

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

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

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

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

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

All other trademarks are the property of their respective owners.

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

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 Unpublished-rights reserved under the copyright laws of the United States.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

Contents

Using Application APIs	3
Using AEM APIs	4
Exercise 1: Create an AEM page programmatically	5
Task 1: Create a page creator servlet	5
Task 2: Install via command line	9
Task 3: Test the service	10
Exercise 2: Import multiple pages (Optional)	12
Workflow Process Steps	14
Exercise 3: Use a Workflow for Asset Review	15
Task 1: Create a Workflow Process	15
Task 2: Install via command line	18
Task 3: Install and update a workflow	19
Task 4: Test the workflow process	24
References	27

Using Application APIs

Introduction

Application Programming Interfaces (APIs) are at the highest layer of the Adobe Experience Manager (AEM) architecture stack, and are most closely related to the different business tasks a typical content producer would create in AEM. You can use APIs in AEM to automate nearly any business task. Some examples include:

- CRUD page actions
- CRUD content on pages
- CRUD assets actions
- Trigger publishing, workflows, and other actions
- Custom workflows

Objectives

After completing this course, you will be able to:

- Create an AEM page programmatically
- Import multiple pages
- Explain the steps in a workflow
- Customize a workflow process step

Using AEM APIs

Create Pages Dynamically

AEM has 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.

Create Assets Dynamically

Assets consist of nodes and properties along with a **sling:resourceType**. You can use the **com.day.cq.dam.api.AssetManager** API to create the asset, and the **com.day.cq.tagging.TagManager** API 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 (DAM), with all the asset information that is required.

More AEM APIs

Content

Every object within AEM is a Java object and therefore can be manipulated programmatically. Below is a list of some common AEM packages for many AEM related objects and services.

- `com.day.cq.wcm.api`
- `com.day.cq.wcm.api.components`
- `com.day.cq.wcm.msm`
- `com.adobe.granite.workflow`
- `com.adobe.granite.taskmanagement`
- `com.adobe.granite.timeline`
- `com.adobe.granite.translation.api`

You can look these packages up in the online java docs:

- [Cloud Service](#)
- [AEM 6.5](#)

Exercise 1: Create an AEM page programmatically

When migrating a website into AEM, it might make sense to create a process to create AEM pages to quickly create a site structure for content authors. This becomes particularly powerful when initial site structure is beyond manual page creation. In this exercise you will create a Java class that takes in a comma-delimited string that is parsed and a page is created based on the input.

This exercise includes the following tasks:

1. Create a page creator servlet
2. Install via command line
3. Test the service

Task 1: Create a page creator servlet

In this task, you will create a servlet named **PageCreator.java**. This Servlet will take an input parameter in the form of a string or CSV (New Page Path, Page Title, Template, Page Tag) and outputs AEM page(s).

1. In your IDE, navigate to **core > src > main/java/com/adobe/training/core/servlets**.
2. Right-click the folder **servlets** and select **New File**.
3. Name the file as **PageCreator.java**.
4. Copy the contents from **Exercise_Files-EC** under **/core/src/main/java/com/adobe/training/core/servlets/PageCreator.java** to **PageCreator.java** in your IDE. The **PageCreator.java** class is created.
5. Observe the code, as shown:

```
package com.adobe.training.core.servlets;
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.security.AccessControlException;
import java.util.HashMap;
import javax.servlet.Servlet;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.SlingHttpServletResponse;
import org.apache.sling.api.servlets.HttpConstants;
```

```

import org.apache.sling.api.servlets.SlingAllMethodsServlet;
import org.apache.sling.servlets.annotations.SlingServletResourceTypes;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Reference;
import com.day.cq.tagging.InvalidTagFormatException;
import com.day.cq.tagging.Tag;
import com.day.cq.tagging.TagManager;
import com.day.cq.wcm.api.Page;
import com.day.cq.wcm.api.PageManager;
import com.day.cq.wcm.api.PageManagerFactory;
import com.day.cq.wcm.api.WCMException;
import com.fasterxml.jackson.databind.ObjectMapper;
/**
 * This Servlet take an input parameter in the form of a string or CSV:
 *
 * New Page Path, Page Title, Template, Page Tag
 *
 * And outputs AEM page(s) created.
 *
 * To test, you must create a content node with a resourceType=training/tools/pagecreator
 *
 * /content/pagecreator {sling:resourceType=training/tools/pagecreator}
 *
 * Example cURL Command:
 * String Example:
 *
 * $ curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.json -F importer=»/content/training/us/en/
community,Our Community,/conf/training/settings/wcm/templates/page-content,/content/cq:tags/training/community»
 *
 * CSV example:
 *
 * $ curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.json -F importer=@PageCreator.csv
 */

@Component(service = { Servlet.class })
@SlingServletResourceTypes(
    resourceTypes="training/tools/pagecreator",
    methods=HttpConstants.METHOD_POST)
public class PageCreator extends SlingAllMethodsServlet {
    private static final long serialVersionUID = 1L;

    public static final String DEFAULT_TEMPLATE_PATH = "/conf/training/settings/wcm/templates/page-content";

    //Get a PageManager instance from the factory Service
    @Reference private PageManagerFactory pageManagerFactory;

    @Override
    public void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response) throws IOException {
        response.setContentType("application/json");
        HashMap<String, HashMap<String, String>> resultObject = new HashMap<>();

        String param = request.getParameter("importer");
        byte[] input = param.getBytes(StandardCharsets.UTF_8);
        InputStream stream = new ByteArrayInputStream(input);
        try {
            resultObject = readInput(request, stream);
        } catch (IOException e) {
            resultObject = null;
        }

        ObjectMapper objMapper = new ObjectMapper();
        if (resultObject != null) {
            //Write the result to the page
            response.getWriter().print(objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(resultObject));
        }
    }
}

```

```

    } else {
        HashMap<String, String> errorObject = new HashMap<String, String>() {
            private static final long serialVersionUID = 1L;
            {
                put("Error","Could not read csv input");
            }
        };
        response.getWriter().print(objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(errorObject));
    }
    response.getWriter().close();
}

/**
 * Reads the input. The input MUST be in the form of:
 * JCR path, Page Title, Page Template, AEM Tag
 *
 * @param request resourceResolver will be derived from the request
 * @param stream Stream from the input
 * @return JSON object that contains the results of the page creation process
 */
private HashMap<String, HashMap<String, String>> readInput(SlingHttpServletRequest request, InputStream stream) throws IOException {
    HashMap<String, HashMap<String, String>> out = new HashMap<>();

    String line;
    String[] newPage;
    HashMap<String, String> createdPageObject = null;

    try (BufferedReader br = new BufferedReader(new InputStreamReader(stream))) {
        //Read each line of the CSV and if the input is a string, this will loop once
        while ((line = br.readLine()) != null) {
            newPage = line.split(",");
            String inputPath="", inputTitle="", inputTemplatePath="", inputTag="";

            //set values from input
            if (newPage.length > 0)
                inputPath = newPage[0]; //Add the creation path
            if (newPage.length > 1)
                inputTitle = newPage[1]; //Add the Title of the new page
            if (newPage.length > 2)
                inputTemplatePath = newPage[2]; //Add the desired template
            if (newPage.length > 3)
                inputTag = newPage[3]; //Add requested tags

            //Create a new page based on a line of input
            createdPageObject = createTrainingPage(request, inputPath, inputTitle, inputTemplatePath, inputTag);

            //add the status of the row into the json array
            if(createdPageObject.get("Status").equals("Successful")) {
                out.put(inputPath, createdPageObject);
            } else {
                out.put(line, createdPageObject);
            }
            createdPageObject = null;
        }
    }
    return out;
}

/** Helper method to create the page based on available input
 *
 * @param request resourceResolver will be derived from the request
 * @param path JCR location of the page to be created

```

```

* @param title Page Title
* @param template AEM Template this page should be created from. The template must exist in the JCR already.
* @param tag Tag must already be created in AEM. The tag will be in the form of a path. Ex /content/cq:tags/marketing/
* @return HashMap
*/

private HashMap<String, String> createTrainingPage(SlingHttpServletRequest request, String path, String title, String template, String tagPath) {

    HashMap<String, String> pageInfo = new HashMap<>();
    pageInfo.put("Status", "Error");

    if (path != null && !path.isEmpty()) {
        //Parse the path to get the pageNodeName and parentPath
        int lastSlash = path.lastIndexOf("/");
        String pageNodeName = path.substring(lastSlash + 1);
        String parentPath = path.substring(0, lastSlash);

        //Set a default template if none is given
        boolean invalidTemplate = false;
        if (template == null || template.isEmpty()) { //if no template has been given, assign the default
            template = DEFAULT_TEMPLATE_PATH;
        }
        else if (request.getResourceResolver().getResource(template) == null) { //check to see if the template exists
            invalidTemplate = true;
            pageInfo.put("Template", "The template " + template + " doesn not exist.");
        }

        //Create page
        PageManager pageManager = pageManagerFactory.getPageManager(request.getResourceResolver());

        Page p = null;
        //Verify parentPath exists, a node for this page exists, and the template is valid
        if (pageManager != null && !parentPath.isEmpty() && !pageNodeName.isEmpty() && !invalidTemplate) {
            if (title == null || title.isEmpty()) {
                pageInfo.put("Warning", "No Page title given, using path name: " + pageNodeName);
                title = pageNodeName;
            }
            try {
                p = pageManager.create(parentPath,
                    pageNodeName,
                    template,
                    title);
            } catch (WCMException e) {
                pageInfo.put("Error", "Page couldn't be created. Parent path probably doesn't exist.");
            }

            //Check to see if the page was successfully created
            if (p != null) {
                //Add a tag to the page
                if (tagPath != null && !tagPath.isEmpty()) {
                    //Make sure tag namespaces are properly formed
                    if (tagPath.contains("/content/cq:tags") || tagPath.contains(".") || tagPath.contains("/etc/tags")) {
                        //TagManager can be retrieved via adaptTo
                        TagManager tm = request.getResourceResolver().adaptTo(TagManager.class);
                        Tag tag;
                        try {

```



```

        tag = tm.resolve(tagPath); //check if tag already exists
        if(tag == null) {
            pageInfo.put("Warning","Tag doesn't exist, creating new tag: " + tagPath);
            tag = tm.createTag(tagPath, null, null);
        }
        tm.setTags(p.getContentResource(), new Tag[] {tag}, true);
    } catch (AccessControlException e) {
        pageInfo.put("Warning","Could not access the tags.");
    } catch (InvalidTagFormatException e) {
        pageInfo.put("Warning","Invalid Tag.");
    }
} else {
    pageInfo.put("Warning", "Tag path malformed and not added: " + tagPath);
}
}

pageInfo.put("Status", "Successful");
pageInfo.put("Location", p.getPath());
pageInfo.put("Title", p.getTitle());
pageInfo.put("Template Used", p.getTemplate().getPath());
String tags = "";
for(Tag t : p.getTags() ) { tags = tags + t.getTitle() + " "; }
pageInfo.put("Tagged with", tags);
}
} else {
    pageInfo.put("Error", "Page path not provided");
}
}
return pageInfo;
}
}

```

6. Save the changes.

Task 2: Install via command line

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



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

2. In the command prompt run the command:

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

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

Task 3: Test the service

The servlet created requires a resource to contain a resourceType=**training/tools/pagecreator**. To test this servlet, you need to create a resource with this resourceType to call your servlet.

1. Open CRXDE Lite <http://localhost:4502/crx/de/index.jsp> and navigate to **/content**.
2. Right-click **content**, and select **Create > Create node**. Provide the following values:
 - a. Enter **pagecreator** for the **Name** field.
 - b. Select **nt:unstructured** from the **Type** drop-down menu.
3. Click **OK**. The pagecreator node is created.
4. Click **Save All**.
5. On the **pagecreator** node, add the following property, as shown:
 - Name: **sling:resourceType**
 - Type: **String**
 - Value: **training/tools/pagecreator**

Properties			
Name		Type	Value
1	jcr:primaryType	Name	nt:unstructured
2	sling:resourceType	String	training/tools/pagecreator

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



Note: You have content that you can post and a servlet that will be triggered. You can now test with a cURL command.

7. Copy the first command from **Exercise_Files-EC** under **/training-files/Dive_into_AEM_APIs/pagecreator-curl-commands.txt**.
8. Open your command prompt window, and paste the command, as shown:

```

Microsoft Windows [Version 10.0.17134.1425]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\prpurush>curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.json -F importer="/content/training/us/en/community,Our Community,/conf/training/settings/wcm/templates/page-content,/content/cq:tags/training/community"
  
```



Note: If you are on a Mac, you can format the returned json by adding `| json_pp`.

9. Verify the newly created page in **Sites > TrainingProject > us > en**, as shown:



Note: If you want to use the browser rather than cURL, you can install the **http-pagecreator.zip** content package (from the exercise folder) and do a request for <http://localhost:4502/content/pagecreator.html>.

Remember! In this exercise, you created content within your local AEM environment. If this content should be part of your code base, it needs to be synchronized back to the Maven project.

- In **ui.content/src/main/content/META-INF/vault**, update **filter.xml** with:
 - › `<filter root="/content/pagecreator" mode="merge" />`
- In your IDE, right click **ui.content** > **Import from the AEM Server**

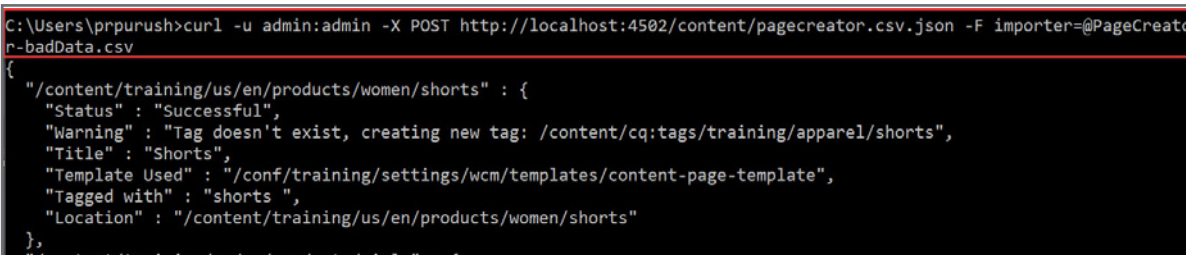
Exercise 2: Import multiple pages (Optional)

As you might have noticed, this java class also supports csv files. In this exercise, you will use a set of bad csv data for the importer and observe the output and then use good csv data to see it successfully import.

1. Try the servlet with bad data first. Execute the following statements, as shown:

```
cd ../Exercise_Files-EC/training-files/Dive-into-AEM-APIs
```

```
curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.csv.json -F importer=@PageCreator-badData.csv
```

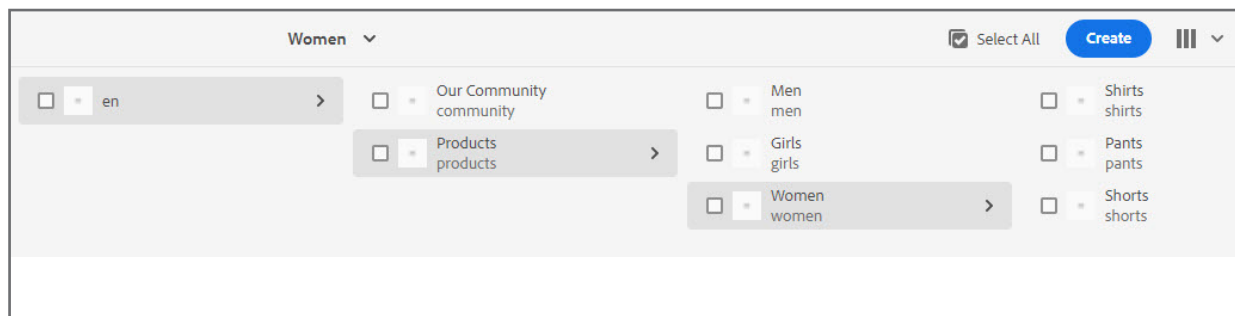


```
C:\Users\prpurush>curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.csv.json -F importer=@PageCreator-badData.csv
{
  "/content/training/us/en/products/women/shorts" : {
    "Status" : "Successful",
    "Warning" : "Tag doesn't exist, creating new tag: /content/cq:tags/training/apparel/shorts",
    "Title" : "Shorts",
    "Template Used" : "/conf/training/settings/wcm/templates/content-page-template",
    "Tagged with" : "shorts ",
    "Location" : "/content/training/us/en/products/women/shorts"
  },
  ...
}
```



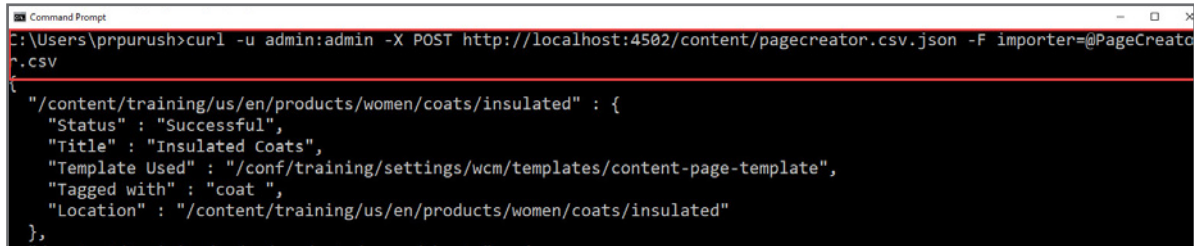
Note: You can also copy the command from **Exercise_Files-EC** under **/training-files/Dive_into_AEM_APIs/pagecreator-curl-commands.txt**.

2. In AEM, navigate to **Sites > TrainingProject > us > en**. Compare the set of child pages that were created to the pages defined in the csv file. Notice that not all the pages were created.



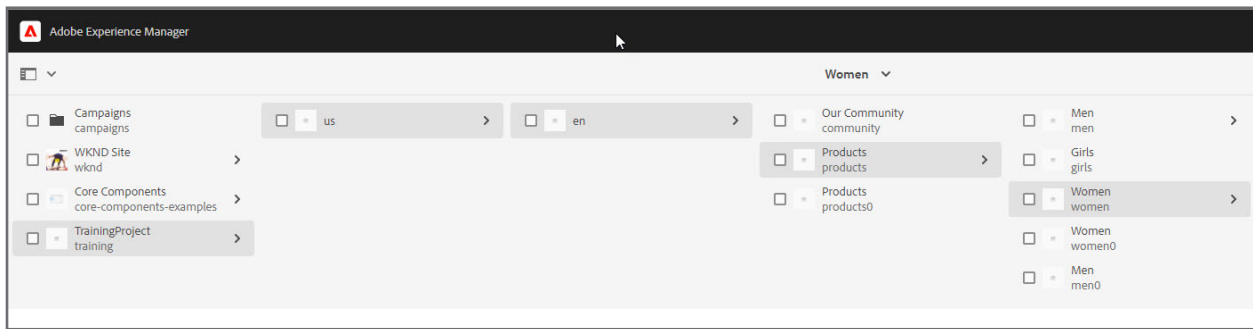
3. Try the servlet with well-formatted data by executing the following statement, as shown:

```
curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.csv.json -F importer=@PageCreator.csv
```



```
Command Prompt
C:\Users\prpurush>curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.csv.json -F importer=@PageCreator.csv
{"Status": "Successful", "Title": "Insulated Coats", "Template Used": "/conf/training/settings/wcm/templates/content-page-template", "Tagged with": "coat ", "Location": "/content/training/us/en/products/women/coats/insulated"}
}
```

4. Verify the newly created pages in AEM: **Sites > TrainingProject > us > en > Products**, as shown:



Note: Notice duplicate pages are created. For each page that was created in Step 1, a duplicate page is created in Step 3.

Note: If you would like to use an AEM component to support your page automation process, you can install training-files/Dive-into-AEM-APIs/http-pagecreator.zip in **Package Manager**. You then can access a single page input using <http://localhost:4502/content/pagecreator.html>. If you would like to upload a csv file, you can simply change the request to <http://localhost:4502/content/pagecreator.csv.html>.

Workflow Process Steps

In AEM, there are a number of steps available for workflows, such as Participant, Process, Create Task, Delete Node, Dialog Participant, Dynamic Participant, and Form Participant. Each step can contain any number of actions and associated conditions. For example, a step in a publish workflow may involve the step approval from an editor. Two of the most commonly used workflow steps are Participant and Process.

Process Step

The Process step involves automatic actions that are executed by the system if specific conditions are met. This step:

- Executes an ECMA script or calls an OSGi service to automate the process.
- Offers the following built-in processes:
 - › Workflow control processes: Control the behavior of the workflow and do not perform any action on content
 - › Basic processes: Delete the item at a given path or logs a debug message
 - › WCM processes: Perform WCM-related tasks, such as activating a page or confirming registration
 - › Versioning processes: Perform version-related tasks, such as creating versions of the payload
 - › DAM processes: Perform DAM-related tasks, such as creating thumbnails, creating sub-assets, and extracting metadata
 - › Collaboration processes: Are related to the collaboration features of AEM, such as the collaboration with social communities.

Developing Custom Steps

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

- Java class bundles: Create a bundle with the Java class, and then deploy the bundle into the OSGi container by using the Web console.
- ECMA scripts: Scripts are located in JCR 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.

Exercise 3: Use a Workflow for Asset Review

Scenario: AEM Assets has a review status field in the metadata. Typically this field is used in conjunction with the Asset Review Task or with document review for DITA content. In this scenario, ACME company wants to use this field to formalize the publishing process which will require a custom Workflow process. The desired process is:

- Designer uploads the asset
- Designer starts the **Asset Review Workflow**
- The workflow instructs the Designer to add descriptive metadata to the asset based on company standards
- The workflow then goes to the Reviewer who must review the asset and metadata and:
 - › Approve the asset which will trigger auto-publish
 - › Request changes and add comments to go back to the designer
 - › Reject the asset and inform the designer it should be removed

In this exercise, you will create a Workflow Process that will set the asset review status and the comment based on the process arguments. You will then install a training workflow that outlines the scenario above so you can add your new workflow process to it. Finally you will test the new workflow with your process to make sure the review status is set properly.

This exercise includes the following tasks:

1. Create a WorkflowProcess
2. Install via Maven command
3. Install and update a workflow
4. Test the workflow and process

Task 1: Create a Workflow Process

In this task, you will create a workflow process named **ReviewStatusWriter.java**. This workflow process writes the asset review status. If the user has added in a workflow comment, it is added to the review status comment.

1. In your IDE, navigate to **core > src > main/java/com/adobe/training/core**.

2. Right-click **core** and select **New File**.
3. Name the file as **ReviewStatusWriter.java**.
4. Copy the contents from **Exercise_Files-EC** under **/core/src/main/java/com/adobe/training/core/ReviewStatusWriter.java** to **ReviewStatusWriter.java** in your IDE. The **ReviewStatusWriter.java** class is created.
5. Observe the code, as shown:

```
package com.adobe.training.core;
import java.util.List;
import com.adobe.granite.asset.api.Asset;
import com.adobe.granite.workflow.PayloadMap;
import com.adobe.granite.workflow.WorkflowException;
import com.adobe.granite.workflow.WorkflowSession;
import com.adobe.granite.workflow.exec.HistoryItem;
import com.adobe.granite.workflow.exec.WorkItem;
import com.adobe.granite.workflow.exec.WorkflowData;
import com.adobe.granite.workflow.exec.WorkflowProcess;
import com.adobe.granite.workflow.metadata.MetaDataMap;
import com.day.cq.commons.jcr.JcrConstants;
import org.apache.sling.api.resource.ModifiableValueMap;
import org.apache.sling.api.resource.PersistenceException;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.api.resource.ResourceResolver;
import org.osgi.framework.Constants;
import org.osgi.service.component.annotations.Component;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * This workflow process writes the asset review status. If the user has added in
 * a workflow comment, it is added to the review status comment.
 *
 * /content/dam/<payload-node>/jcr:content/metadata
 * + dam:status = approved | rejected | changesRequested
 * + dam:statusComment = «string here»
 */
@Component(service = WorkflowProcess.class,
    property = {Constants.SERVICE_DESCRIPTION + "=Workflow process to set Asset Review Status",
        Constants.SERVICE_VENDOR + "=Adobe",
        "process.label=Set Asset Review Status"}
)
public class ReviewStatusWriter implements WorkflowProcess {

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

    private static final String REVIEW_APPROVED = "approved";
    private static final String REVIEW_REJECTED = "rejected";
    private static final String REVIEW_CHANGES_REQUESTED = "changesRequested";
```



```

@Override public void execute(
WorkflowItem item, WorkflowSession workflowSession, MetaDataMap args) throws WorkflowException {

    // Get the status review status to set in the process arguments.
    String assetReviewStatusArg = readArgument(args);
    if(assetReviewStatusArg != null && !assetReviewStatusArg.isEmpty()) {

        WorkflowData workflowData = item.getWorkflowData();
        if (workflowData.getPayloadType().equals(PayloadMap.TYPE_JCR_PATH)) {
            String contentPath = workflowData.getPayload().toString(); //content path attached to this workflow
            try (ResourceResolver rr = workflowSession.adaptTo(ResourceResolver.class)){

                Resource resource = rr.getResource(contentPath);

                //Verify that the content attached to this workflow is an asset
                Asset asset = resource.adaptTo(Asset.class);
                if (asset != null){
                    // Get a modifiable object for the metadata attached to this asset
                    String metadataPath = JcrConstants.JCR_CONTENT + "/metadata"; // jcr:content/metadata
                    ModifiableValueMap map = asset.getChild(metadataPath).adaptTo(ModifiableValueMap.class);

                    //Update the asset review status property based on the process arguments
                    logger.info(searchableLogStr + "Updating asset review status to: " + assetReviewStatusArg);
                    map.put("dam:status", assetReviewStatusArg);

                    // If there was a comment in the last completed workflow step, add it to the asset
                    List<HistoryItem> historyList = workflowSession.getHistory(item.getWorkflow());
                    String lastComment = getLastStepComment(historyList);
                    if(lastComment != null){
                        if(!lastComment.isEmpty()){
                            map.put("dam:statusComment", lastComment);
                            logger.info("Updating asset review status comment to: " + lastComment);
                        } else {
                            map.remove("dam:statusComment");
                            logger.info("Removing asset review status comment");
                        }
                    }
                    rr.commit(); // Save all changes
                }
            } else {
                // If the payload is not an AEM asset, throw a workflow error
                throw new WorkflowException(searchableLogStr + " The path: " + contentPath + " is not an asset.");
            }
        } catch (PersistenceException e) {
            throw new WorkflowException(e.getMessage(), e);
        }
    } else {
        // If the process arguments are not valid to set the review status, throw an error
        throw new WorkflowException(searchableLogStr + " Process argument is not a valid, check the process step in the workflow. Valid arguments are ["

```

```

        + REVIEW_APPROVED + " | " + REVIEW_REJECTED + " | " + REVIEW_CHANGES_REQUESTED + "]);
    }
}

/**
 * This method takes in the history of an active workflow and returns the comment of the last completed step.
 * @param historyList List of previous workitems in the workflow
 * @return The comment from the last workitem or an empty string if there no comment.
 */
private String getLastStepComment(List<HistoryItem> historyList) {
    int listSize = historyList.size();
    HistoryItem lastItem = historyList.get(listSize-1);
    String comment = lastItem.getComment();
    if(comment != null && comment.length() > 0) {
        return comment;
    }
    return "";
}

/**
 * This method reads the arguments that were added into the workflow step.
 * To set these argument go into the workflow and find the Process step > Process tab > Arguments
 * @param args These optional arguments added to the process step
 * @return returns the correct normalized value for the asset review status if it was set
 */
private static String readArgument(MetaDataMap args) {
    String argument = args.get("PROCESS_ARGS", "");
    if(argument.equalsIgnoreCase(REVIEW_APPROVED)){
        return REVIEW_APPROVED;
    } else if(argument.equalsIgnoreCase(REVIEW_REJECTED)){
        return REVIEW_REJECTED;
    } else if(argument.equalsIgnoreCase(REVIEW_CHANGES_REQUESTED)){
        return REVIEW_CHANGES_REQUESTED;
    }
    return "";
}
}

```

Task 2: Install via command line

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



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

2. In the command prompt run the command:

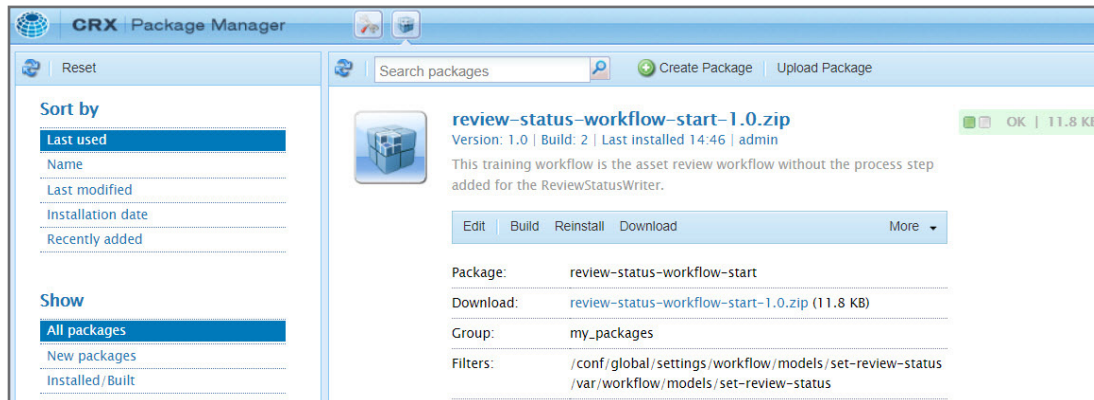
```
$ mvn clean install -Padobe-public -PautoInstallSinglePackage
```

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

Task 3: Install and update a workflow

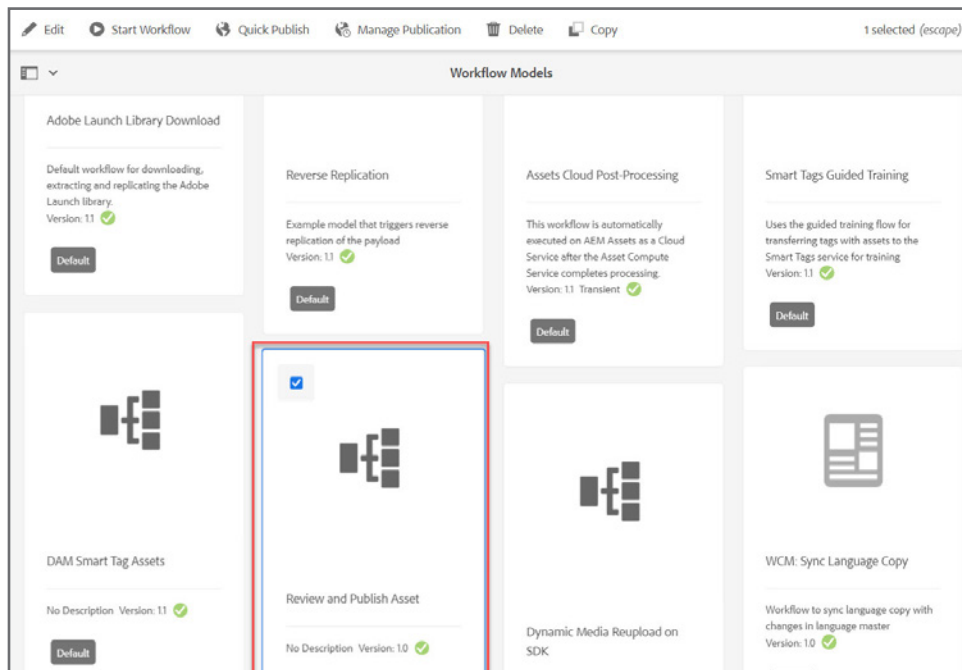
In this task you will install a workflow model that is already created based on the logic for this exercise scenario. You will then add the workflow process step based on the logic needed.

1. Log in to AEM and navigate to <http://localhost:4502/crx/packmgr/index.jsp>.
2. Install **Exercise Files-EC > training_files > Dive-into-AEM-APIs > review-status-workflow-start-1.0.zip**.
3. Verify the content packages are successfully installed:



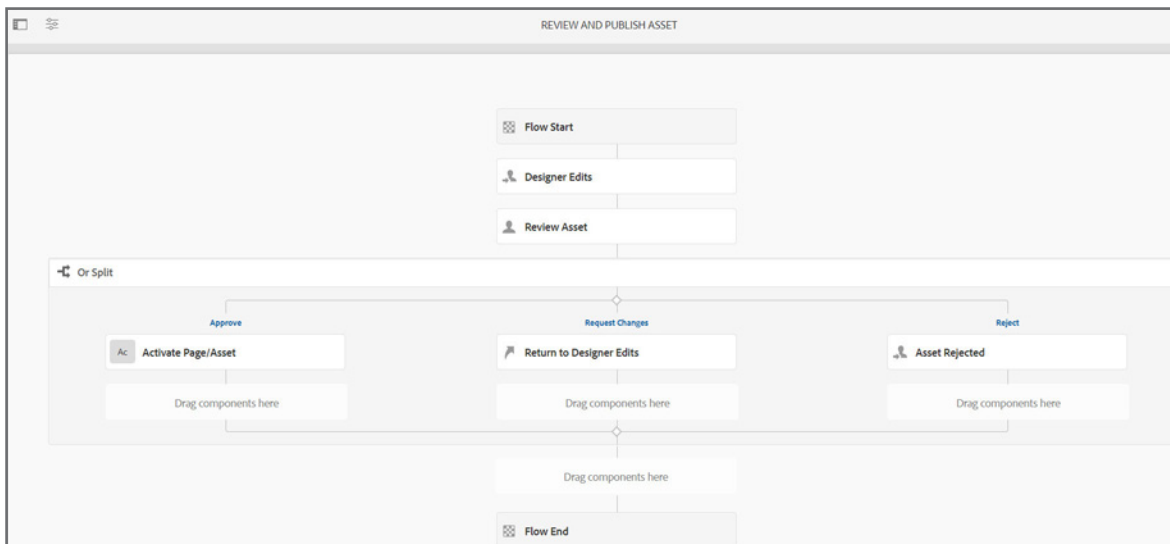
Now that a training workflow has been installed, the custom workflow process can be added.

4. Click the browser tab with the Adobe author service.
5. Click **Adobe Experience Manager** in the upper left.
6. Go to **Tools > Workflows > Models**. The **Workflow Models** page opens.
7. Select **Review and Publish Asset**, as shown:

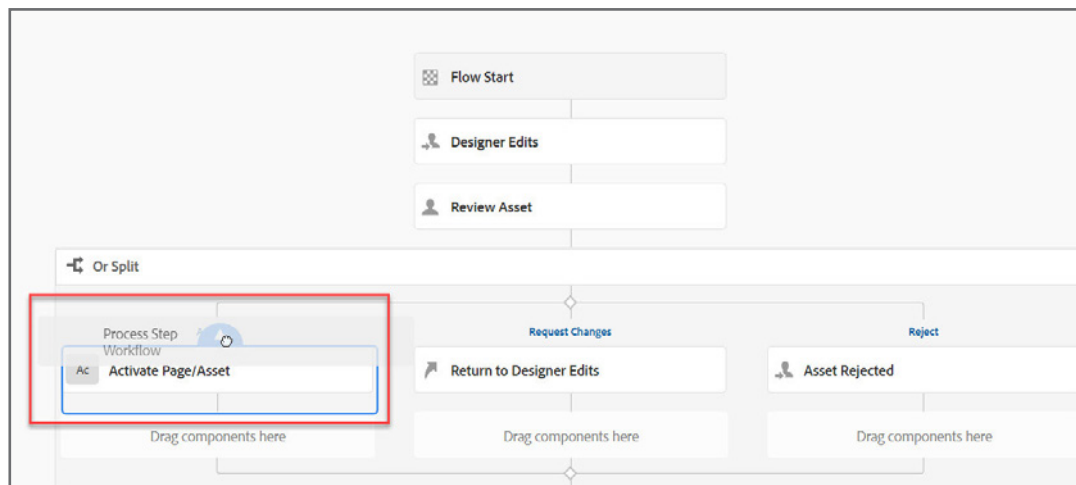


8. Click **Edit** from the action bar.

The **Review and Publish Asset** Workflow Model opens for editing in a new tab in your browser, as shown:



9. Drag the **Process step** component from the left-hand side onto the **Drag components here** box in the **Approve** branch, as shown:

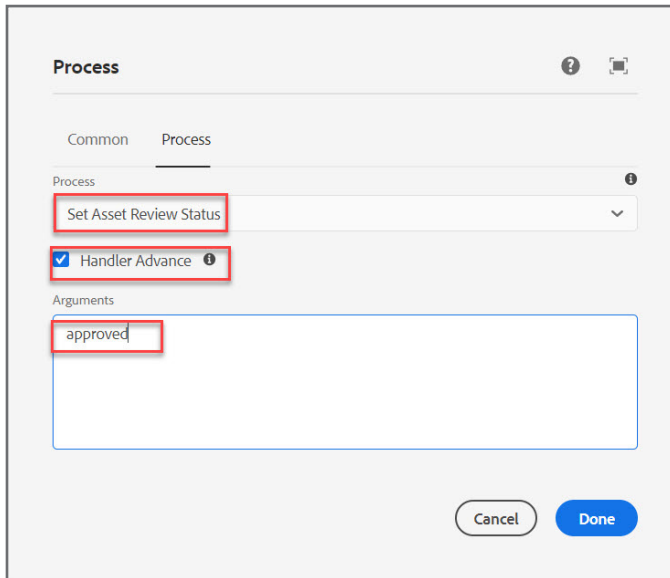



10. Click the **Process** step and click the **Configure** icon (wrench) to open the **Process** dialog box. By default, the dialog box opens on the **Common** tab.

11. In the **Title** field, type: **Approve Asset**

12. Click the **Process** tab and provide the following details, as shown:

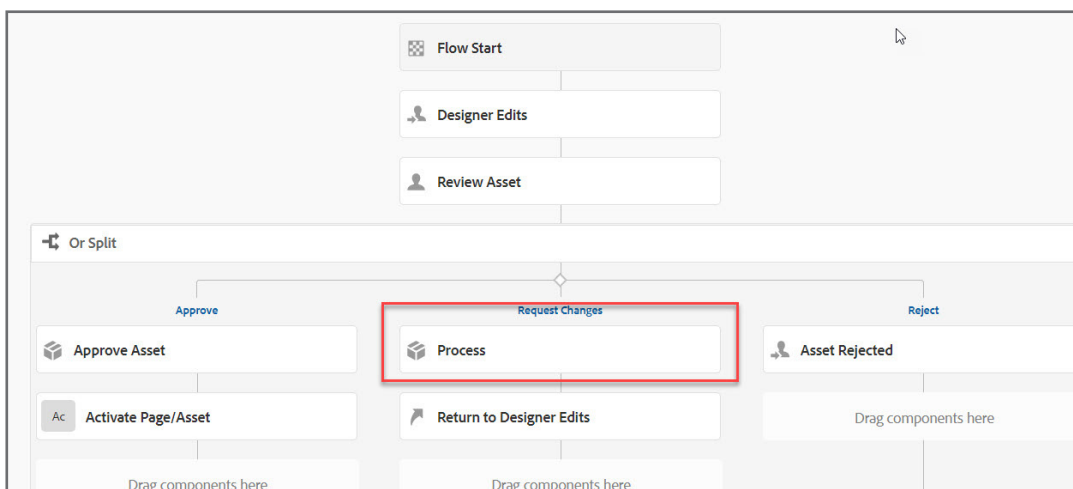
- › From the Process drop-down menu, select **Set Asset Review Status**.
- › Select the **Handler Advance** checkbox
- › Arguments: **approved**



 **Note:** Set Asset Review Status is the process.label in the ReviewStatusWriter.java component.

13. Click **Done**.

14. Drag another **Process step** component from the left-hand side onto the **Drag components here** box in the **Request Changes** branch, as shown:

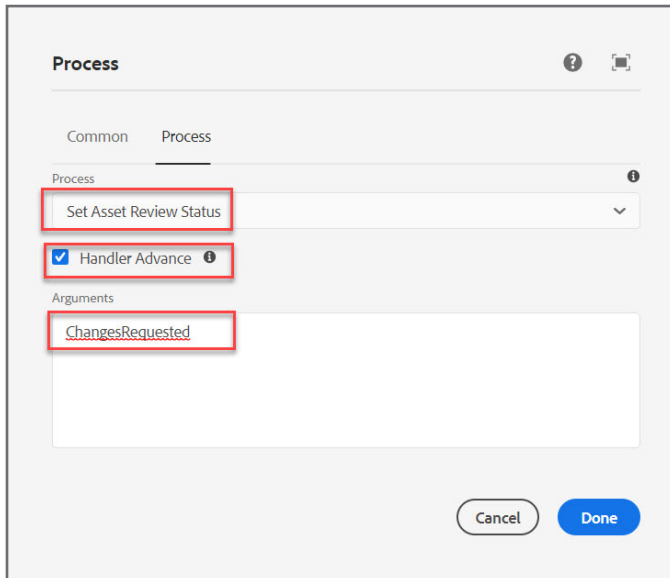


15. Click the **Process** step and click the **Configure** icon (wrench) to open the **Process** dialog box. By default, the dialog box opens on the **Common** tab.

16. In the **Title** field, type: **Request Changes**

17. Click the **Process** tab and provide the following details, as shown:

- › From the Process drop-down menu, select **Set Asset Review Status**.
- › Select the **Handler Advance** checkbox
- › Arguments: **ChangesRequested**



The screenshot shows a 'Process' dialog box with two tabs: 'Common' and 'Process'. The 'Process' tab is active. It contains a 'Process' dropdown menu with 'Set Asset Review Status' selected, a 'Handler Advance' checkbox which is checked, and an 'Arguments' text area containing 'ChangesRequested'. At the bottom are 'Cancel' and 'Done' buttons. Red boxes highlight the 'Set Asset Review Status' dropdown, the 'Handler Advance' checkbox, and the 'ChangesRequested' text.

18. Click **Done**.

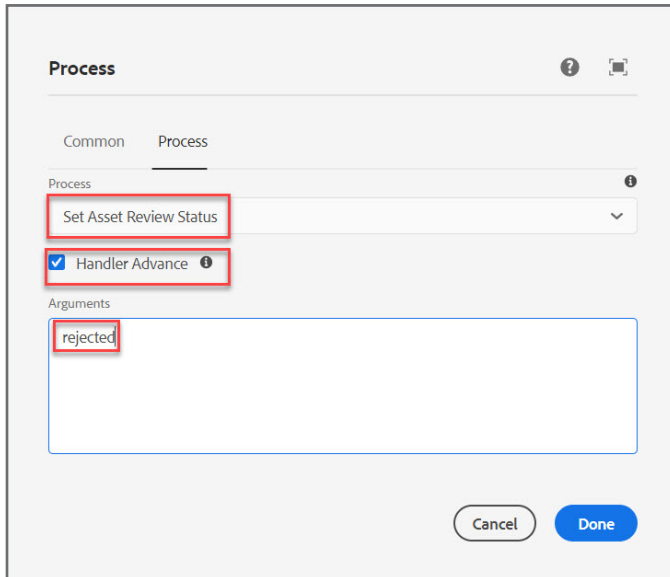
19. Drag another **Process step** component from the left-hand side onto the **Drag components here** box in the **Reject Asset** branch.

20. Click the **Process** step and click the **Configure** icon (wrench) to open the **Process** dialog box. By default, the dialog box opens on the **Common** tab.

21. In the **Title** field, type: **Reject Asset**

22. Click the **Process** tab and provide the following details, as shown:

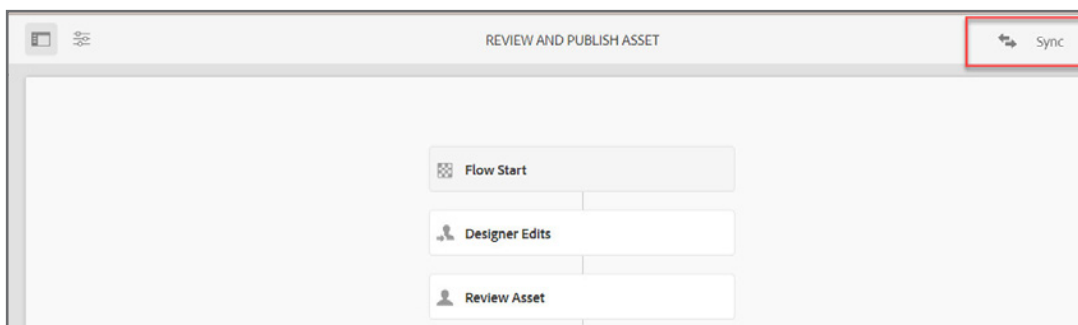
- › From the Process drop-down menu, select **Set Asset Review Status**.
- › Select the **Handler Advance** checkbox
- › Arguments: **rejected**



The screenshot shows a configuration window titled "Process". It has two tabs: "Common" and "Process". The "Process" tab is active. Under the "Process" section, there is a dropdown menu set to "Set Asset Review Status". Below this, the "Handler Advance" checkbox is checked. In the "Arguments" section, the text "rejected" is entered. At the bottom right, there are "Cancel" and "Done" buttons. The "Done" button is highlighted in blue.

23. Click **Done**.

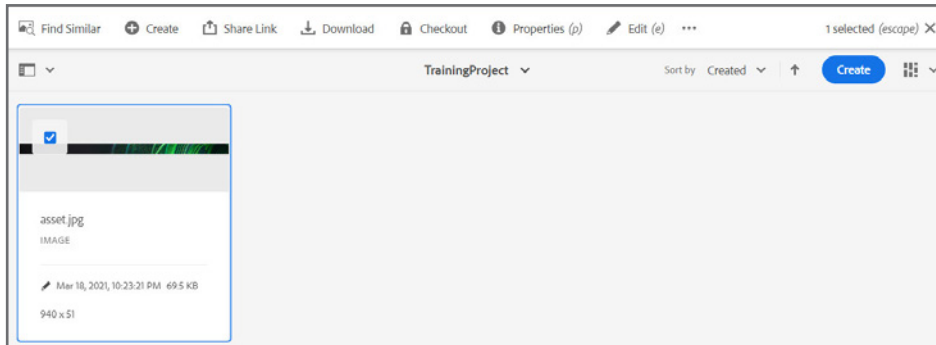
24. At this point the workflow is complete. In the upper right click **Sync**, as shown. This will write the workflow model.



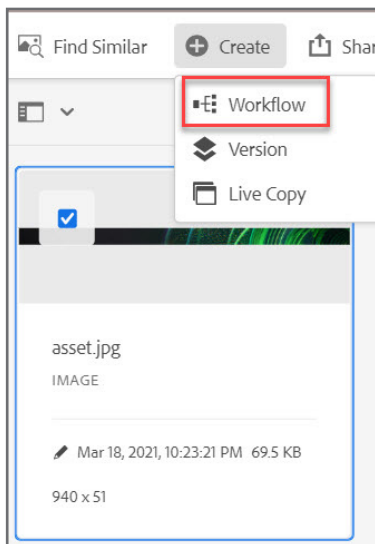
The screenshot shows a workflow editor titled "REVIEW AND PUBLISH ASSET". In the top right corner, there is a "Sync" button with a red box around it. The main workspace displays a workflow diagram with three steps: "Flow Start", "Designer Edits", and "Review Asset".

Task 4: Test the workflow process

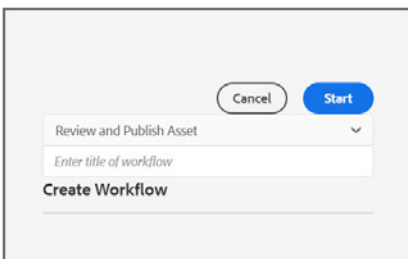
1. In AEM, go to the **Navigation** screen and click **Assets**.
2. Navigate to **Assets > Files > TrainingProject** and select **asset.jpg**, as shown:



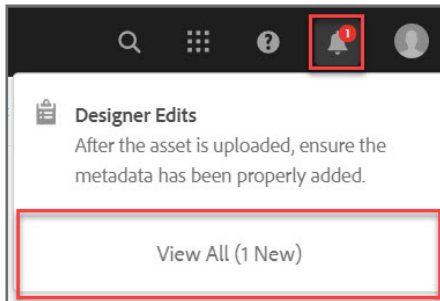
3. Select **Create > Workflow** in the action bar, as shown:



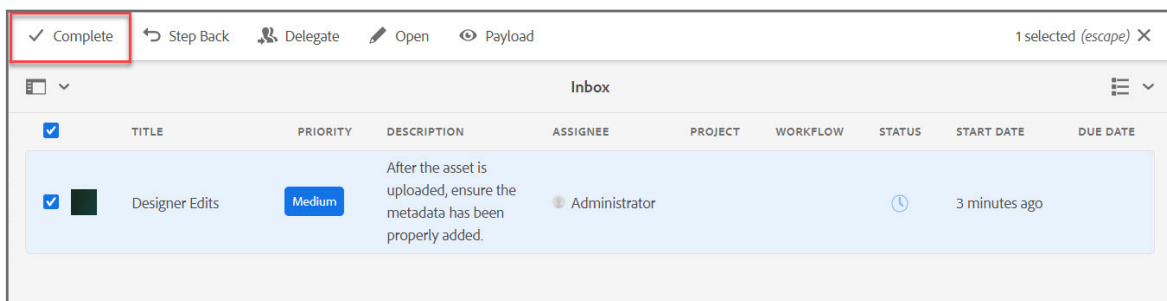
4. Select **Review and Publish Asset** and click **Start**, as shown:




5. Click **Proceed** in the **Warning** window.
6. Go up to the Notification bell in the top right and click to open the Inbox.



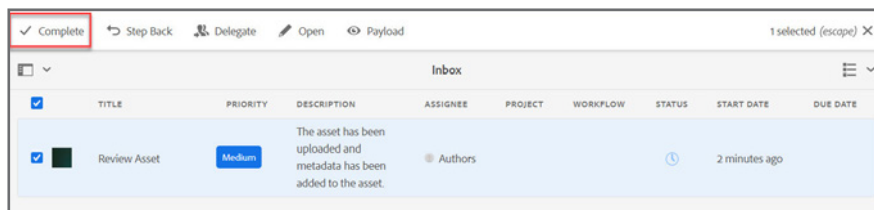
7. Select the **Designer Edits** and click **Complete** in the action bar, as shown:




8. In the **Complete Work Item** dialog, click **Ok**.

 **Note:** Realistically the designer would complete this step by going in and modifying the metadata of the asset. For training purposes, this is skipped.

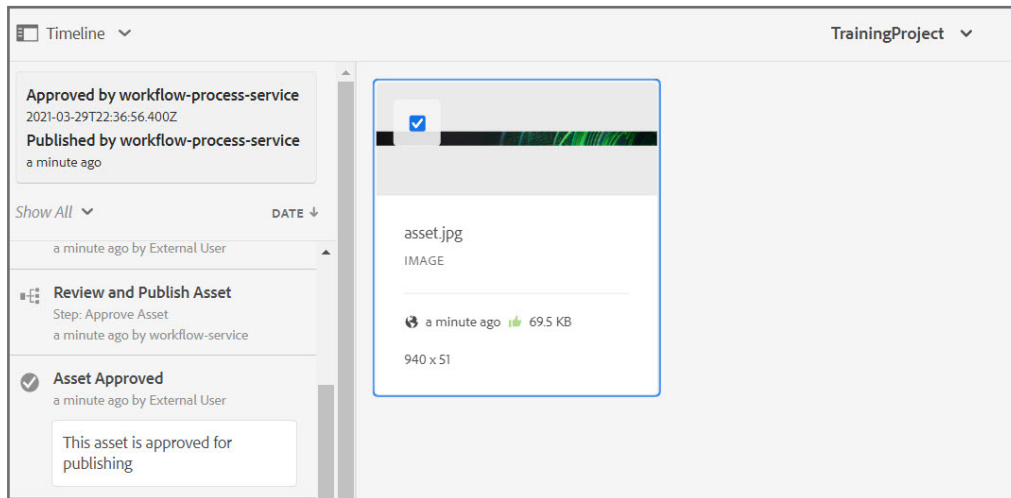
9. Refresh the browser.
10. Select the **Review Asset** and click **Complete** in the action bar, as shown:



 **Note:** Realistically the reviewer would go into the asset and verify the metadata was in compliance with the company standards. For training purposes, this is skipped.

11. In the **Complete Work Item** dialog, select **Approve Asset** and add a comment **This asset is approved for publishing** and click **Ok**.

12. The workflow is now complete. To verify the asset was approved and published, go back to the AEM start screen (<http://localhost:4502>).
13. Go to **Assets > Files > TrainingProject** and select **asset.jpg**.
14. On the left, select the rail icon > Timeline. Observe the asset has been approved with a comment and published, as shown:



Note: In this exercise, you only verified the Approve Asset path. In a real scenario, you would also want to verify the Request Changes and Reject Asset paths to make sure the workflow and process are working as expected.

Remember! In this exercise, you created content within your local AEM environment. If this content should be part of your code base, it needs to be synchronized back to the Maven project. The content to import from AEM: Review and Approve workflow.

- In **ui.content**, update the **filter.xml** with:

```
<filter root="/conf/global/settings/workflow/models/set-review-status" />  
<filter root="/var/workflow/models/set-review-status"/>
```
- In your IDE, right click **ui.content** > **Import from the AEM Server**.

References

For more information on Workflows:

<https://docs.adobe.com/help/en/experience-manager-cloud-service/sites/authoring/workflows/overview.html>