Adobe CQ Help / Developing CQ Custom Workflow Steps using Maven

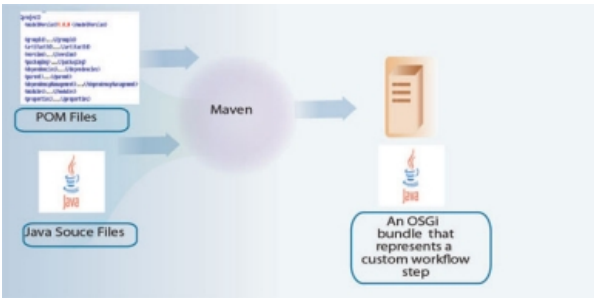
Article summary

Summary	Discusses how to create a custom Workflow step for Adobe CQ by using Maven.
Digital Marketing Solution(s)	Adobe Experience Manager (Adobe CQ)
Audience	Developer (intermediate)
Required Skills	Java, Maven, XML
Tested On	Adobe CQ 5.5

Introduction

You can create an Adobe CQ custom workflow step by using Maven. A custom workflow step lets you customize a CQ workflow to meet your business requirements. An Adobe CQ workflow consists of steps that are either participant steps or process steps.

Participant steps require manual intervention by a person to advance the workflow. Process steps are actions performed by Adobe CQ if certain configured conditions are satisfied. You can create either a participant or process custom workflow step using Maven. You use the Adobe CQ Java API to create a custom step.



Maven creating an OSGi bundle that represents a custom workflow step

Note: For a detailed reference guide on using Maven, see http://www.sonatype.com/books/mvnref-book/reference/public-book.html.

To create an Adobe CQ custom workflow step using Maven, perform the following tasks:

1. Setup Maven in your development environment.
2. Create a Maven Archetype for the Adobe CQ custom workflow step.
3. Develop custom workflow application logic by using the Java CQ API.
4. Build the OSGi bundle.
5. Deploy the custom workflow step to Adobe CQ.
6. Create a workflow with the new step.

Note: This development article uses Maven to build an OSGi bundle that represents a custom worflow step. You can build an OSGi bundle using Eclipse. For information, see http://scottsdigitalcommunity.blogspot.ca/2012/07/creating-custom-cq-email-services.html.

Setup Maven in your development environment

To the top

You can use Maven to build an Adobe CQ custom workflow step that consists of a Java server-side class. You can download Maven 3 from the following URL:

http://maven.apache.org/download.html

After you download and extract Maven, create an environment variable named `M3_HOME`. Assign the Maven install location to this environment variable. For example:

```
C:\Programs\Apache\apache-maven-3.0.4
```

You can test to determine if Maven is properly setup by entering the following command into a command prompt:

```
%M3_HOME%\bin\mvn -version
```

This command provides Maven and Java install details and resembles the following message:

```
Apache Maven 3.0.4 (r1232337; 2012-01-17 03:44:56-0500)

Maven home: C:\Programs\Apache\Maven\apache-maven-3.0.4

Java version: 1.6.0_31, vendor: Sun Microsystems Inc.

Java home: C:\Programs\Java64-6\jre

Default locale: en_US, platform encoding: Cp1252

OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

Next, copy the Maven configuration file named *settings.xml* from [install location]\apache-maven-3.0.4\conf\ to your user profile. For example, C:\Users\JAYAN.m2\.

Note: For more information about setting up Maven, see http://cq-ops.tumblr.com/post/31404223822/how-to-use-apache-maven-to-build-and-deploy-osgi.

You have to configure your *settings.xml* file to use Adobe's public repository. For information, see Adobe Public Maven Repository at http://repo.adobe.com/.

The following XML represents the settings.xml file used to build the Adobe CQ workflow step created in this development article.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org

    <modelVersion>4.0.0</modelVersion>

    <groupId>Jayan-Workflow</groupId>
    <artifactId>parent</artifactId>
    <version>1.0.0.0</version>
    <packaging>pom</packaging>

    <name>${project.groupId} - ${project.artifactId}</name>

    <description>
        Parent Maven POM for the 'Jayan-Workflow' project.
    </description>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncod
        <!--
            the following CQ connection properties are the default out-of-the-b
            you should override them in your local settings.xml file if your CQ
        -->
        <cq.host>jayan</cq.host>
        <cq.port>4502</cq.port>
        <cq.user>admin</cq.user>
        <cq.password>admin</cq.password>
    </properties>

    <scm>
        <connection><!-- TODO --></connection>
        <developerConnection><!-- TODO --></developerConnection>
        <url><!-- TODO --></url>
    </scm>

    <distributionManagement>
        <repository>
            <id></id>
            <name></name>
            <url></url>
        </repository>
        <snapshotRepository>
            <id></id>
            <name></name>
            <url></url>
        </snapshotRepository>
    </distributionManagement>

    <modules>
        <module>CQWorkflowProcessStep</module>
    <module>my-cq-project</module>
  </modules>

    <dependencyManagement>
```

```
        <dependencies>
            <dependency>
                <groupId>com.cqblueprints</groupId>
                <artifactId>cqdependencies</artifactId>
                <version>5.5.0</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>com.squeakysand.jcr</groupId>
                <artifactId>squeakysand-jcr-taglib</artifactId>
                <version>0.3.0</version>
                <scope>provided</scope>
            </dependency>
            <dependency>
                <groupId>com.squeakysand.jsp</groupId>
                <artifactId>squeakysand-jsp</artifactId>
                <version>0.4.0</version>
            </dependency>
            <dependency>
                <groupId>com.squeakysand.osgi</groupId>
                <artifactId>squeakysand-osgi</artifactId>
                <version>0.4.0</version>
            </dependency>
            <dependency>
                <groupId>com.squeakysand.sling</groupId>
                <artifactId>squeakysand-sling-taglib</artifactId>
                <version>0.3.0</version>
                <scope>provided</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>com.day.jcr.vault</groupId>
                    <artifactId>maven-vault-plugin</artifactId>
                    <version>0.0.10</version>
                    <configuration>
                        <verbose>true</verbose>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>com.squeakysand.jsp</groupId>
                    <artifactId>jsptld-maven-plugin</artifactId>
                    <version>0.4.0</version>
                </plugin>
                <plugin>
                    <groupId>net.sourceforge.maven-taglib</groupId>
                    <artifactId>maven-taglib-plugin</artifactId>
                    <version>2.4</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.felix</groupId>
                    <artifactId>maven-bundle-plugin</artifactId>
                    <version>2.3.7</version>
                    <configuration>
                        <instructions>
                            <Embed-Dependency>*;scope=compile|runtime</Embed-De
                            <Embed-Directory>OSGI-INF/lib</Embed-Directory>
                            <Embed-Transitive>true</Embed-Transitive>
                        </instructions>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.felix</groupId>
                    <artifactId>maven-scr-plugin</artifactId>
                    <version>1.7.4</version>
                    <executions>
                        <execution>
                            <id>generate-scr-descriptor</id>
                            <goals>
                                <goal>scr</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-clean-plugin</artifactId>
                    <version>2.4.1</version>
                </plugin>
```

```
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>2.3.2</version>
                    <configuration>
                        <showDeprecation>true</showDeprecation>
                        <showWarnings>true</showWarnings>
                        <source>1.6</source>
                        <target>1.6</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-install-plugin</artifactId>
                    <version>2.3.1</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-resources-plugin</artifactId>
                    <version>2.5</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-site-plugin</artifactId>
                    <version>3.0</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-surefire-plugin</artifactId>
                    <version>2.12</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.sling</groupId>
                    <artifactId>maven-sling-plugin</artifactId>
                    <version>2.1.0</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>

    <profiles>
        <profile>
            <id>eclipse</id>
            <build>
                <pluginManagement>
                    <plugins>
                        <!--
                            this plugin holds configuration information for Mav
                            it does not affect the actual Maven build process.
                            you should activate the "eclipse" profile for this
                            some of the error messages you see.
                         -->
                        <plugin>
                            <groupId>org.eclipse.m2e</groupId>
                            <artifactId>lifecycle-mapping</artifactId>
                            <version>1.0.0</version>
                            <configuration>
                                <lifecycleMappingMetadata>
                                    <pluginExecutions>
                                        <pluginExecution>
                                            <pluginExecutionFilter>
                                                <groupId>com.squeakysand.jsp</g
                                                <artifactId>jsptld-maven-plugin
                                                <versionRange>[0.3.0,)</version
                                                <goals>
                                                    <goal>generate</goal>
                                                </goals>
                                            </pluginExecutionFilter>
                                            <action>
                                                <execute>
                                                    <runOnIncremental>true</run
                                                </execute>
                                            </action>
                                        </pluginExecution>
                                        <pluginExecution>
                                            <pluginExecutionFilter>
                                                <groupId>org.apache.felix</grou
                                                <artifactId>maven-scr-plugin</a
                                                <versionRange>[1.7.4,)</version
                                                <goals>
                                                    <goal>scr</goal>
                                                </goals>
                                            </pluginExecutionFilter>
                                            <action>
```

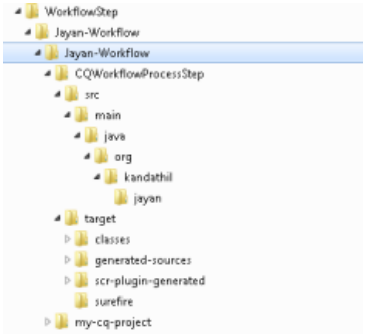
```
                                        <execute>
                                            <runOnIncremental>false</r
                                        </execute>
                                    </action>
                                </pluginExecution>
                            </pluginExecutions>
                        </lifecycleMappingMetadata>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</profile>
</profiles>

</project>
```

Note: For more information about setting up Maven, see http://cq-ops.tumblr.com/post/31404223822/how-to-use-apache-maven-to-build-and-deploy-osgi.

Create a Maven Archetype for the Adobe CQ custom workflow step

To the top

An archetype is a Maven template project that you can use to create a custom workflow step. This template project contains a location that allows you to place Java source files that contain application logic for the workflow step. Likewise, it contains a target directory in which Maven places the OSGI bundle. This OSGi bundle represents the custom workflow step.



The Maven template project that is used in this development article

The Java source files are placed in the `src/main/java` folder. Notice that the sub-folder structure reflects the Java package. In this example, the Java package is named `org.kandathil.jayan`. (The Java source file is created later in this development article.)

After you build the project using Maven, the JAR file (the OSGi bundle) is placed in the target directory.

To create the Maven Archetype template project, you have to perform a series of steps. For information, see *The CQ Project Maven Archetype* at http://www.cqblueprints.com/xwiki/bin/view/Blue+Prints/The+CQ+Project+Maven+Archetype.

Note: To make it easier to follow along with this development article and create a custom worflow step, the Maven template project shown in the previous illustration has been placed in a ZIP file named Jayan-Workflow.zip. You can download this file and follow along with the rest of this development article. Ensure that you add the settings.xml in this development article to your settings.xml file. To get this ZIP file, click this link.

Develop custom workflow application logic by using the Java CQ API

To the top

You can create application logic for the custom workflow step by using Adobe CQ Java API. To create a custom workflow step, your Java class must inherit from the `WorkflowProcess` class. The following Java code represents the custom workflow step. Notice that the `jjkCQProcessStep` class is located in a package named `org.kandathil.jayan`.

```
package org.kandathil.jayan;

// from http://www.jarvana.com/jarvana/archive-details/org/apache/felix/org.apa
import org.osgi.framework.Constants;

// following imports are from from /libs/cq/workflow/install/cq-workflow-api-5.
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;

// following imports are from /etc/crxde/profiles/default/libs/org.apache.felix.
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;

// following imports are from /etc/crxde/profiles/default/libs/jcr-2.0.jar
import javax.jcr.Node;
import javax.jcr.RepositoryException;

@Component
@Service
@Properties({
    @Property(name = Constants.SERVICE_DESCRIPTION, value = "Jayan Kandathil wor
    @Property(name = Constants.SERVICE_VENDOR, value = "Jayan Kandathil (Adobe (
    @Property(name = "process.label", value = "Jayan Test Workflow Step")
})
public class jjkCQProcessStep implements WorkflowProcess
{

private static final String TYPE_JCR_PATH = "JCR_PATH";

    @Override
    public void execute(WorkItem item, WorkflowSession session, MetaDataMap args
    {

        WorkflowData workflowData = item.getWorkflowData();

        // If the workflow payload is a path in the JCR
        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("JJKProperty", readArgument(args));
                    session.getSession().save();
                }
            }
            catch (RepositoryException e)
            {
                throw new WorkflowException(e.getMessage(), e);
            }
        }
    }

    private static boolean readArgument(MetaDataMap args)
    {
        String argument = args.get("PROCESS_ARGS", "false");
        return argument.equalsIgnoreCase("true");
    }
}
```

Build the OSGi bundle using Maven

To the top

Notice that the `jjkCQProcessStep` class is located in a package named `org.kandathil.jayan`. In the Maven template application, setup a folder under `CQWorkflowProcessStep\src\main\java` to reflect this package. That is, create the following folder structure:

```
CQWorkflowProcessStep\src\main\java\org\kandathil\jayan
```

Place the *jjkCQProcessStep.java* file in the jayan folder. You run Maven in the Command Prompt to build the package. Maven builds a package as a JAR file and places it in the template application's package/target folder. If the build is successful, you will see a message in the Command Prompt similar to the following message:

```
[INFO] ------------------------------------------------------------------------
------

[INFO] Reactor Summary:

[INFO]
```

```
[INFO] Jayan-Workflow - parent ........................... SUCCESS
[0.603s]

[INFO] Jayan-Workflow - CQWorkflowProcessStep ............ SUCCESS
[8.524s]

[INFO] com.cqblueprints.example - parent .................. SUCCESS
[0.050s]

[INFO] com.cqblueprints.example - my-cq-project-view ..... SUCCESS
[1.079s]

[INFO] com.cqblueprints.example - my-cq-project-services . SUCCESS
[19.532s]

[INFO] com.cqblueprints.example - my-cq-project-config ... SUCCESS
[0.462s]

[INFO] com.cqblueprints.example - my-cq-project-taglib ... SUCCESS
[3:11.055s]

[INFO] com.cqblueprints.example - my-cq-project-content .. SUCCESS
[0.481s]

[INFO] com.cqblueprints.example - my-cq-project-all ...... SUCCESS
[0.461s]

[INFO] ------------------------------------------------------------------------
------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------
------

[INFO] Total time: 4:25.841s

[INFO] Finished at: Mon Sep 17 12:02:42 EDT 2012

[INFO] Final Memory: 36M/172M

[INFO] ------------------------------------------------------------------------
------
```

To create the OSGi custom workflow step bundle by running Maven, perform the following step:

1. Start the Command Prompt.

2. Change the directory to the location of CQWorkflowProcessStep (the Maven application template).

3. Enter the following command: `%M3_HOME%\bin\mvn clean install.`

Note: The OSGi bundle will be written out to the CQWorkflowProcessStep\target folder. By default, the filename of the OSGi bundle is CQWorkflowProcessStep-1.0.0.0.jar.

Deploy the custom workflow step to Adobe CQ

To the top

After you build the custom workflow step, deploy it to Adobe CQ by performing the following tasks:

1. Login to Adobe CQ's Apache Felix Web Console at http://server:port/system/console/bundles (default admin user = admin with password= admin).

2. Sort the bundle list by "Id" and note the Id of the last bundle.

3. Click the "Install/Update" button.

4. Check the "Start Bundle" checkbox.

5. Browse to the OSGi bundle JAR file you just built using Maven. (CQWorkflowProcessStep\target folder).

6. Click "Install or Update".

7. Click the 'Refresh Packages' button.

8. Check the bundle with the highest Id.

9. Your new bundle should now be listed with the status 'Active'.

Create an Adobe CQ workflow with the custom step

To the top

You can create an Adobe CQ workflow that uses the new custom step. To create a workflow, you

use the Adobe CQ Workflow console located at the following URL:

```
http://localhost:4503/libs/cq/workflow/content/console.html
```

To create a workflow with a custom step, perform the following tasks:

1. Log into the Adobe CQ Workflow console.

2. Choose the Models tab and create a workflow by choosing New from the toolbar.

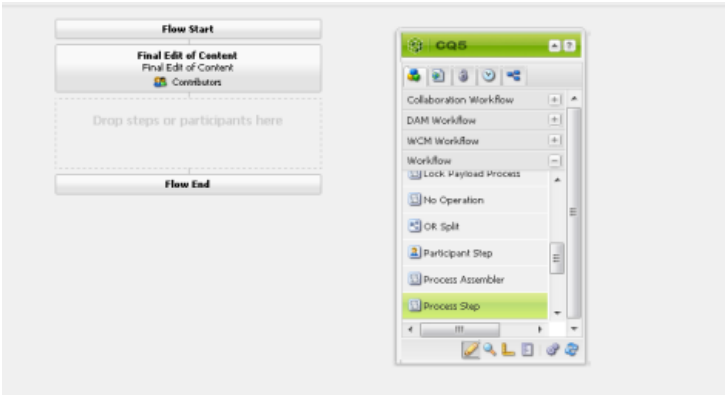
3. Enter Approval as the workflow title.

4. Open the Approval workflow by double-clicking on the name located in the grid view.The Approval workflow model has a Start, Step 1, and an End.

5. Edit Step 1 by double-clicking on the step. Enter the following property values

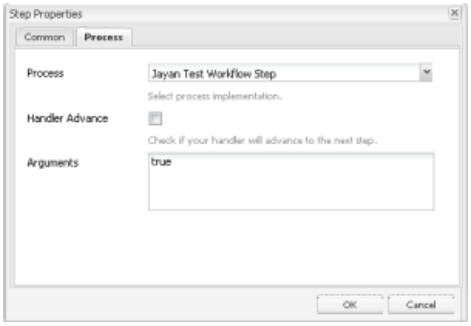
- **Description** = Final Edit of Content
- **Title** = Content Validation
- **User/Group** = Contributors

6. Add a Process step to the workflow by dragging-and-dropping the Process Step component from the sidekick onto the workflow model.



7. Open the Process Step dialog and set the properties of your new step:

- **Description** = Set approved property
- **Implementation** = Jayon Test Workflow Step (This value was specified in the Java file using the @Property annotation)
- **Title** = Final Approval
- **Process Arguments** = true



8. Click Save.

9. Test your new workflow. In the WebSites panel, select any page. Bring up the right context menu and choose Workflow.

10. Select the new Approval workflow and click Start.

11. Monitor the progress of your workflow from the inbox and move the page through the workflow. Check for the approved property on the page once the workflow has completed.