEM Home > Tools > Web Console > Configuration Manager**. Find the **Adobe Granite Active Bundles Health Check**.



Pro Tip: There is a short cut to go directly to the **Adobe Granite Active Bundles Health Check** bundle from the Active Bundles Health Check. Click the Configuration button on the Active Bundles card.



8. For testing purposes, you can quickly fill in the form in the Web Console configuration tab, but it is preferable to use a node of type **sling:OsgiConfig** with the appropriate name and properties as described below.

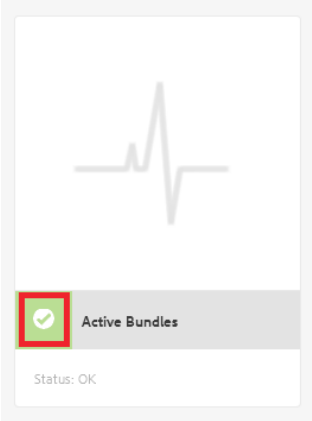
Node:

Node Type	sling:OsgiConfig
Name	com.adobe.granite.bundles.hc.impl.InactiveBundlesHealthCheck

Properties:

hc.tags	String[]= "" (leave this blank)
ignored.bundles	String[]= org.apache.sling.jcr.webdav

9. Return to the **Health Reports** and you will notice that the **Active Bundles** card has returned to the green **OK** status:



The OK status is reached because you specified that the org.apache.sling.jcr.webdav bundle be ignored.

Congratulations! You successfully used the Operations dashboard to check the status of your AEM instance concerning the 'Active Bundles', and you have learned how to configure the health check bundle associated with this card.

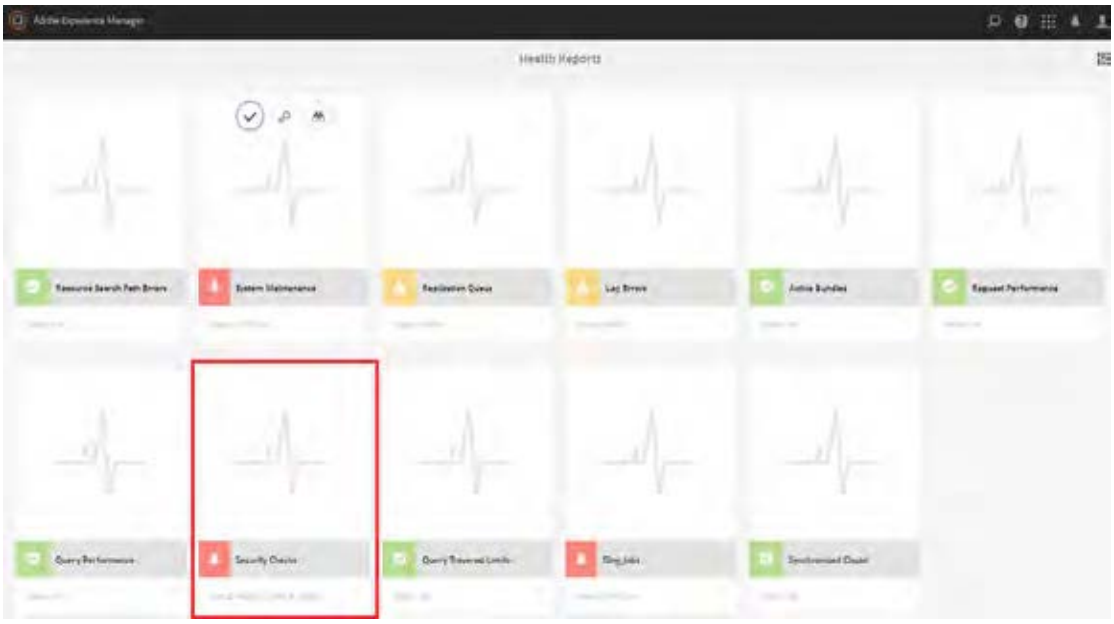
Composite Health Checks

A Composite Health Check's role is to aggregate a number of individual Health Checks sharing a set of common features. For instance, the Security Composite Health Check groups together all the individual health checks performing security checks.

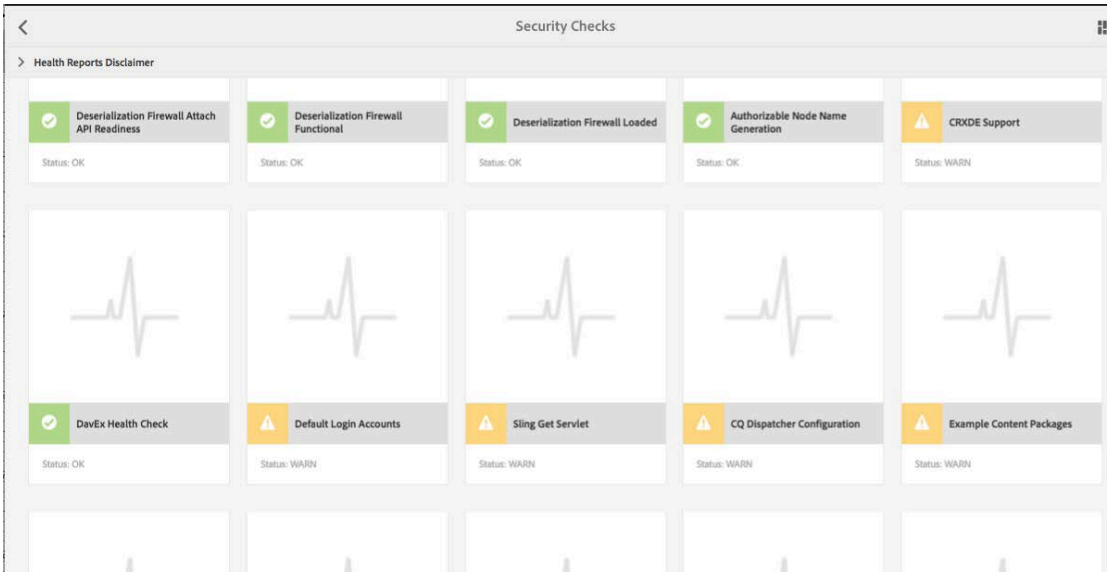
You can create custom composite Health Checks by configuring the Composite Health Check OSGi component.

Exercise 6 Working with Security Checks

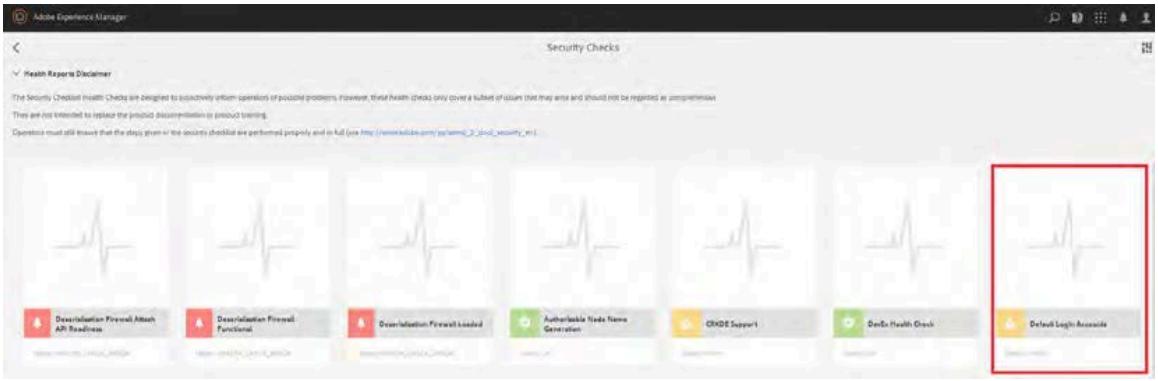
1. Navigate to **Tools > Operations > Health Reports > Security Checks**. You will see some cards for a number of key security points. Note the disclaimer as it is important to realize that there is more to security than what you find in the dashboard. For more help on this point, you should consult the online documentation as concerns the Security checklist: *https://docs.adobe.com/docs/en/aem/6-2/administer/security/security-check- list.html*



Most of the cards in **Security Checklist** dashboard are in yellow WARN status.



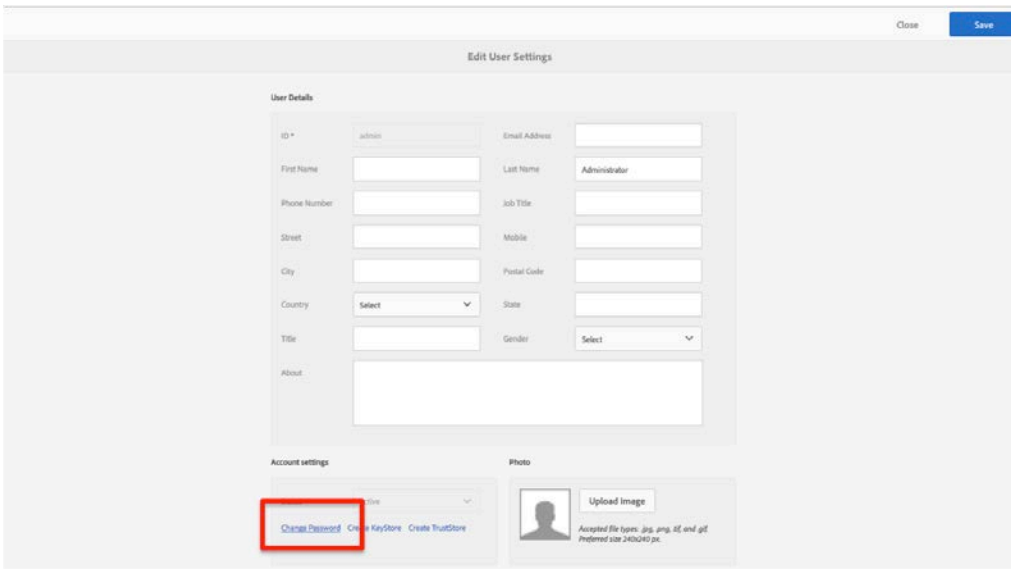
2. Notice that the Default Logins card is one of the Security Checks with WARN status.



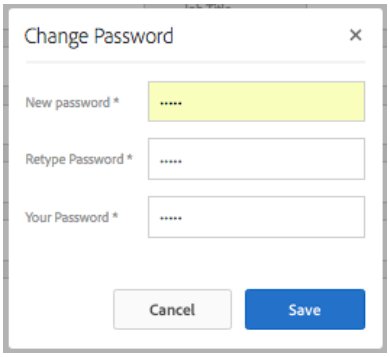
3. Click on the Default Logins card to see a diagnosis and Hint for solutions.

Follow the hints to find a solution for this Security Check.

4. Change the CRX Password.
    a. Navigate to **AEM Home** > **Tools** > **Security**.
    b. Select the **Administrator** user.

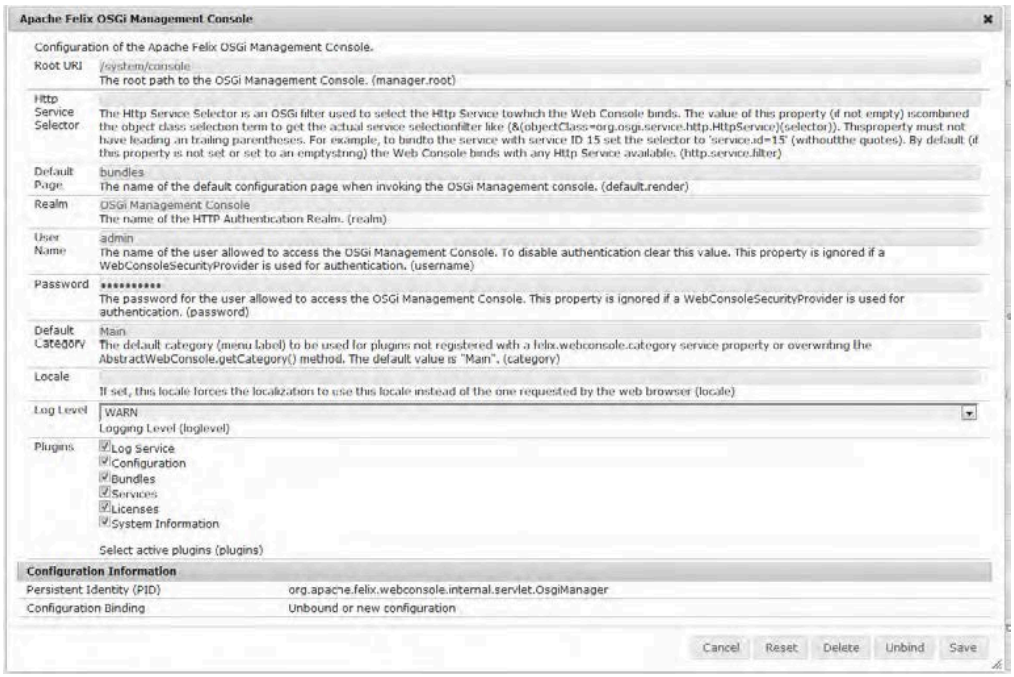


c. Click the **Change Password** link.
d. Modify the password.



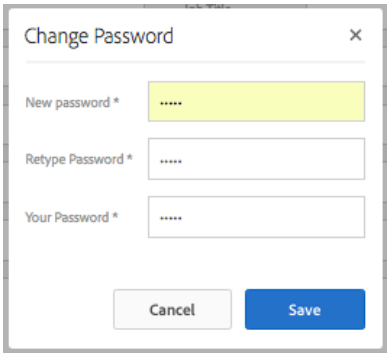
NOTE: **do not forget the password!! This is the new administrator password!!**

5. Change the Web Console password
    a. Navigate to **AEM Home > Tools > Web Console > Configuration Manager**.
    b. Find the **Apache Felix OSGi Management Console** component and modify the password directly in the console.

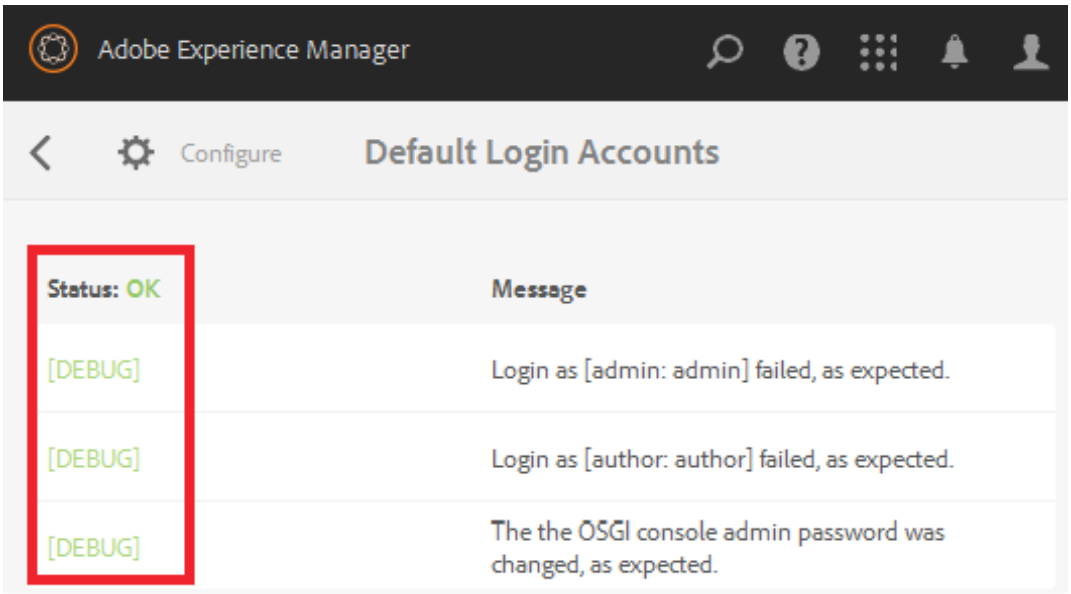


Note: this is only item that should be configured directly in the Web Console.

6. Change the password for the default user Author.
    a. Navigate to **AEM Home** > **Tools** > **Security**.
    b. Select the **Author** user.
    c. Click the **Change Password** link.
    d. Modify the password.



7. Return to the Security Checks page in the Dashboard. The Default Logins card is now green (OK status).



Congratulations! You consulted the Security Checks dashboard, addressed some of the points in the Security checklist to change the status of the Default Login's security card, and learned about the Security checklists.