Adobe Experience Manager Help / Uploading files to Adobe Experience Manager

Article summary

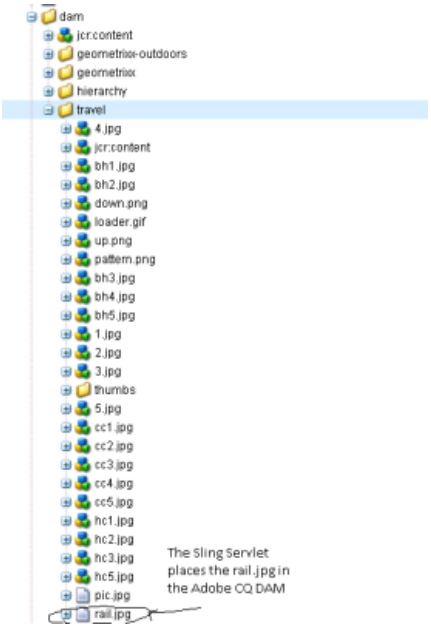
Summary	Discusses how to create an AEM application that lets users select image files and upload them to a Java Sling Servlet. Once uploaded, the Servlet uses the JCR API to store the image file in the Adobe CQ DAM. This article uses an Adobe Maven Archetype project to build an OSGi bundle. If you are not familiar with an Adobe Maven Archetype project, it is recommended that you read the following article: Creating your first AEM Service using an Adobe Maven Archetype project.
Digital Marketing Solution(s)	Adobe Experience Manager (Adobe CQ)
Audience	Developer (intermediate)
Required Skills	Java, JCR API, JavaScript, AJAX, HTML
Tested On	Adobe CQ 5.5, Adobe CQ 5.6

Introduction

You can create an Adobe CQ application that lets a user select a file from their local desktop and upload it to Adobe Experience Manager. The file is posted to a custom Sling Servlet that is able to handle the file and perform a given task. For example, the servlet can place an image file in the Adobe CQ Digital Asset Manager (DAM).

An Adobe AEM client web page that lets a user select a file and upload it to AEM

In this example, notice that a file named rail.jpg is selected. Once the file is uploaded, the Sling Servlet persists the file in the Adobe CQ DAM, as shown in the following illustration.



A file is located in the Adobe CQ DAM that was uploaded using a Sling Servlet

Note: Before following along with this development article, create a folder in the AEM DAM named travel located at /content/dam/. For information, see Managing Digital Assets.

This development article walks you through how to create this CQ application that lets a user select and upload a file to the AEM DAM. Once uploaded, the Java Sling Servlet places the file into the AEM DAM using the JCR API.

To create an Adobe CQ application that uploads a file to Adobe CQ and saves the file in the DAM, perform these tasks:

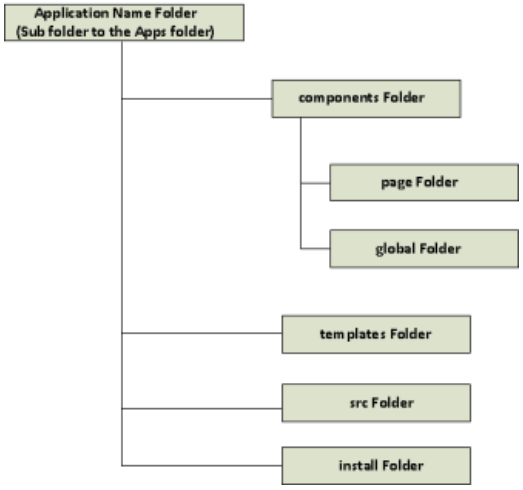
1. Create an Adobe CQ application folder structure.
2. Create a template on which the page component is based.
3. Create a render component that uses the template.
4. Setup Maven in your development environment.
5. Create an Adobe CQ archetype project.
6. Add Java files that represent the Sling Servet (that handles uploaded files) to the Maven project.
7. Modify the Maven POM file.
8. Build the OSGi bundle using Maven.
9. Deploy the bundle to Adobe CQ.
10. Add CSS and JQuery files to a cq:ClientLibraryFolder node.
11. Modify the render component to post files to the Sling Servlet.
12. Create a site that contains a page that lets a user select and upload a file.

Note: This development article dicusses how to create a component that lets the user choose a file to upload to the AEM DAM. In contrast, you can create a Java application using the JCR API that uploads multiple files to the AEM DAM. For information, see http://helpx.adobe.com/experience-manager/using/multiple-digital-assets.html.

Create a CQ application folder structure

To the top

Create an Adobe CQ application folder structure that contains templates, components, and pages by using CRXDE Lite.



A CQ application folder structure

The following describes each application folder:

- **application name**: contains all of the resources that an application uses. The resources can be templates, pages, components, and so on.
- **components**: contains components that your application uses.
- **page**: contains page components. A page component is a script such as a JSP file.
- **global**: contains global components that your application uses.
- **template**: contains templates on which you base page components.
- **src**: contains source code that comprises an OSGi component (this development article does not create an OSGi bundle using this folder).
- **install**: contains a compiled OSGi bundles container.

To create an application folder structure:

1. To view the CQ welcome page, enter the URL http://[host name]:[port] into a web browser. For example, http://localhost:4502.
2. Select CRXDE Lite.
3. Right-click the apps folder (or the parent folder), select Create, Create Folder.
4. Enter the folder name into the Create Folder dialog box. Enter *slingFile*.
5. Repeat steps 1-4 for each folder specified in the previous illustration.
6. Click the Save All button.

Note: You have to click the Save All button when working in CRXDELite for the changes to be made.

Create a template

To the top

You can create a template by using CRXDE Lite. A CQ template enables you to define a consistent style for the pages in your application. A template comprises of nodes that specify the page structure. For more information about templates, see http://dev.day.com/docs/en/cq/current/developing/templates.html.

To create a template, perform these tasks:

1. To view the CQ welcome page, enter the URL http://[host name]:[port] into a web browser. For example, http://localhost:4502.
2. Select CRXDE Lite.
3. Right-click the template folder (within your application), select Create, Create Template.
4. Enter the following information into the Create Template dialog box:

- **Label**: The name of the template to create. Enter *templateUpload*.
- **Title**: The title that is assigned to the template.
- **Description**: The description that is assigned to the template.
- **Resource Type**: The component's path that is assigned to the template and copied to implementing pages. Enter *slingFile/components/page/templateUpload*.

- **Ranking**: The order (ascending) in which this template will appear in relation to other templates. Setting this value to 1 ensures that the template appears first in the list.

5. Add a path to Allowed Paths. Click on the plus sign and enter the following value: */content(/.*)?*.
6. Click Next for Allowed Parents.
7. Select OK on Allowed Children.

To the top

Create a render component that uses the template

Components are re-usable modules that implement specific application logic to render the content of your web site. You can think of a component as a collection of scripts (for example, JSPs, Java servlets, and so on) that completely realize a specific function. In order to realize this functionality, it is your responsibility as a CQ developer to create scripts that perform specific functionality. For more information about components, see http://dev.day.com/docs/en/cq/current/developing/components.html.

By default, a component has at least one default script, identical to the name of the component. To create a render component, perform these tasks:
1. To view the CQ welcome page, enter the URL http://[host name]:[port] into a web browser. For example, http://localhost:4502.
2. Select CRXDE Lite.
3. Right-click */apps/slingFile/components/page*, then select
Create, Create Component.
4. Enter the following information into the Create Component dialog box:

- **Label**: The name of the component to create. Enter *templateUpload*.
- **Title**: The title that is assigned to the component.
- **Description**: The description that is assigned to the template.

5. Select Next for Advanced Component Settings and Allowed Parents.
6. Select OK on Allowed Children.
7. Open the slingTemplateJCR.jsp located at:
/apps/slingFile/components/page/templateUpload/templateUpload.jsp.
8. Enter the following JSP code.

```
<html>
<head>
<title>Hello World !!!</title>
</head>
<body>
<h1>Hello Sling Servlet!!!</h1>
<h2>This page will upload a file to Adobe CQ</h2>
</body>
</html>
```

Setup Maven in your development environment

To the top

You can use Maven to build an OSGi bundle that contains a Sling Servlet. Maven manages required JAR files that a Java project needs in its class path. Instead of searching the Internet trying to find and download third-party JAR files to include in your project's class path, Maven manages these dependencies for you.

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
```

Set up a system environment variable to reference Maven. To test whether you properly setup Maven, enter the following Maven command into a command prompt:

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

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

```
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"
```

Note: For more information about setting up Maven and the Home variable, see: Maven in 5 Minutes.

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\scottm\.m2\.

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 code represents a settings.xml file that you can use.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Licensed to the Apache Software Foundation (ASF) under one
or more contributor license agreements.  See the NOTICE file
distributed with this work for additional information
regarding copyright ownership.  The ASF licenses this file
to you under the Apache License, Version 2.0 (the
"License"); you may not use this file except in compliance
with the License.  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing,
software distributed under the License is distributed on an
"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied.  See the License for the
specific language governing permissions and limitations
under the License.
-->

<!--
 | This is the configuration file for Maven. It can be specified at two levels:
 |
 |  1. User Level. This settings.xml file provides configuration for a single u
 |                 and is normally provided in ${user.home}/.m2/settings.xml.
 |
 |                 NOTE: This location can be overridden with the CLI option:
 |
 |                 -s /path/to/user/settings.xml
 |
 |  2. Global Level. This settings.xml file provides configuration for all Mave
 |                 users on a machine (assuming they're all using the same Mave
 |                 installation). It's normally provided in
 |                 ${maven.home}/conf/settings.xml.
 |
 |                 NOTE: This location can be overridden with the CLI option:
 |
 |                 -gs /path/to/global/settings.xml
 |
 | The sections in this sample file are intended to give you a running start at
 | getting the most out of your Maven installation. Where appropriate, the defa
 | values (values used when the setting is not specified) are provided.
 |
 |-->
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://mav
  <!-- localRepository
   | The path to the local repository maven will use to store artifacts.
   |
   | Default: ~/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  -->

  <!-- interactiveMode
   | This will determine whether maven prompts you when it needs input. If set
   | maven will use a sensible default value, perhaps based on some other setti
   | the parameter in question.
   |
   | Default: true
  <interactiveMode>true</interactiveMode>
  -->

  <!-- offline
   | Determines whether maven should attempt to connect to the network when exe
   | This will have an effect on artifact downloads, artifact deployment, and o
   |
   | Default: false
  <offline>false</offline>
  -->

  <!-- pluginGroups
   | This is a list of additional group identifiers that will be searched when
   | when invoking a command line like "mvn prefix:goal". Maven will automatica
   | "org.apache.maven.plugins" and "org.codehaus.mojo" if these are not alread
```

```
    |-->
  <pluginGroups>
    <!-- pluginGroup
     | Specifies a further group identifier to use for plugin lookup.
    <pluginGroup>com.your.plugins</pluginGroup>
    -->
  </pluginGroups>

  <!-- proxies
   | This is a list of proxies which can be used on this machine to connect to
   | Unless otherwise specified (by system property or command-line switch), th
   | specification in this list marked as active will be used.
   |-->
  <proxies>
    <!-- proxy
     | Specification for one proxy, to be used in connecting to the network.
     |
    <proxy>
      <id>optional</id>
      <active>true</active>
      <protocol>http</protocol>
      <username>proxyuser</username>
      <password>proxypass</password>
      <host>proxy.host.net</host>
      <port>80</port>
      <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
    </proxy>
    -->
  </proxies>

  <!-- servers
   | This is a list of authentication profiles, keyed by the server-id used wit
   | Authentication profiles can be used whenever maven must make a connection
   |-->
  <servers>
    <!-- server
     | Specifies the authentication information to use when connecting to a par
     | a unique name within the system (referred to by the 'id' attribute belov
     |
     | NOTE: You should either specify username/password OR privateKey/passphra
     |       used together.
     |
    <server>
      <id>deploymentRepo</id>
      <username>repouser</username>
      <password>repopwd</password>
    </server>
    -->

    <!-- Another sample, using keys to authenticate.
    <server>
      <id>siteServer</id>
      <privateKey>/path/to/private/key</privateKey>
      <passphrase>optional; leave empty if not used.</passphrase>
    </server>
    -->
  </servers>

  <!-- mirrors
   | This is a list of mirrors to be used in downloading artifacts from remote
   |
   | It works like this: a POM may declare a repository to use in resolving cer
   | However, this repository may have problems with heavy traffic at times, so
   | it to several places.
   |
   | That repository definition will have a unique id, so we can create a mirro
   | repository, to be used as an alternate download site. The mirror site will
   | server for that repository.
   |-->
  <mirrors>
    <!-- mirror
     | Specifies a repository mirror site to use instead of a given repository.
     | this mirror serves has an ID that matches the mirrorOf element of this r
     | for inheritance and direct lookup purposes, and must be unique across th
     |
    <mirror>
      <id>mirrorId</id>
      <mirrorOf>repositoryId</mirrorOf>
      <name>Human Readable Name for this Mirror.</name>
      <url>http://my.repository.com/repo/path</url>
    </mirror>
     -->
  </mirrors>

```

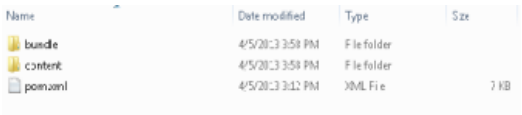
```
  <!-- profiles
   | This is a list of profiles which can be activated in a variety of ways, ar
   | the build process. Profiles provided in the settings.xml are intended to p
   | specific paths and repository locations which allow the build to work in t
   |
   | For example, if you have an integration testing plugin - like cactus - tha
   | your Tomcat instance is installed, you can provide a variable here such th
   | dereferenced during the build process to configure the cactus plugin.
   |
   | As noted above, profiles can be activated in a variety of ways. One way -
   | section of this document (settings.xml) - will be discussed later. Another
   | relies on the detection of a system property, either matching a particular
   | or merely testing its existence. Profiles can also be activated by JDK ver
   | value of '1.4' might activate a profile when the build is executed on a JD
   | Finally, the list of active profiles can be specified directly from the co
   |
   | NOTE: For profiles defined in the settings.xml, you are restricted to spec
   |       repositories, plugin repositories, and free-form properties to be us
   |       variables for plugins in the POM.
   |
   |-->
  <profiles>
    <!-- profile
     | Specifies a set of introductions to the build process, to be activated u
     | mechanisms described above. For inheritance purposes, and to activate pr
     | or the command line, profiles have to have an ID that is unique.
     |
     | An encouraged best practice for profile identification is to use a consi
     | for profiles, such as 'env-dev', 'env-test', 'env-production', 'user-jdk
     | This will make it more intuitive to understand what the set of introduce
     | to accomplish, particularly when you only have a list of profile id's fo
     |
     | This profile example uses the JDK version to trigger activation, and pro
    <profile>
      <id>jdk-1.4</id>

      <activation>
        <jdk>1.4</jdk>
      </activation>

      <repositories>
        <repository>
          <id>jdk14</id>
          <name>Repository for JDK 1.4 builds</name>
          <url>http://www.myhost.com/maven/jdk14</url>
          <layout>default</layout>
          <snapshotPolicy>always</snapshotPolicy>
        </repository>
      </repositories>
    </profile>
    -->

    <!--
     | Here is another profile, activated by the system property 'target-env' w
     | which provides a specific path to the Tomcat instance. To use this, your
     | might hypothetically look like:
     |
     | ...
     | <plugin>
     |   <groupId>org.myco.myplugins</groupId>
     |   <artifactId>myplugin</artifactId>
     |
     |   <configuration>
     |     <tomcatLocation>${tomcatPath}</tomcatLocation>
     |   </configuration>
     | </plugin>
     | ...
     |
     | NOTE: If you just wanted to inject this configuration whenever someone s
     |       anything, you could just leave off the <value/> inside the activat
     |
    <profile>
      <id>env-dev</id>

      <activation>
        <property>
          <name>target-env</name>
          <value>dev</value>
        </property>
      </activation>

      <properties>
        <tomcatPath>/path/to/tomcat/instance</tomcatPath>
      </properties>
```

```
    </profile>
    -->


<profile>

                <id>adobe-public</id>

                <activation>

                    <activeByDefault>true</activeByDefault>

                </activation>

                <repositories>

                  <repository>

                    <id>adobe</id>

                    <name>Nexus Proxy Repository</name>

                    <url>http://repo.adobe.com/nexus/content/groups/public/</ur

                    <layout>default</layout>

                  </repository>

                </repositories>

                <pluginRepositories>

                  <pluginRepository>

                    <id>adobe</id>

                    <name>Nexus Proxy Repository</name>

                    <url>http://repo.adobe.com/nexus/content/groups/public/</ur

                    <layout>default</layout>

                  </pluginRepository>

                </pluginRepositories>

            </profile>

</profiles>

  <!-- activeProfiles
   | List of profiles that are active for all builds.
   |
  <activeProfiles>
    <activeProfile>alwaysActiveProfile</activeProfile>
    <activeProfile>anotherAlwaysActiveProfile</activeProfile>
  </activeProfiles>
  -->
</settings>
```

Create an Adobe CQ archetype project

To the top

You can create an Adobe CQ archetype project by using the Maven archetype plugin. In this example, assume that the working directory is C:\AdobeCQ.



Default files created by the Maven archetype plugin

To create an Adobe CQ archetype project, perform these steps:

1. Open the command prompt and go to your working directory (for example, C:\AdobeCQ).

2. Run the following Maven command:

```
mvn archetype:generate -DarchetypeGroupId=com.day.jcr.vault -
DarchetypeArtifactId=multimodule-content-package-archetype -
DarchetypeVersion=1.0.0 -DarchetypeRepository=adobe-public-releases
```

3. When prompted for additional information, specify these values:

- **groupId**: com.adobe.cq.sling.upload
- **artifactId**: upload
- **version**: 1.0-SNAPSHOT
- **package**: com.adobe.cq.sling.upload
- **appsFolderName**: adobe-training
- **artifactName**: Upload Training Package
- **packageGroup**: adobe training
- **confirm**: Y

4. Once done, you will see a message like:
[[INFO] Total time: 14:46.131s
[INFO] Finished at: Wed Mar 27 13:38:58 EDT 2013
[INFO] Final Memory: 10M/184M

5. Change the command prompt to the generated project. For example: C:\AdobeCQ\upload. Run the following Maven command:

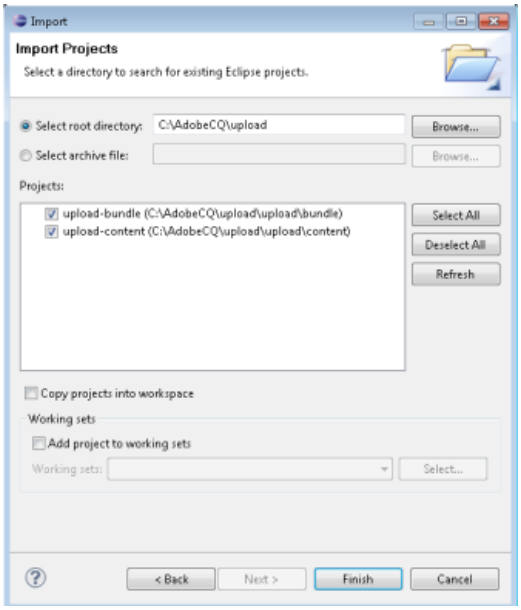
```
mvn eclipse:eclipse
```

After you run this command, you can import the project into Eclipse as discussed in the next section.

Add Java files to the Maven project using Eclipse

To the top

To make it easier to work with the Maven generated project, import it into the Eclipse development environment, as shown in the following illustration.



The Eclipse Import Project dialog

The next step is to add a Java file to the `com.adobe.cq.sling.upload` package. The Java class that you create in this section extends the Sling class named `org.apache.sling.api.servlets.SlingAllMethodsServlet`. This class supports the `doPost` method that lets you submit data from an Adobe CQ web page to the Sling servlet. In this example, a file is uploaded from a CQ web page to the sling servlet.

For information about this class, see Class SlingAllMethodsServlet.

Create a Java class named `HandleFile` that extends `org.apache.sling.api.servlets.SlingAllMethodsServlet`. Within the `doPost` method, create Java Sling application logic that reads the file that is uploaded to the Sling servlet. The fully qualified names of the Java objects are used so you understand the data types used in this code fragment.

```
1 @Override
2     protected void doPost(SlingHttpServletRequest request, SlingHttpServletResp
```

```
3
4
5           try
6           {
7           final boolean isMultipart = org.apache.commons.fileupload.servlet.Serv
8           PrintWriter out = null;
9
10            out = response.getWriter();
11            if (isMultipart) {
12              final java.util.Map<String, org.apache.sling.api.request.RequestPa
13              for (final java.util.Map.Entry<String, org.apache.sling.api.reques
14                final String k = pairs.getKey();
15                final org.apache.sling.api.request.RequestParameter[] pArr = pai
16                final org.apache.sling.api.request.RequestParameter param = pArr
17                final InputStream stream = param.getInputStream();
18
19                    //Save the uploaded file into the Adobe CQ DAM
20                 out.println("The Sling Servlet placed the uploaded file here: "
21
22              }
23            }
24          }
25
26          catch (Exception e) {
27              e.printStackTrace();
28          }
29
30       }
```

The uploaded file is placed into an `InputStream` instance named `stream`. Next, the uploaded file is written to the Adobe CQ DAM using the JCR API. The following Java code represents a method named `writeToDam`. This method uses the JCR API to place the file into the following CQ DAM location:

`/content/dam/travel`

The `writeToDam` method accepts the `InputStream` instance (the uploaded file) and the uploaded file name as parameters.

```
1  //Save the uploaded file into the Adobe CQ DAM
2  private String writeToDam(InputStream is, String fileName)
3  {
4  try
5  {
6      //Invoke the adaptTo method to create a Session
7          ResourceResolver resourceResolver =     resolverFactory.getAdministrati
8          session = resourceResolver.adaptTo(Session.class);
9      Node node = session.getNode("/content/dam/travel");
10     javax.jcr.ValueFactory valueFactory = session.getValueFactory();
11     javax.jcr.Binary contentValue = valueFactory.createBinary(is);
12     Node fileNode = node.addNode(fileName, "nt:file");
13     fileNode.addMixin("mix:referenceable");
14     Node resNode = fileNode.addNode("jcr:content", "nt:resource");
15     resNode.setProperty("jcr:mimeType", "image/jpeg");
16     resNode.setProperty("jcr:data", contentValue);
17     Calendar lastModified = Calendar.getInstance();
18     lastModified.setTimeInMillis(lastModified.getTimeInMillis());
19     resNode.setProperty("jcr:lastModified", lastModified);
20     session.save();
21         session.logout();
22
23
24     // Return the path to the document that was stored in CRX.
25     return fileNode.getPath();
26 }
27 catch(Exception e)
28 {
29     e.printStackTrace();
30 }
31 return null;
32 }
```

The following Java code represents the `HandleFile` class that extends `org.apache.sling.api.servlets.SlingAllMethodsServlet`.

```
1  package com.adobe.cq.upload;
2
3
4  import java.io.BufferedReader;
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.io.InputStreamReader;
```

```
 8  import java.io.PrintWriter;
 9  import java.net.HttpURLConnection;
10  import java.net.URL;
11  import java.rmi.ServerException;
12  import java.util.Dictionary;
13  import java.util.Calendar;
14  import java.io.*;
15
16  import org.apache.felix.scr.annotations.Properties;
17  import org.apache.felix.scr.annotations.Property;
18  import org.apache.felix.scr.annotations.Reference;
19  import org.apache.felix.scr.annotations.sling.SlingServlet;
20  import org.apache.sling.api.SlingHttpServletRequest;
21  import org.apache.sling.api.SlingHttpServletResponse;
22  import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
23  import org.apache.sling.commons.osgi.OsgiUtil;
24  import org.apache.sling.jcr.api.SlingRepository;
25  import org.apache.felix.scr.annotations.Reference;
26  import org.osgi.service.component.ComponentContext;
27  import javax.jcr.Session;
28  import javax.jcr.Node;
29  import org.apache.commons.fileupload.FileItem;
30  import org.apache.commons.fileupload.disk.DiskFileItemFactory;
31  import org.apache.commons.fileupload.servlet.ServletFileUpload;
32  import org.apache.commons.fileupload.util.Streams;
33  import org.apache.felix.scr.annotations.Component;
34  import org.apache.felix.scr.annotations.Service;
35  import javax.jcr.ValueFactory;
36  import javax.jcr.Binary;
37
38  import javax.servlet.Servlet;
39  import javax.servlet.ServletException;
40  import javax.servlet.http.HttpServlet;
41  import javax.servlet.http.HttpServletRequest;
42  import javax.servlet.http.HttpServletResponse;
43  import java.io.FileOutputStream;
44  import java.util.Iterator;
45  import java.util.List;
46  import java.io.OutputStream;
47  import org.slf4j.Logger;
48  import org.slf4j.LoggerFactory;
49
50  import java.io.StringWriter;
51
52  import java.util.ArrayList;
53
54  import javax.jcr.Repository;
55  import javax.jcr.SimpleCredentials;
56  import javax.xml.parsers.DocumentBuilder;
57  import javax.xml.parsers.DocumentBuilderFactory;
58
59  import org.apache.jackrabbit.commons.JcrUtils;
60
61  import javax.xml.transform.Transformer;
62  import javax.xml.transform.TransformerFactory;
63  import javax.xml.transform.dom.DOMSource;
64  import javax.xml.transform.stream.StreamResult;
65
66
67
68  import javax.jcr.Session;
69  import javax.jcr.Node;
70
71  //Sling Imports
72  import org.apache.sling.api.resource.ResourceResolverFactory ;
73  import org.apache.sling.api.resource.ResourceResolver;
74  import org.apache.sling.api.resource.Resource;
75
76  //This is a component so it can provide or consume services
77  @SlingServlet(paths="/bin/upfile", methods = "POST", metatype=true)
78  public class HandleFile extends org.apache.sling.api.servlets.SlingAllMethodsSe
79   private static final long serialVersionUID = 2598426539166789515L;
80
81   private Session session;
82
83   //Inject a Sling ResourceResolverFactory
84   @Reference
85   private ResourceResolverFactory resolverFactory;
86
87   @Override
88   protected void doGet(SlingHttpServletRequest request, SlingHttpServletRespons
89
90
91       }
```

```
92
93
94    @Override
95    protected void doPost(SlingHttpServletRequest request, SlingHttpServletRespons
96
97         try
98            {
99            final boolean isMultipart = org.apache.commons.fileupload.servlet.Serv
100           PrintWriter out = null;
101
102             out = response.getWriter();
103             if (isMultipart) {
104               final java.util.Map<String, org.apache.sling.api.request.RequestPa
105               for (final java.util.Map.Entry<String, org.apache.sling.api.reques
106                 final String k = pairs.getKey();
107                 final org.apache.sling.api.request.RequestParameter[] pArr = pai
108                 final org.apache.sling.api.request.RequestParameter param = pArr
109                 final InputStream stream = param.getInputStream();
110
111                     //Save the uploaded file into the Adobe CQ DAM
112                  out.println("The Sling Servlet placed the uploaded file here: '
113
114               }
115             }
116           }
117
118           catch (Exception e) {
119               e.printStackTrace();
120           }
121
122       }
123
124
125   //Save the uploaded file into the Adobe CQ DAM
126   private String writeToDam(InputStream is, String fileName)
127   {
128   try
129   {
130       //Invoke the adaptTo method to create a Session
131       ResourceResolver resourceResolver = resolverFactory.getAdministrativeResour
132       session = resourceResolver.adaptTo(Session.class);
133
134       Node node = session.getNode("/content/dam/travel");
135       javax.jcr.ValueFactory valueFactory = session.getValueFactory();
136       javax.jcr.Binary contentValue = valueFactory.createBinary(is);
137       Node fileNode = node.addNode(fileName, "nt:file");
138       fileNode.addMixin("mix:referenceable");
139       Node resNode = fileNode.addNode("jcr:content", "nt:resource");
140       resNode.setProperty("jcr:mimeType", "image/jpeg");
141       resNode.setProperty("jcr:data", contentValue);
142       Calendar lastModified = Calendar.getInstance();
143       lastModified.setTimeInMillis(lastModified.getTimeInMillis());
144       resNode.setProperty("jcr:lastModified", lastModified);
145       session.save();
146        session.logout();
147
148       // Return the path to the document that was stored in CRX.
149       return fileNode.getPath();
150   }
151   catch(Exception e)
152   {
153       e.printStackTrace();
154   }
155   return null;
156   }
157
158   }
```

The Java class uses a `SlingServlet` annotation:

```
@SlingServlet(paths="/bin/upfile", methods = "POST", metatype=true)
```

The paths property corresponds to the URL that you specify when using an AJAX request. That is, to use an AJAX request to post data to this Sling Servlet, you use this syntax:

```
//Use JQuery AJAX request to post data to a Sling Servlet
$.ajax({
    type: 'POST',
    url:'/bin/upfile',
    processData: false,
    contentType: false,
    data:formData,
    success: function(msg){
```

```
        alert(msg); //display the data returned by the servlet
}
});
```

Notice that the `url` in the AJAX request maps to the path property in the `SlingServlet` annotation. The type in the AJAX request maps to the methods property in the `SlingServlet` annotation. Finally notice that the AJAX request specifies the form data that is submitted. The uploaded file is located in the `formData` variable.

Note: This AJAX request is used in the client web page that is created later in this development article.

Modify the Maven POM file

To the top

Modify the POM files to successfully build the OSGi bundle that contains the Sling servlet. In the POM file located at C:\AdobeCQ\claimupload, add the following dependencies.

- org.apache.felix.scr
- org.apache.felix.scr.annotations
- org.apache.jackrabbit
- org.apache.sling

The following XML represents this POM file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
    <modelVersion>4.0.0</modelVersion>
    <!-- ======================================================================
    <!-- P A R E N T P R O J E C T D E S C R I P T I O N -->
    <!-- ======================================================================
    <parent>
        <groupId>com.adobe.cq.upload</groupId>
        <artifactId>upload</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <!-- ======================================================================
    <!-- P R O J E C T D E S C R I P T I O N -->
    <!-- ======================================================================

    <artifactId>upload-bundle</artifactId>
    <packaging>bundle</packaging>
    <name>Upload Package Bundle</name>

    <!-- ======================================================================
    <!-- B U I L D D E F I N I T I O N -->
    <!-- ======================================================================
    <build>

        <plugins>
            <plugin>
                <groupId>org.apache.felix</groupId>
                <artifactId>maven-scr-plugin</artifactId>
                <executions>
                    <execution>
                        <id>generate-scr-descriptor</id>
                        <goals>
                            <goal>scr</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.felix</groupId>
                <artifactId>maven-bundle-plugin</artifactId>
                <extensions>true</extensions>
                <configuration>
                    <instructions>
                        <Bundle-SymbolicName>com.adobe.cq.upload.upload-bundle<
                    </instructions>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.sling</groupId>
                <artifactId>maven-sling-plugin</artifactId>
                <configuration>
                    <slingUrl>http://${crx.host}:${crx.port}/apps/upload-train
                    <usePut>true</usePut>
                </configuration>
```

```
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
    </dependency>

    <dependency>
        <groupId>org.apache.sling</groupId>
        <artifactId>org.apache.sling.commons.osgi</artifactId>
        <version>2.2.0</version>
    </dependency>

    <dependency>
        <groupId>org.apache.sling</groupId>
        <artifactId>org.apache.sling.jcr.api</artifactId>
        <version>2.0.4</version>
      </dependency>


    <dependency>
            <groupId>commons-fileupload</groupId>
            <artifactId>commons-fileupload</artifactId>
            <version>1.2</version>
        </dependency>

        <dependency>
         <groupId>org.apache.felix</groupId>

         <artifactId>org.osgi.core</artifactId>

         <version>1.4.0</version>
      </dependency>



      <dependency>
            <groupId>org.apache.sling</groupId>
            <artifactId>org.apache.sling.api</artifactId>
            <version>2.2.4</version>
            <scope>provided</scope>
        </dependency>


    <dependency>
            <groupId>org.osgi</groupId>
            <artifactId>org.osgi.compendium</artifactId>
        </dependency>
        <dependency>
            <groupId>org.osgi</groupId>
            <artifactId>org.osgi.core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.felix</groupId>
            <artifactId>org.apache.felix.scr.annotations</artifactId>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
        </dependency>


        <dependency>
         <groupId>org.apache.felix</groupId>

         <artifactId>org.osgi.core</artifactId>

         <version>1.4.0</version>
      </dependency>



    <dependency>
    <groupId>org.apache.jackrabbit</groupId>
    <artifactId>jackrabbit-core</artifactId>
    <version>2.4.3</version>
    </dependency>

    <dependency>
    <groupId>org.apache.jackrabbit</groupId>
```

```
     <artifactId>jackrabbit-jcr-commons</artifactId>
     <version>2.4.3</version>
     </dependency>
    <dependency>
             <groupId>junit</groupId>
             <artifactId>junit</artifactId>
         </dependency>

     <dependency>
             <groupId>org.apache.sling</groupId>
             <artifactId>org.apache.sling.api</artifactId>
             <version>2.2.4</version>
             <scope>provided</scope>
         </dependency>

       <dependency>
          <groupId>javax.jcr</groupId>
          <artifactId>jcr</artifactId>
          <version>2.0</version>
       </dependency>

        <dependency>
             <groupId>com.day.cq.wcm</groupId>
             <artifactId>cq-wcm-api</artifactId>
             <version>5.5.0</version>
             <scope>provided</scope>
         </dependency>

         <dependency>
             <groupId>com.day.cq</groupId>
             <artifactId>cq-commons</artifactId>
             <version>5.5.0</version>
             <scope>provided</scope>
         </dependency>


     </dependencies>

     <repositories>
         <repository>
             <id>adobe</id>
             <name>Adobe Public Repository</name>
             <url>http://repo.adobe.com/nexus/content/groups/public/</url>
             <layout>default</layout>
         </repository>
     </repositories>
     <pluginRepositories>
         <pluginRepository>
             <id>adobe</id>
             <name>Adobe Public Repository</name>
             <url>http://repo.adobe.com/nexus/content/groups/public/</url>
             <layout>default</layout>
         </pluginRepository>
     </pluginRepositories>

 </project>
```

Build the OSGi bundle using Maven

To the top

Build the OSGi bundle by using Maven. When Maven builds the bundle, it also creates a serviceComponents.xml file based on the annotations that are included in the `com.adobe.cq.slingupload.HandleFile` class. The following XML represents this file.

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns:scr="http://www.osgi.org/xmlns/scr/v1.0.0">
    <scr:component enabled="true" name="com.adobe.cq.upload.SimpleDSComponent">
        <implementation class="com.adobe.cq.upload.SimpleDSComponent"/>
        <service servicefactory="false">
            <provide interface="java.lang.Runnable"/>
        </service>
        <property name="service.pid" value="com.adobe.cq.upload.SimpleDSComponer
    </scr:component>
    <scr:component enabled="true" name="com.adobe.cq.upload.HandleFile">
        <implementation class="com.adobe.cq.upload.HandleFile"/>
        <service servicefactory="false">
            <provide interface="javax.servlet.Servlet"/>
        </service>
        <property name="sling.servlet.paths" value="/bin/upfile"/>
        <property name="sling.servlet.methods" value="POST"/>
        <property name="service.pid" value="com.adobe.cq.upload.HandleFile"/>
        <reference name="resolverFactory" interface="org.apache.sling.api.resou
    </scr:component>
```

```
20 </components>
```

Notice that the implementation class element specifies `com.adobe.cq.sling.upload.HandleFile`. This lines up with the Java class that extends `org.apache.sling.api.servlets.SlingAllMethodsServlet` that was created in an earlier step.

To build the OSGi component by using Maven, perform these steps:

1. Open the command prompt and go to the C:\AdobeCQ\upload folder.
2. Run the following maven command: `mvn clean install`.
3. The OSGi component can be found in the following folder: C:\AdobeCQ\upload\bundle\target. The file name of the OSGi component is upload-bundle-1.0-SNAPSHOT.jar.

Deploy the bundle to Adobe CQ

To the top

Once you deploy the OSGi bundle, you can upload a JPG file to the Sling Servlet (this is shown later in this development article). After you deploy the OSGi bundle, you will be able to see it in the Adobe CQ Apache Felix Web Conole.

Apache Felix Web Console Bundles view

Deploy the OSGi bundle that contains the Sling Servlet to Adobe CQ by performing these steps:

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. Click the Bundles tab, sort the bundle list by Id, and note the Id of the last bundle.
3. Click the Install/Update button.
4. Browse to the bundle JAR file you just built using Maven. (C:\AdobeCQ\upload\bundle\target).
5. Click Install.
6. Click the Refresh Packages button.
7. Check the bundle with the highest Id.
8. Click Active.
9. Your new bundle should now be listed with the status Active.
10. If the status is not Active, check the CQ error.log for exceptions.

Add JQuery files to a CQ:ClientLibraryFolder node

To the top

Add the JQuery framework file to a `cq:ClientLibraryFolder` node. The JQuery framework file that is added is named jquery-1.6.3.min.js.

To add the JQuery framework to your component, add a `cq:ClientLibraryFolder` node to your component. After you create the node, set properties that allow the JSP script to find the JQuery library file.

Name	Type	Value
dependencies	String[]	cq.jquery
categories	String[]	jquerysamples

Text files

Add a text file to the clientlibs folder that maps to the JQuery JS file. The name of the text file is js.txt. The js.txt file contains the JS JQeury file name: jquery-1.6.3.min.js.

Add the files to the ClientLibs folder

1. Right-click /apps/slingFile/components then select New, Node.
2. Make sure that the node type is `cq:ClientLibraryFolder` and name the node clientlibs.
3. Right click on clientlibs and select Properties. Add the two properties specified in the previous table to the node.
4. On your file system, navigate to the folder where the JQuery JS file is located. Drag and drop the jquery-1.6.3.min.js file to the clientlibs node by using CRXDE.
5. Add a TXT file to the clientlibs folder named js.txt. The content of the js.txt file is the JQuery JS file name.

Modify the templateUpload JSP to post a file to the Sling Servlet

To the top

Modify the templateUpload.jsp file to post a file to the Sling Servlet that was created in this development article. In this example, a JQuery Ajax Post request is used and the file is posted to the Sling Servlet's `doPost` method (the method defined in the `HandleFile`Java class).

The following code represents the AJAX request.

```
$.ajax({
    type: 'POST',
    url:'/bin/upfile',
    processData: false,
    contentType: false,
    data:formData,
    success: function(msg){
        alert(msg); //display the data returned by the servlet
    }
});
```

Notice that the `url` specifies the value of the `path` attribute in the `SlingServlet` annotation defined in the `HandleFile` method.

```
<%@include file="/libs/foundation/global.jsp"%>
<cq:includeClientLib categories="jquerysamples" />
<script type="text/javascript">

<script>
jQuery(function ($) {

      });

</script>

<body>
<div>
<h2>Upload files to the Adobe CQ DAM</h2>
        <p id="support-notice">Your browser does not support Ajax uploads :-(Th

        <!-- The form starts -->
        <form action="/" method="POST" enctype="multipart/form-data" id="form-

            <!-- The file to upload -->
            <p><input id="file-id" type="file" name="our-file" />

                <!--
                  Also by default, we disable the upload button.
                  If Ajax uploads are supported we'll enable it.
                -->
                <input type="button" value="Upload" id="upload-button-id" disab

            <script>
                // Function that will allow us to know if Ajax uploads are supp
                function supportAjaxUploadWithProgress() {
                    return supportFileAPI() && supportAjaxUploadProgressEvents(

                    // Is the File API supported?
                    function supportFileAPI() {
                        var fi = document.createElement('INPUT');
                        fi.type = 'file';
                        return 'files' in fi;
                    };
```

```
                    // Are progress events supported?
                    function supportAjaxUploadProgressEvents() {
                        var xhr = new XMLHttpRequest();
                        return !! (xhr && ('upload' in xhr) && ('onprogress' in
                    };

                    // Is FormData supported?
                    function supportFormData() {
                        return !! window.FormData;
                    }
                }

                // Actually confirm support
                if (supportAjaxUploadWithProgress()) {
                    // Ajax uploads are supported!
                    // Change the support message and enable the upload button
                    var notice = document.getElementById('support-notice');
                    var uploadBtn = document.getElementById('upload-button-id')
                    notice.innerHTML = "Your browser supports HTML uploads to A
                    uploadBtn.removeAttribute('disabled');

                    // Init the Ajax form submission
                    initFullFormAjaxUpload();

                    // Init the single-field file upload
                    initFileOnlyAjaxUpload();
                }

                function initFullFormAjaxUpload() {
                    var form = document.getElementById('form-id');
                    form.onsubmit = function() {
                        // FormData receives the whole form
                        var formData = new FormData(form);

                        // We send the data where the form wanted
                        var action = form.getAttribute('action');

                        // Code common to both variants
                        sendXHRequest(formData, action);

                        // Avoid normal form submission
                        return false;
                    }
                }

                function initFileOnlyAjaxUpload() {
                    var uploadBtn = document.getElementById('upload-button-id')
                    uploadBtn.onclick = function (evt) {
                        var formData = new FormData();

                        // Since this is the file only, we send it to a specif
                        //   var action = '/upload';

                        // FormData only has the file
                        var fileInput = document.getElementById('file-id');
                        var file = fileInput.files[0];
                        formData.append('our-file', file);

                        // Code common to both variants
                        sendXHRequest(formData);
                    }
                }

                // Once the FormData instance is ready and we know
                // where to send the data, the code is the same
                // for both variants of this technique
                function sendXHRequest(formData) {

                    var test = 0;

                    $.ajax({
                        type: 'POST',
                        url:'/bin/upfile',
                        processData: false,
                        contentType: false,
                        data:formData,
                        success: function(msg){
                          alert(msg); //display the data returned by the servle
                        }
                    });

                }
```

```
                // Handle the start of the transmission
                function onloadstartHandler(evt) {
                    var div = document.getElementById('upload-status');
                    div.innerHTML = 'Upload started!';
                }

                // Handle the end of the transmission
                function onloadHandler(event) {
                    //Refresh the URL for Form Preview
                    var msg = event.target.responseText;

                   alert(msg);
                }

                // Handle the progress
                function onprogressHandler(evt) {
                    var div = document.getElementById('progress');
                    var percent = evt.loaded/evt.total*100;
                    div.innerHTML = 'Progress: ' + percent + '%';
                }

                // Handle the response from the server
                function onreadystatechangeHandler(evt) {
                    var status = null;

                    try {
                        status = evt.target.status;
                    }
                    catch(e) {
                        return;
                    }

                    if (status == '200' && evt.target.responseText) {
                        var result = document.getElementById('result');
                        result.innerHTML = '<p>The server saw it as:</p><pre>'
                    }
                }
            </script>

            <!-- Placeholders for messages set by event handlers -->
            <p id="upload-status"></p>
            <p id="progress"></p>
            <pre id="result"></pre>

        </form>

</div>


</body>
</html>
```

Modify the templateUpload JSP file

1. To view the CQ welcome page, enter the URL: http://[host name]:[port] into a web browser. For example, http://localhost:4502.
2. Select CRXDE Lite.
3. Double-click */apps/slingFile/components/page/templateUpload/templateUpload.jsp*.
4. Replace the JSP code with the new code shown in this section.
5. Click Save All.

Create a CQ web page that lets users upload files to the CQ DAM

To the top

The final task is to create a site that contains a page that is based on the templateUpload (the template created earlier in this development article). When the user selects a file and submits it, the file is persisted in the Adobe CQ DAM.

Create a CQ web page that lets users upload files to the Adobe CQ DAM:

1. Go to the CQ Websites page at http://localhost:4502/siteadmin#/content.
2. Select New Page.
3. Specify the title of the page in the Title field.
4. Specify the name of the page in the Name field.
5. Select *templateUpload* from the template list that appears. This value represents the template that is created in this development article. If you do not see it, then repeat the steps in this development article. For example, if you made a typing mistake when entering in path

information, the template will not show up in the New Page dialog box.

6. Open the new page that you created by double-clicking it in the right pane. The new page opens in a web browser.

See also

Congratulations, you have just created an AEM OSGi bundle that contains a sling servlet by using an Adobe Maven Archetype project. Please refer to the AEM community page for other articles that discuss how to build AEM services/applications by using an Adobe Maven Archetype project.