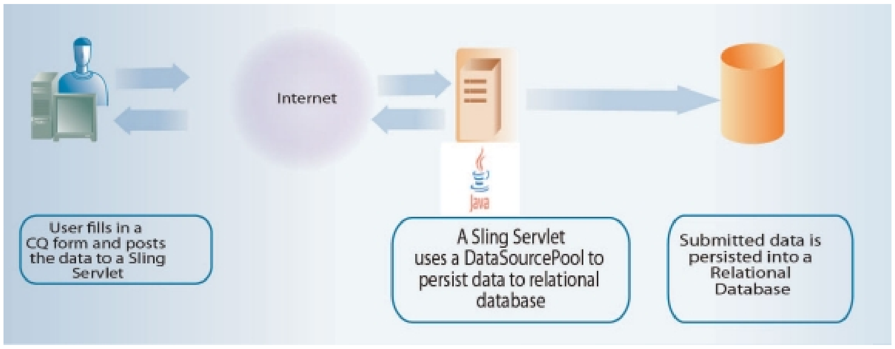
Adobe CQ Help / Injecting a DataSourcePool into Adobe Experience Manager Sling Servlets

Article summary

Summary	Discusses how to create an Apache Sling Servlet in which a DataSourcePool is injected into. The servlet also uses Java JDBC APIs to persist the data into MySQL. This article discusses the following tasks: • how to use Maven to develop the Sling Servlet • how to inject a DataSourcePool instance into the servlet • how to deploy it to AEM • how to post data to the servlet from a client web page 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, JQuery, AJAX, CSS, Maven, JSON, HTML
Tested On	Adobe CQ 5.5, Adobe CQ 5.6

Introduction

You can create an Adobe Experience Manager (AEM) application that lets a user enter data into a web page and post the data to an AEM Sling Servlet. The Sling Servlet can use a `DataSourcePool` to persist the submitted data into a relational database, as shown in the following illustration.


Internet
Java
User fills in a CQ form and posts the data to a Sling Servlet
A Sling Servlet uses a DataSourcePool to persist data to relational database
Submitted data is persisted into a Relational Database

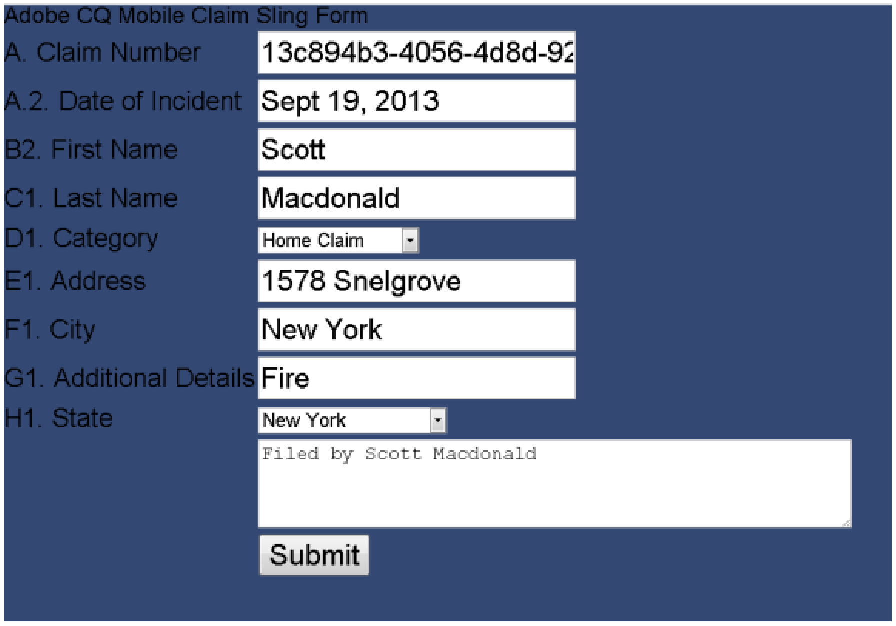
An end user filling in a CQ form and posting the data to a Sling Servlet

The Sling Servlet that is created uses a `DataSourcePool` to persist the data into MySQL. Next the sling servlet encodes the submitted form data into JSON formatted data.

This article guides you through creating a Sling Servlet that uses a `DataSourcePool` to persist the submitted form data into a MySQL table named `Customer`. The following describes the `Customer` table.

Field name	Field Type	Key
custId	An integer that specifies the customer identifier value.	PK
custFirst	A string value that specifies the customer's first name.	N/A
custLast	A string value that specifies the customer's last name.	N/A
custDesc	A string value that specifies the customer's description.	N/A
custAddress	A string value that specifies the customer's address or phone number.	N/A

In addition to persisting the submitted data, the Sling Servlet also encodes the data to JSON. The string (Filed by Scott Macdonald) in the Text Area control located at the bottom of this AEM application is a parsed JSON string. (This is shown later in this development article.)



Adobe CQ Mobile Claim Sling Form	
A. Claim Number	13c894b3-4056-4d8d-92
A.2. Date of Incident	Sept 19, 2013
B2. First Name	Scott
C1. Last Name	Macdonald
D1. Category	Home Claim
E1. Address	1578 Snelgrove
F1. City	New York
G1. Additional Details	Fire
H1. State	New York
	Filed by Scott Macdonald

A web application displaying parsed JSON values returned by a Sling Servlet

A custom Sling Servlet is an OSGi bundle. However, a difference between an OSGi bundle that contains a service and an OSGi bundle that contains a Sling Servlet is the former requires that you create an instance of the service. For example, assume an OSGi bundle contains a service based on a Java class named `com.adobe.cq.CustomerService`. To get data from the client web page to this OSGi service, you have to create an instance of `com.adobe.cq.CustomerService`, as shown in this example.

```
com.adobe.cq.CustomerService cs = sling.getService(com.adobe.cq.CustomerService.(
```

Then you invoke a service method, as shown in this example that invokes the `injestCustData` method.

```
cs.injestCustData(first, last, phone, desc) ;
```

Note: For information about how to create an Adobe CQ application that builds an OSGi bundle that contains a service (not a Sling Servlet), see Querying Adobe Experience Manager Data using the JCR API.

Note: For information about creating a similiar CQ application that uses a Sling Servlet; however, does not persist the submitted data, see Submitting Adobe CQ form data to Java Sling Servlets.

Note: For information about injecting a DataSourcePool into an OSGi bundle that is not based on a Sling Servlet, but rather a custom Java interface and class, see Injecting a DataSourcePool Service into an Adobe Experience Manager OSGi bundle.

In contrast, when working with an OSGi bundle that contains a Sling Servlet, you post data to the Sling Servlet's `doPost` method. That is, you can use a JQuery AJAX request to post data to the Sling Servlet, as shown in the following example.

```
//Use JQuery AJAX request to post data to a Sling Servlet
 $.ajax({
         type: 'POST',
         url:'/bin/mySearchServlet',
         data:'id='+ claimId+'&firstName='+ myFirst+'&lastName='+ myLast+'&addre
         success: function(msg){
           alert(msg); //display the data returned by the servlet
         }
     });
```

To create an Adobe CQ application that injects a DataSourcePool into a Sling Servlet, 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.

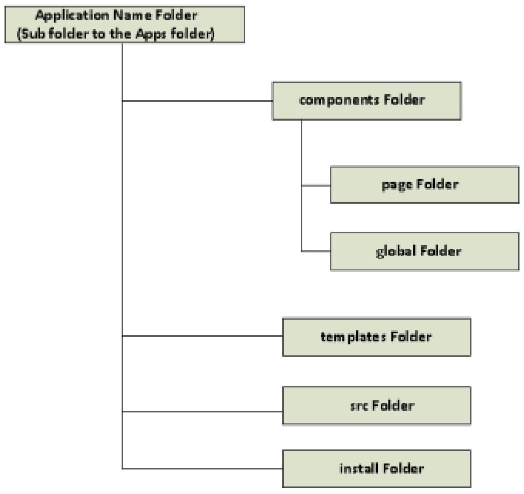
4. Configure the DataSourcePool connection properties.
5. Setup Maven in your development environment.
6. Create an Adobe CQ archetype project.
7. Add Java files that represent the Sling Servet to the Maven project.
8. Modify the Maven POM file.
9. Build the OSGi bundle using Maven.
10. Deploy the bundle to Adobe CQ.
11. Add CSS and JQuery files to a `cq:ClientLibraryFolder` node.
12. Modify the render component to post form data to the Sling Servlet.
13. Create a site that contains a page that lets a user enter and submit customer data.

Note: *Before following along with this development article, setup MySQL and create a database schema named CQ that contains the Customer table. See www.mysql.com.*

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 *slingSevletApp*.
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 *slingTemplate.*
- **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 *slingSevletApp/components/page/slingTemplate.*
- **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/slingSevletApp/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 `slingTemplate`.
- **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/slingServletApp/components/page/slingTemplateJCR/slingTemplateJCR.jsp.*
8. Enter the following JSP code.

```
<html>
<head>
<title>Hello World !!!</title>
</head>
<body>
<h1>Hello Sling Servlet!!!</h1>
<h2>This page will post data to an Adobe CQ Sling Servlet</h2>
</body>
</html>
```

Configure the DataSourcePool connection properties

To the top

Add a configuration for the JDBC Connections Pool service that uses the JDBC driver to create data source objects. The OSGi bundle created in this development article uses this service to connect to the MySQL database. For information, see http://dev.day.com/docs/en/cq/current/developing/jdbc.html.

To configure a DataSourcePool, peform these 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. Click the Configuration tab..

3. Click the + icon that appearts in the JDBC Configuration Pool row.

4. Enter the following values:

- **JBDC Driver class** - the driver class to use to connect to the database. To connect to MySQL, enter com.mysql.jdbc.Driver.
- **JDBC connection URI** - the URI to the database. In this example, enter *jdbc:mysql://localhost:3306/cq*.
- **Username** - the user name to use to connect to MySQL.
- **Password** - the corresponding password.
- **Datasource name** - the datasource name. This is the value that you reference in the Java logic located in the OSGi bunlde. In this example, enter *Customer*.

5. Click Save.

To the top

Setup Maven in your development environment

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: It is recommended that you use Maven 3.0.3 or greater. 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 wi
   | Authentication profiles can be used whenever maven must make a connection
   |-->
  <servers>
    <!-- server
     | Specifies the authentication information to use when connecting to a pa
     | a unique name within the system (referred to by the 'id' attribute belo
     |
     | NOTE: You should either specify username/password OR privateKey/passphr
     |       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 ce
   | However, this repository may have problems with heavy traffic at times, s
   | it to several places.
   |
   | That repository definition will have a unique id, so we can create a mirr
   | repository, to be used as an alternate download site. The mirror site wil
   | 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 
     | for inheritance and direct lookup purposes, and must be unique across t
     |
    <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, a
   | the build process. Profiles provided in the settings.xml are intended to 
   | specific paths and repository locations which allow the build to work in 
   |
   | For example, if you have an integration testing plugin - like cactus - th
   | your Tomcat instance is installed, you can provide a variable here such t
   | 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. Anothe
   | relies on the detection of a system property, either matching a particula
   | or merely testing its existence. Profiles can also be activated by JDK ve
   | value of '1.4' might activate a profile when the build is executed on a J
   | Finally, the list of active profiles can be specified directly from the c
   |
   | NOTE: For profiles defined in the settings.xml, you are restricted to spe
   |       repositories, plugin repositories, and free-form properties to be u
   |       variables for plugins in the POM.
   |
   |-->
  <profiles>
    <!-- profile
```

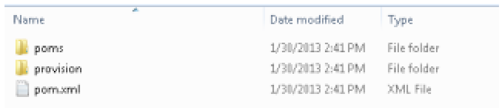
```
      | Specifies a set of introductions to the build process, to be activated u
      | mechanisms described above. For inheritance purposes, and to activate pr
      | or the command line, profiles have to have an ID that is unique.
      |
      | An encouraged best practice for profile identification is to use a cons
      | for profiles, such as 'env-dev', 'env-test', 'env-production', 'user-jd
      | 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/</u

                    <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.



Name	Date modified	Type
poms	1/30/2013 2:41 PM	File folder
provision	1/30/2013 2:41 PM	File folder
pom.xml	1/30/2013 2:41 PM	XML File

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.ds
- **artifactId**: claimds
- **version**: 1.0-SNAPSHOT
- **package**: com.adobe.cq.sling.ds
- **appsFolderName**: adobe-training
- **artifactName**: Claim DS Training Package Bundle
- **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\claimds. Run the following Maven command:

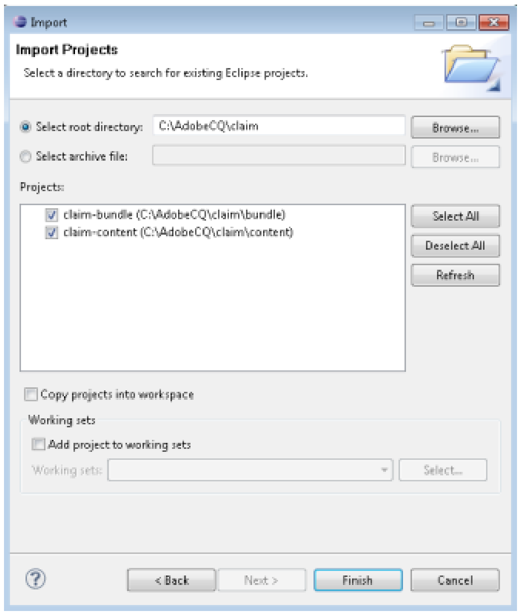
```
mvn eclipse:eclipse
```

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.



Select root directory: C:\AdobeCQ\claim
Select archive file:
Projects:
☑ claim-bundle (C:\AdobeCQ\claim\bundle)
☑ claim-content (C:\AdobeCQ\claim\content)
☐ Copy projects into workspace
Working sets
☐ Add project to working sets
Working sets:

The Eclipse Import Project dialog

The next step is to add a Java file to the `com.adobe.cq.sling.ds` package named `HandleClaim`. 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. For information about this class, see Class SlingAllMethodsServlet.

The `HandleClaim` class uses the following Apache Felix SCR annotations to create the OSGi component: `@Reference`. This annotation injects a `DataSourcePool` into the Sling Servlet. For information about Apache Felix SCR annotations, see http://felix.apache.org/documentation/subprojects/apache-felix-maven-scr-plugin/scr-annotations.html.

In this development article, a `DataSourcePool` instance is injected into the `getConnection` method. This method uses a `DataSourcePool` to return a `Connection` instance to the database. To inject a `DataSourcePool` instance, you use the `@Reference` annotation to define a class member, as shown in the following example.

Notice that `DataSourcePool` instance's `getDataSource` method is called and that the datasource named `Customer` is referenced. This is the name of the `DataSourcePool` that was configured earlier in this development article.

```
@Reference
   private DataSourcePool source;


 //Returns a connection using the configured DataSourcePool
 private Connection getConnection()
 {
 DataSource dataSource = null;
Connection con = null;
try
{
    //Inject the DataSourcePool right here!
   dataSource = (DataSource) source.getDataSource("Customer");
   con = dataSource.getConnection();
    return con;

      }
catch (Exception e)
{
     e.printStackTrace();
     }
return null;
  }
```

The Sling Servlet encodes submitted data into JSON formatted data by using an `org.json.simple.JSONObject` instance, as shown in the following code example.

```
//Encode the submitted form data to JSON
JSONObject obj=new JSONObject();
obj.put("id","id");
obj.put("firstname",firstName);
obj.put("lastname",lastName);
obj.put("address",address);
obj.put("cat",cat);
obj.put("state",state);
obj.put("details",details);
obj.put("date",date);
obj.put("city",city);

//Get the JSON formatted data
String jsonData = obj.toJSONString();
```

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

```
package com.adobe.cq.sling.ds;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.rmi.ServerException;
import java.util.Dictionary;

import org.apache.felix.scr.annotations.Properties;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.sling.SlingServlet;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.SlingHttpServletResponse;
import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
import org.apache.sling.commons.osgi.OsgiUtil;
import org.apache.sling.jcr.api.SlingRepository;
import org.apache.felix.scr.annotations.Reference;
import org.osgi.service.component.ComponentContext;
import javax.jcr.Session;
import javax.jcr.Node;
import org.json.simple.JSONObject;
import java.util.UUID;

//import MySQL APIs
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;
import javax.sql.DataSource;

//Import CQ DataSOurcePool
import com.day.commons.datasource.poolservice.DataSourcePool;

@SlingServlet(paths="/bin/mySearchServlet", methods = "POST", metatype=true)
public class HandleClaim extends org.apache.sling.api.servlets.SlingAllMethodsS
     private static final long serialVersionUID = 2598426539166789515L;

     @Reference
     private DataSourcePool source;


     @Override
     protected void doPost(SlingHttpServletRequest request, SlingHttpServletRes

      try
      {
         //Get the submitted form data that is sent from the
         //CQ web page
          String id = UUID.randomUUID().toString();
          String firstName = request.getParameter("firstName");
          String lastName = request.getParameter("lastName");
```

```
            String address = request.getParameter("address");
            String cat = request.getParameter("cat");
            String state = request.getParameter("state");
            String details = request.getParameter("details");
            String date = request.getParameter("date");
            String city = request.getParameter("city");


            //Persist the Data into MySQL by using connection build with the Data
            injestCustData(firstName, lastName, address, details);

            //Encode the submitted form data to JSON
            JSONObject obj=new JSONObject();
            obj.put("id",id);
            obj.put("firstname",firstName);
            obj.put("lastname",lastName);
            obj.put("address",address);
            obj.put("cat",cat);
            obj.put("state",state);
            obj.put("details",details);
            obj.put("date",date);
            obj.put("city",city);

            //Get the JSON formatted data
            String jsonData = obj.toJSONString();

               //Return the JSON formatted data
           response.getWriter().write(jsonData);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
      }


    //Returns a connection using the configured DataSourcePool
      private Connection getConnection()
      {
      DataSource dataSource = null;
     Connection con = null;
     try
     {
         //Inject the DataSourcePool right here!
        dataSource = (DataSource) source.getDataSource("Customer");
        con = dataSource.getConnection();
         return con;

           }
     catch (Exception e)
     {
         e.printStackTrace();
         }
     return null;
       }

      //Adds a new customer record in the Customer table
      public int injestCustData(String firstName, String lastName, String phone,
         Connection c = null;

         int rowCount= 0;
         try {

               // Create a Connection object
               c =  getConnection();

                ResultSet rs = null;
                Statement s = c.createStatement();
                Statement scount = c.createStatement();

                //Use prepared statements to protected against SQL injection att
                PreparedStatement pstmt = null;
                PreparedStatement ps = null;

                //Set the query and use a preparedStatement
                String query = "Select * FROM Customer";
                pstmt = c.prepareStatement(query);
                rs = pstmt.executeQuery();

                while (rs.next())
                        rowCount++;

                //Set the PK value
                int pkVal = rowCount + 2;

                String insert = "INSERT INTO Customer(custId,custFirst,custLast
                ps = c.prepareStatement(insert);
                ps.setInt(1, pkVal);
                ps.setString(2, firstName);
                ps.setString(3, lastName);
                ps.setString(4, phone);
                ps.setString(5, desc);
                ps.execute();
                return pkVal;
         }
         catch (Exception e) {
           e.printStackTrace();
          }
         finally {
           try
           {
             c.close();
           }

             catch (SQLException e) {
               e.printStackTrace();
             }
     }
         return 0;
  }

 }
```

The Java class uses a `SlingServlet` annotation:

```
@SlingServlet(paths="/bin/mySearchServlet", 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/mySearchServlet',
data:'id='+ claimId+'&firstName='+ myFirst+'&lastName='+
myLast+'&address='+ address+'&cat='+ cat+'&state='+ state+'&details='+
details+'&date='+ date+'&city='+ city,
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. Each form field is retrieved in the `doPost` method by using the `request.getParameter` method.

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

Add the MySQL driver file and org.json.simple.JSONObject data type to Adobe CQ

You have to deploy a bundle fragment to Adobe CQ that contains the database driver file and the `org.json.simple.JSONObject` class to Adobe CQ. The reason is because the `doPost` method in the Sling Servlet uses the `JSONObject` class to encode form data to JSON formatted data. If you do not add this class to Adobe CQ, then you are unable to place the OSGi bundle that contains the Sling Servlet into an Active state. Likewise, without the MySQL driver file, you cannot persist data into MySQL using the DataSourcePool that you configured.

To add the `org.json.simple.JSONObject` class to Adobe CQ, add it to a bundle fragment and then deploy the bundle fragment to Adobe CQ, as discussed in this section. First, download the json-simple JAR from the following URL:

https://code.google.com/p/json-simple/

To create an OSGi bundle fragment that contains the MySQL Driver file and the `org.json.simple.JSONObject` class, perform these tasks:

1. Start Eclipse (Indigo). The steps below have been tested on Eclipse Java EE IDE for Web Developers version Indigo Service Release 1.

2. Select File, New, Other.

3. Under the Plug-in Development folder, choose Plug-in from Existing JAR Archives. Name your

project *jsonBundle*.

4. In the JAR selection dialog, click the Add external button, and browse to the *json-simple* JAR file that you downloaded and the database driver file.

5. Click Next.

6. In the Plug-in Project properties dialog, ensure that you check the checkbox for Analyze library contents and add dependencies.

7. Make sure that the Target Platform is the standard OSGi framework.

8. Ensure the checkboxes for Unzip the JAR archives into the project and Update references to the JAR files are both checked.

9. Click Next, and then Finish.

10. Click the Runtime tab.

11. Make sure that the Exported Packages list is populated.

12. Make sure these packages have been added under the Export-Package header in MANIFEST.MF. Remove the version information in the MANIFEST.MF file. Version numbers can cause conflicts when you upload the OSGi bundle to Adobe CQ.

13. Also make sure that the Import-Package header in MANIFEST.MF is also populated, as shown here (notice that `Export-Package` is the MySQL packages and `org.json.simple`).

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: JsonObject
Bundle-SymbolicName: jsonObject
Bundle-Version: 1.0.0
Export-Package: com.mysql.jdbc,
com.mysql.jdbc.authentication,
com.mysql.jdbc.exceptions,
com.mysql.jdbc.exceptions.jdbc4,
com.mysql.jdbc.integration.c3p0,
com.mysql.jdbc.integration.jboss,
com.mysql.jdbc.interceptors,
com.mysql.jdbc.jdbc2.optional,
com.mysql.jdbc.jmx,
com.mysql.jdbc.log,
com.mysql.jdbc.profiler,
com.mysql.jdbc.util,
org.gjt.mm.mysql,
org.json.simple,
org.json.simple.parser
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

14. Save the project.

15. Build the OSGi bundle by right-clicking the project in the left pane, choose Export, Plug-in Development, Deployable plug-ins and fragments, and click Next.

16. Select a location for the export (C:\TEMP) and click Finish. (Ignore any error messages).

17. In C:\TEMP\plugins, you should now find the OSGi bundle.

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

19. Sort the bundle list by Id and note the Id of the last bundle.

20. Click the Install/Update button.

21. Check the Start Bundle checkbox.

22. Browse to the bundle JAR file you just built. (C:\TEMP\plugins).

23. Click Install.

24. Click the Refresh Packages button.

25. Check the bundle with the highest Id.

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

27. If the status is not Active, check the CQ error.log for exceptions. If you get "org.osgi.framework.BundleException: Unresolved constraint" errors, check the MANIFEST.MF for strict version requirements which might follow: javax.xml.namespace; version="3.1.0"

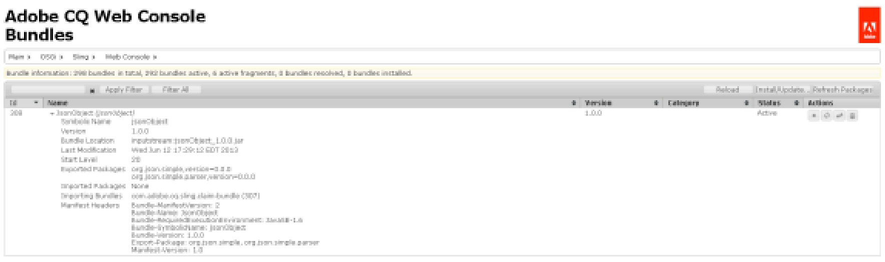
28. If the version requirement causes problems, remove it so that the entry looks like this: javax.xml.namespace.

29. If the entry is not required, remove it entirely.

30. Rebuild the bundle.

31. Delete the previous bundle and deploy the new one.

You will see the OSGi bundle fragment in an Active state, as shown in the following illustration.


Adobe CQ Web Console
Bundles

Main » OSGi » Sling » Web Console »

Bundle information: 298 bundles in total, 292 bundles active, 6 active fragments, 0 bundles resolved, 0 bundles installed.

Id	Name	Version	Category	Status	Actions
308	JsonObject (jsonObject)	1.0.0		Active	

Symbolic Name	jsonObject
Version	1.0.0
Bundle Location	inputstream:jsonObject_1.0.0.jar
Last Modification	Wed Jun 12 17:29:12 EDT 2013
Start Level	20
Exported Packages	org.json.simple,version=0.0.0 org.json.simple.parser,version=0.0.0
Imported Packages	None
Importing Bundles	com.adobe.cq.sling.claim-bundle (307)
Manifest Headers	Bundle-ManifestVersion: 2 Bundle-Name: JsonObject Bundle-RequiredExecutionEnvironment: JavaSE-1.6 Bundle-SymbolicName: jsonObject Bundle-Version: 1.0.0 Export-Package: org.json.simple, org.json.simple.parser Manifest-Version: 1.0

The OSGI bundle fragment that contains the org.json.simple package in an Active state

Modify the Maven POM file

To the top

Modify the POM files to successfully build the OSGi bundle. In the POM file located at C:\AdobeCQ\claim\bundle, add the following dependencies.

- org.apache.felix.scr
- org.apache.felix.scr.annotations
- org.apache.jackrabbit
- org.apache.sling
- com.googlecode.json-simple

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.or
    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.sling.ds</groupId>
        <artifactId>claimds</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>claimds-bundle</artifactId>
    <packaging>bundle</packaging>
    <name>Claim DS Training 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.sling.ds.claimds-bun
                    </instructions>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.sling</groupId>
                <artifactId>maven-sling-plugin</artifactId>
                <configuration>
                    <slingUrl>http://${crx.host}:${crx.port}/apps/adobe-traini
                    <usePut>true</usePut>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <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>junit</groupId>
            <artifactId>junit</artifactId>
        </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.commons.osgi</artifactId>
        <version>2.2.0</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>org.apache.sling</groupId>
        <artifactId>org.apache.sling.jcr.api</artifactId>
        <version>2.0.4</version>
      </dependency>

       <dependency>
        <groupId>org.apache.sling</groupId>
        <artifactId>org.apache.sling.api</artifactId>
        <version>2.0.2-incubator</version>
      </dependency>

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

```

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

    <dependency>
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
            <version>1.1</version>
        </dependency>

        <dependency>
            <groupId>com.day.commons</groupId>
            <artifactId>day.commons.datasource.poolservice</artifactId>
            <version>1.0.10</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>
```

To the top

Build the OSGi bundle using Maven

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.sling.ds.HandleClaim` 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.sling.ds.SimpleDSComponent'
        <implementation class="com.adobe.cq.sling.ds.SimpleDSComponent"/>
        <service servicefactory="false">
            <provide interface="java.lang.Runnable"/>
        </service>
        <property name="service.pid" value="com.adobe.cq.sling.ds.SimpleDSCompo
    </scr:component>
    <scr:component enabled="true" name="com.adobe.cq.sling.ds.HandleClaim">
        <implementation class="com.adobe.cq.sling.ds.HandleClaim"/>
        <service servicefactory="false">
            <provide interface="javax.servlet.Servlet"/>
        </service>
        <property name="sling.servlet.paths" value="/bin/mySearchServlet"/>
        <property name="sling.servlet.methods" value="POST"/>
        <property name="service.pid" value="com.adobe.cq.sling.ds.HandleClaim"/:
        <reference name="source" interface="com.day.commons.datasource.poolserv:
    </scr:component>
</components>
```

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

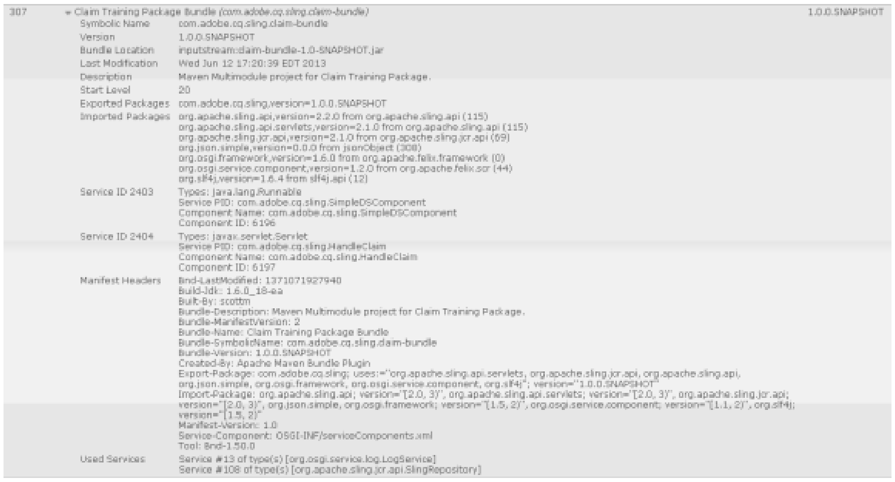
To build the OSGi component by using Maven, perform these steps:

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

Deploy the bundle to Adobe CQ

To the top

Once you deploy the OSGi bundle, you can post form data 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.



```
307       ▾ Claim Training Package Bundle (com.adobe.cq.sling.claim-bundle)                                  1.0.0.SNAPSHOT
            Symbolic Name       com.adobe.cq.sling.claim-bundle
            Version             1.0.0.SNAPSHOT
            Bundle Location     inputstream:claim-bundle-1.0-SNAPSHOT.jar
            Last Modification   Wed Jun 12 17:20:39 EDT 2013
            Description         Maven Multimodule project for Claim Training Package.
            Start Level         20
            Exported Packages   com.adobe.cq.sling,version=1.0.0.SNAPSHOT
            Imported Packages   org.apache.sling.api,version=2.2.0 from org.apache.sling.api (115)
                                org.apache.sling.api.servlets,version=2.1.0 from org.apache.sling.api (115)
                                org.apache.sling.jcr.api,version=2.1.0 from org.apache.sling.jcr.api (69)
                                org.json.simple,version=0.0.0 from jsonObject (308)
                                org.osgi.framework,version=1.6.0 from org.apache.felix.framework (0)
                                org.osgi.service.component,version=1.2.0 from org.apache.felix.scr (44)
                                org.slf4j,version=1.6.4 from slf4j.api (12)
            Service ID 2403     Types: java.lang.Runnable
                                Service PID: com.adobe.cq.sling.SimpleDSComponent
                                Component Name: com.adobe.cq.sling.SimpleDSComponent
                                Component ID: 6196
            Service ID 2404     Types: javax.servlet.Servlet
                                Service PID: com.adobe.cq.sling.HandleClaim
                                Component Name: com.adobe.cq.sling.HandleClaim
                                Component ID: 6197
            Manifest Headers    Bnd-LastModified: 1371071927940
                                Build-Jdk: 1.6.0_18-ea
                                Built-By: scottm
                                Bundle-Description: Maven Multimodule project for Claim Training Package.
                                Bundle-ManifestVersion: 2
                                Bundle-Name: Claim Training Package Bundle
                                Bundle-SymbolicName: com.adobe.cq.sling.claim-bundle
                                Bundle-Version: 1.0.0.SNAPSHOT
                                Created-By: Apache Maven Bundle Plugin
                                Export-Package: com.adobe.cq.sling; uses:="org.apache.sling.api.servlets, org.apache.sling.jcr.api, org.apache.sling.api,
                                org.json.simple, org.osgi.framework, org.osgi.service.component, org.slf4j"; version="1.0.0.SNAPSHOT"
                                Import-Package: org.apache.sling.api; version="[2.0, 3)", org.apache.sling.api.servlets; version="[2.0, 3)", org.apache.sling.jcr.api;
                                version="[2.0, 3)", org.json.simple, org.osgi.framework; version="[1.5, 2)", org.osgi.service.component; version="[1.1, 2)", org.slf4j;
                                version="[1.5, 2)"
                                Manifest-Version: 1.0
                                Service-Component: OSGI-INF/serviceComponents.xml
                                Tool: Bnd-1.50.0
            Used Services       Service #13 of type(s) [org.osgi.service.log.LogService]
                                Service #108 of type(s) [org.apache.sling.jcr.api.SlingRepository]
```

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\claimds\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 CSS and JQuery files to a CQ:ClientLibraryFolder node

To the top

You add a CSS file and a JQuery framework file to a `cq:ClientLibraryFolder` node to define the style of the client JSP. The JQuery framework file that is added is named jquery-1.6.3.min.js.

To add CSS files and 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 CSS files and the JQuery library files.

To add the JQuery framework, add a new node named clientlibs to your component (as discussed later). Add these two properties to this node.

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

The dependencies property informs CQ to include the CSS and JQuery libraries in the page. The categories property informs CQ which clientlibs must be included.

After you create the Clientlibs folder, add a CSS file, and the JQuery library file, and two map text files.

Site CSS file

The site.css file defines the display style for the client JSP file that lets the user enter and submit data. The following code represents the site.css file.

```
/* reset */
html, body, div, span, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td {
 margin: 0;
 padding: 0;
 border: 0;
 font-size: 100%;
 font: inherit;
 vertical-align: baseline;
}
html , body{
 line-height: 1;
 background-color: #334873;
 background-image: url(../_images/bg-page2.png);
}

ol, ul {
 list-style: none;
}


table {
 border-collapse: collapse;
 border-spacing: 0;
}
/* end reset*/



h1, h2, h3 {
 font-family: 'ColaborateRegular', Arial, sans-serif;
}


strong {
 font-family: 'ColaborateMediumRegular', Arial, sans-serif;
}

em {
 font-family: 'ColaborateThinRegular', Arial, sans-serif;
}

.content {
 max-width: 760px;
 margin: 20px 0 0 100px;
}

.clear:after {
content: "."; display: block; height: 0; clear: both; visibility: hidden;
}

.clear {
 min-height: 1px;
}

* html .clear {
 height: 1px;
}

.header {
 position: relative;
 border-top: solid 6px white;
 padding: 10px 0 10px 0;
 margin-bottom: 20px;
}


.main {
 xxposition: relative;
 padding-bottom: 1em;
 border-bottom: solid 1px rgba(255,255,255,.5);
 xxoverflow:hidden;
 xxmin-height: 300px;
}
```

```

.main h1 {
 font-size: 32px;
 color: white;
 text-shadow: 1px 1px 1px rgba(0,0,0,.75);
 border-bottom: solid 1px rgba(255,255,255,.5);
 margin-bottom: 0.75em;
}


p , li, legend , form{
 font-size: 18px;
 color: white;
 font-family: 'ColaborateLightRegular', Arial, sans-serif;
 line-height: 125%;
 margin-bottom: 10px;
}

fieldset {
 padding: 10px;
 border: 1px solid white;
 margin: 25px 0;
}

.nav {
 margin: 10px 0 0 100px;
}

.nav li {
 display: inline-block;
}

.nav a:hover, .example:hover{
 background-color: rgba(255,255,255,.85);
 color: rgb(0,0,0);
}

h3 {
 font-size: 18px;
 color: rgb(227,198,133);;
}

.results h2 {
 color: rgba(255,255,255,1);
}
.results div {
 padding-bottom: 10px;
}
.results div code {
 float: right;
 width: 60%;
}

input {
 font-size: 20px;
}
.form .wide {
 font-size: 18px;
 width: 100%;
}
.resultSection {
 float: right;
 width: 45%;
 margin-left: 20px;
}
#regexTester {
 margin-right: 55%;
}
.sideBySide li {
 float: left;
 overflow: hidden;
 width: 220px;
}
.clickable {
 cursor:pointer;
 margin-bottom: 5px;
}

.clickable:hover {
background-color:#FFC;
}


.col1 {
 float: left;
 width: 75%;
}
.col2 {
 float: right;
 width: 20%;
}

.col2 ul {
 margin-left: 20px;
 list-style: square;
}
.col2 li {
 font-size: 90%;
}


#selectorList {
 overflow: hidden;
}
#selector {
 width: 275px;
}


form#signup .label {
 width: 200px;
}
```

Text files

You have to add two text files to the clientlibs folder. These text files map to the JS file and the CSS file. The names of the text files are: css.txt and js.txt.

The css.txt file contains the CSS file name: site.css. Likewise, the js.txt file contains the JS file name: jquery-1.6.3.min.js.

Add the files to the ClientLibs folder

1. Right-click /apps/slingServletApp/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. On your file system, navigate where you placed the CSS file. Drag and drop the site.css files to the clientlibs folder by using CRXDE.
6. Add a TXT file to the clientlibs folder named js.txt. The content of the js.txt file is the JQuery JS file name.
7. Add a TXT file to the clientlibs node named css.txt. The content of the css.txt file is the CSS file name.

To the top

Modify the slingTemplate JSP to post data to the Sling Servlet

Modify the slingTemplate.jsp file to post data to the Sling Servlet that was created in this development article. In this example, a JQuery Ajax Post request is used and the form data is passed to the Sling Servlet's `doPost` method (the method defined in the `HandleClaim` Java class). The following code represents the AJAX request.

```
//Use JQuery AJAX request to post data to a Sling Servlet
$.ajax({
type: 'POST',
url:'/bin/mySearchServlet',
data:'id='+ claimId+'&firstName='+ myFirst+'&lastName='+
myLast+'&address='+ address+'&cat='+ cat+'&state='+ state+'&details='+
details+'&date='+ date+'&city='+ city,
success: function(msg){

  var json = jQuery.parseJSON(msg);
var msgId= json.id;
var lastName = json.lastname;
var firstName = json.firstname;
$('#ClaimNum').val(msgId);
$('#json').val("Filed by " + firstName + " " + lastName);
  }
  });
});
```

Notice that the url specifies the value of the `path` attribute in the `SlingServlet` annotation defined in the `HandleClaim` method. The JSON formatted data that is returned by the Sling Servlet is written to the Text Area component named `json`.

The following JavaScript code represents the slingTemplate JSP file.

```
<%@include file="/libs/foundation/global.jsp"%>
<cq:includeClientLib categories="jquerysamples" />
<html>
<head>
<meta charset="UTF-8">
<title>Adobe CQ Sling Servlet Page</title>
<style>
#signup .indent label.error {
  margin-left: 0;
}
#signup label.error {
  font-size: 0.8em;
  color: #F00;
  font-weight: bold;
  display: block;
  margin-left: 215px;
}
#signup  input.error, #signup select.error  {
  background: #FFA9B8;
  border: 1px solid red;
}
</style>
<script>
//Creates a GUID value using JavaScript - used for the unique value for the ger
 function createUUID() {

    var s = [];
    var hexDigits = "0123456789abcdef";
    for (var i = 0; i < 36; i++) {
        s[i] = hexDigits.substr(Math.floor(Math.random() * 0x10), 1);
    }
    s[14] = "4";  // bits 12-15 of the time_hi_and_version field to 0010
    s[19] = hexDigits.substr((s[19] & 0x3) | 0x8, 1);  // bits 6-7 of the clocl
    s[8] = s[13] = s[18] = s[23] = "-";

    var uuid = s.join("");
    return uuid;
}

$(document).ready(function() {

    $('body').hide().fadeIn(5000);

$('#submit').click(function() {
    var failure = function(err) {
             alert("Unable to retrive data "+err);
   };

    //Get the user-defined values that represent claim data to persist in the /
    var myFirst= $('#FirstName').val() ;
    var myLast= $('#LastName').val() ;
    var date= $('#DateId').val() ;
    var cat= $('#Cat_Id').val() ;
    var state= $('#State_Id').val() ;
    var details= $('#Explain').val() ;
    var city= $('#City').val() ;
    var address= $('#Address').val() ;
    var claimId = createUUID();


    //Use JQuery AJAX request to post data to a Sling Servlet
    $.ajax({
         type: 'POST',
         url:'/bin/mySearchServlet',
         data:'id='+ claimId+'&firstName='+ myFirst+'&lastName='+ myLast+'&addı
         success: function(msg){

           var json = jQuery.parseJSON(msg);
            var msgId=   json.id;
            var lastName = json.lastname;
            var firstName = json.firstname;

            $('#ClaimNum').val(msgId);
            $('#json').val("Filed by " + firstName + " " + lastName);
        }
    });
  });

}); // end ready
</script>
</head>

<title>Adobe CQ Sling Mobile Page</title>

<body>


<h1>Adobe CQ Mobile Claim Sling Form</h1>

</div>

<form method="#">

 <table border="1" align="left">

 <tr>
 <td>
<label for="ClaimNum" id="ClaimNumLabel" >A. Claim Number</label>
 </td>
 <td>
 <input id="ClaimNum" name="A1. Claim Number" readonly=true type="text" value='
 </td>
 </tr>
 <tr>
 <td>
<label for="DateId" id="DateIncident">A.2. Date of Incident</label>
 </td>
 <td>
 <input id="DateId" name="A.2 Date of Incident"  type="text" value="">
 </td>
 </tr>

  <tr>
 <td>
<label for="FirstName" id="FirstNameLabel" >B2. First Name</label>
 </td>
 <td>
<input id="FirstName" name="B1. First Name    " type="text" value="">
 </td>
 </tr>

 <tr>
 <td>
<label for="LastName" id="LastNameLabel" name="LastNameeLabel">C1. Last Name
 </td>
 <td>
<input id="LastName" name="C1. Last Name     " type="text" value="">
 </td>
 </tr>

 <tr>
 <td>
<label for="Cat_Id">D1. Category </label>
 </td>
 <td>
<select id="Cat_Id" name="Category ">
              <option value="Home">Home Claim</option>
              <option value="Auto">Auto Claim</option>
              <option value="Boat">Boat Claim</option>
              <option value="Personal">Personnal Claim</option>
            </select>
 </td>
 </tr>

 <tr>
 <td>
<label for="Address" id="AddressLabel" name="AddressLabel">E1. Address   </labe
 </td>
 <td>
<input id="Address" name="Address   " type="text" value="">
 </td>
 </tr>

 <tr>
 <td>
<label for="City" id="CityLabel" name="CityLabel">F1. City   </label>
 </td>
 <td>
<input id="City" name="City   " type="text" value="">
 </td>
 </tr>

 <tr>
 <td>
<label for="Explain" id="ExplainLabel" name="ExplainLabel">G1. Additional Deta
 </td>
 <td>
 <input id="Explain" name="Explain   " type="text" value="">
 </td>
 </tr>

 <tr>
 <td>
<label for="State_Id">H1. State </label>
 </td>
 <td>
 <select id="State_Id" name="State ">
              <option value="Alabama">Alabama</option>
              <option value="Alaska">Alaska</option>
              <option value="Arizona">Arizona</option>
              <option value="Arkansas">Arkansas</option>
              <option value="California">California</option>
              <option value="Colorado">Colorado</option>
              <option value="Connecticut">Connecticut</option>
              <option value="Delaware">Delaware</option>
              <option value="District of Columbia">District of Columbia</optio
              <option value="Florida">Florida</option>
              <option value="Georgia">Georgia</option>
              <option value="Hawaii">Hawaii</option>
              <option value="Idaho">Idaho</option>
              <option value="Illinois">Illinois</option>
              <option value="Indiana">Indiana</option>
              <option value="Iowa">Iowa</option>
              <option value="Kansas">Kansas</option>
              <option value="Kentucky">Kentucky</option>
              <option value="Louisiana">Louisiana</option>
              <option value="Maine">Maine</option>
              <option value="Maryland">Maryland</option>
              <option value="Massachusetts">Massachusetts</option>
              <option value="Michigan">Michigan</option>
              <option value="Minnesota">Minnesota</option>
              <option value="Mississippi">Mississippi</option>
              <option value="Missouri">Missouri</option>
              <option value="Montana">Montana</option>
              <option value="Nebraska">Nebraska</option>
              <option value="Nevada">Nevada</option>
              <option value="New Hampshire">New Hampshire</option>
              <option value="New Jersey">New Jersey</option>
              <option value="New Mexico">New Mexico</option>
              <option value="New York">New York</option>
              <option value="North Carolina">North Carolina</option>
              <option value="North Dakota">North Dakota</option>
              <option value="Ohio">Ohio</option>
              <option value="Oklahoma">Oklahoma</option>
              <option value="Oregon">Oregon</option>
              <option value="Pennsylvania">Pennsylvania</option>
              <option value="Rhode Island">Rhode Island</option>
              <option value="South Carolina">South Carolina</option>
              <option value="South Dakota">South Dakota</option>
              <option value="Tennessee">Tennessee</option>
              <option value="Texas">Texas</option>
              <option value="Utah">Utah</option>
              <option value="Vermont">Vermont</option>
              <option value="Virginia">Virginia</option>
              <option value="Washington">Washington</option>
              <option value="West Virginia">West Virginia</option>
              <option value="Wisconsin">Wisconsin</option>
              <option value="Wyoming">Wyoming</option>
            </select>
 </td>
 </tr>

 <tr>
 <td></td>

  <td>
<textarea id="json" rows="4" cols="50">
</textarea>
```

```
 </td>

 </tr>

 <tr>
 <td></td>
 <td>
<input type="button" value="Submit"  name="submit" id="submit" value="Submit">

 </td>

 </tr>

 </table>

</form>




</body>

</html>
```

Modify the slingTemplate 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/slingServletApp/components/page/slingTemplateJCR/slingTemplateJCR.jsp.
4. Replace the JSP code with the new code shown in this section.
5. Click Save All.

Create a CQ web page that displays the client web page

To the top

The final task is to create a site that contains a page that is based on the slingTemplate (the template created earlier in this development article). When the user enters data and submits it, the data is posted to the Sling Servlet and then persisted into the database.

Create a CQ web page that displays the JCR client:

1. Go to the CQ welcome page at http://[host name]:[port]; for example, http://localhost:4502. Select Websites.
2. From the left hand pane, select Websites.
3. Select New Page.
4. Specify the title of the page in the Title field.
5. Specify the name of the page in the Name field.
6. Select *slingTemplate* 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.
7. 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 custom sling servlet that uses a DataSourcePool 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.