

## Adobe CQ Help /

# Using Sling APIs to retrieve content from the Adobe Experience Manager Repository

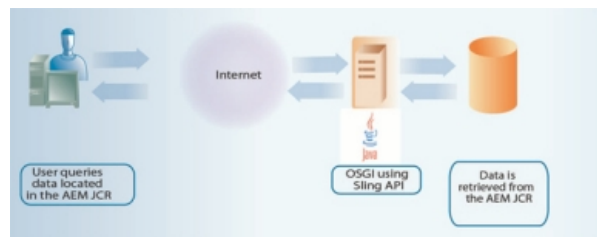
### Article summary

<b>Summary</b>	Discusses how to programmatically retrieve AEM resources from the Java Content Repository (JCR) using Sling APIs. Also discusses how to use the <code>adaptTo</code> method to convert an AEM resource to another type.
<b>Digital Marketing Solution(s)</b>	Adobe Experience Manager (Adobe CQ)
<b>Audience</b>	Developer (intermediate)
<b>Required Skills</b>	Java, Sling, OSGi Maven, HTML
<b>Tested On</b>	Adobe CQ 5.5, Adobe CQ 5.6

### Introduction

Adobe Experience Manager contains a Java Content Repository (JCR) that stores nodes and properties. A node located in the JCR is considered a resource. For example, a web page is a resource in the JCR. You can use the JCR API to retrieve resources from the JCR. For information about using the JCR API, see [Programmatically Accessing Adobe CQ Content using the JCR API](#).

However, you can also retrieve content from the JCR using Sling APIs. In fact, a resource is a central part of Sling and it assumes everything in the JCR is a resource. You can use the Sling API from within an OSGi bundle to retrieve a resource from within the AEM JCR. To use the Sling API from within an OSGi component, you inject an `org.apache.sling.api.resource.ResourceResolverFactory` instance into the service. See [Interface ResourceResolverFactory](#).



The Sling API is used within an OSGi bundle

When using the Sling API to query the AEM JCR, you have access to helper methods that are not available when using the JCR API. For example, the `adaptTo` method converts a resource into an appropriate object representing a certain aspect of this resource. For example to translate a `Resource` object to the corresponding `Node` object, you invoke the `adaptTo` method:

```
Node node = resource.adaptTo(Node.class);
```

This development article guides you through how to build an AEM application that uses the Sling API to retrieve resources from the AEM JCR. An OSGi bundle is created that retrieves a resource, calls the `adaptTo` method and retrieves a value. To create an AEM web application that retrieves resources from the JCR by using the Sling API, perform these tasks:

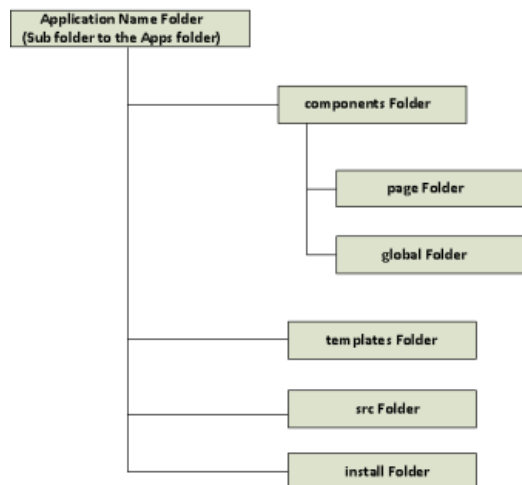
1. Create an Adobe CQ application folder structure.
2. Create a template on which the page component is based.
3. Create a render component that uses the template.
4. Setup Maven in your development environment.
5. Create an Adobe CQ archetype project.
6. Add Java files that use the Sling API to the the Maven project.
7. Modify the Maven POM files.
8. Build the OSGi bundle using Maven.

9. Deploy the bundle to Adobe CQ.
10. Modify the render component to invoke an OSGi operation that retrieves a resource.
11. Create a site that contains a page that lets a user retrieve a resource the Sling API.

## Create an AEM application folder structure

[To the top](#)

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



An AEM 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 AEM 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 (if you are using AEM 5.6, click Tools from the left menu).
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 `slingApp`.
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 CRXDE Lite 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 (if you are using AEM 5.6, click Tools from the left menu).
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 `templateSling`.
- **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 `slingApp/components/page/templateSling`.
- **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.

## Create a render component that uses the template

[To the top](#)

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 (if you are using AEM 5.6, click Tools from the left menu).
3. Right-click `/apps/slingApp/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 `templateSling`.
- **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 `templateSling.jsp` located at:

`/apps/slingApp/components/page/templateSling/templateSling.jsp`.

8. Enter the following JSP code.

```

1  <html>
2  <head>
3  <title>Hello World !!!</title>
4  </head>
5  <body>
6  <h1>Hello Sling API!!!</h1>
7  <h2>This page will retrieve a resource from the AEM JCR using the Sling API</h2>
8  </body>
9  </html>
```

## Setup Maven in your development environment

[To the top](#)

You can use Maven to build an OSGi bundle that uses the QueryBuilder API and is deployed to Adobe CQ. 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:

```

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\.

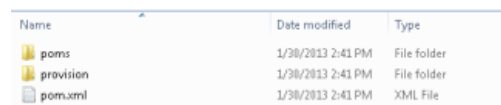
You have to configure your settings.xml file to use Adobe's public repository. For information, see Adobe Public Maven Repository at <http://repo.adobe.com/>.

---

## 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:** custom.sling
- **artifactId:** querysling
- **version:** 1.0-SNAPSHOT
- **package:** custom.sling
- **appsFolderName:** sling-training
- **artifactName:** Sling 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\querysling.

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.

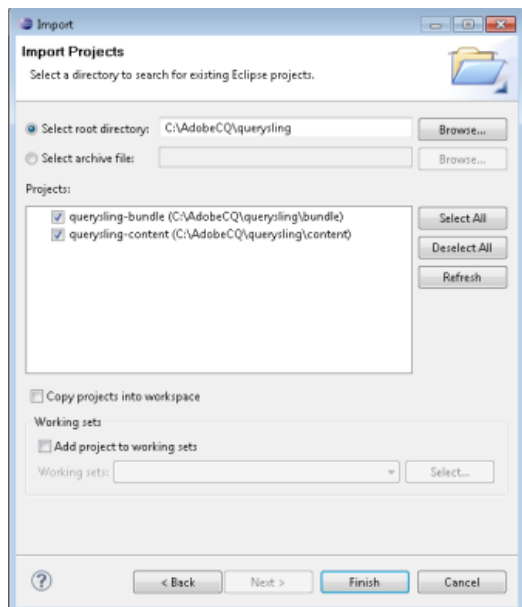
*Note: Ensure that Adobe Maven repository is configured in your POM file. You'll need to do that before this goal will work (and the content-package-maven-plugin can be resolved).*

---

## 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 Java files to the `custom.sling` package. The Java files that you create in this section use the Sling API to retrieve a resource in the AEM JCR. For information, see [Sling API](#).

Add the following Java files to the package named `custom.sling`:

- A Java interface named `Query`.
- A Java class named `QueryImp` that implements the `Query` interface.

### Query interface

The following code represents the `Query` interface that contains a method named `getJCRData`. The implementation logic for this method is located in the `QueryImp` class. The `getJCRData` method uses the Sling API to retrieve a resource located in the AEM JCR.

```

1 package custom.sling;
2
3 public interface Query{
4
5     public String getJCRData(String location) ;
6
7 }
```

### QueryImp class

The `QueryImp` class uses the following Apache Felix SCR annotations to create the OSGi component:

- **@Component** - defines the class as a component
- **@Service** - defines the service interface that is provided by the component
- **@Reference** - injects a service into the component

*Note:* 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 `ResourceResolverFactory` instance is injected into the `getJCRData` method. This instance is required to retrieve a resource from the AEM JCR. To inject a `ResourceResolverFactory` instance, you use the `@Reference` annotation to define a class member, as shown in the following example.

```
//Inject a Sling ResourceResolverFactory
@Reference
private ResourceResolverFactory resolverFactory;
```

Within the `getJCRData` method, invoke the `ResourceResolverFactory` object's `getAdministrativeResourceResolver` method to create a `ResourceResolver` object. You can use a `ResourceResolver` instance to get a resource located in the AEM JCR as shown here.

```
ResourceResolver resourceResolver =
    resolverFactory.getAdministrativeResourceResolver(null);
```

```
Resource res = resourceResolver.getResource(resourcePath);
```

Once you have a `Resource` instance, you can call the `adaptTo` method that converts (adapt) the resource to another type. For example, assume that the resource is a page (`cq:Page`). You can create a `com.day.cq.wcm.api.Page` instance by calling the `adaptTo` method as shown in the following example.

```
//Adapt the resource to another type - in this example to a
com.day.cq.wcm.api.page
Page page = res.adaptTo(Page.class);
String title = page.getTitle();
```

The following Java code represents the `QueryImp` class that uses the Sling API.

```
1  package custom.sling;
2  import org.w3c.dom.Document;
3  import org.w3c.dom.Element;
4
5
6  import org.slf4j.Logger;
7  import org.slf4j.LoggerFactory;
8
9  import java.io.StringWriter;
10 import java.util.Iterator;
11 import java.util.List;
12 import java.util.ArrayList;
13
14 import javax.jcr.Repository;
15 import javax.jcr.SimpleCredentials;
16 import javax.jcr.Node;
17 import javax.xml.parsers.DocumentBuilder;
18 import javax.xml.parsers.DocumentBuilderFactory;
19
20 import org.apache.jackrabbit.commons.JcrUtils;
21
22 import javax.xml.transform.Transformer;
23 import javax.xml.transform.TransformerFactory;
24 import javax.xml.transform.dom.DOMSource;
25 import javax.xml.transform.stream.StreamResult;
26
27 import org.apache.felix.scr.annotations.Component;
28 import org.apache.felix.scr.annotations.Service;
29 import javax.jcr.RepositoryException;
30 import org.apache.felix.scr.annotations.Reference;
31 import org.apache.jackrabbit.commons.JcrUtils;
32
33 import javax.jcr.Session;
34 import javax.jcr.Node;
35
36
37 //Sling Imports
38 import org.apache.sling.api.resource.ResourceResolverFactory ;
39 import org.apache.sling.api.resource.ResourceResolver;
40 import org.apache.sling.api.resource.Resource;
41 import com.day.cq.wcm.api.Page;
42
43 //This is a component so it can provide or consume services
44 @Component
45
46 @Service
47 public class QueryImp implements Query {
48
49 //Inject a Sling ResourceResolverFactory
50 @Reference
51 private ResourceResolverFactory resolverFactory;
52
53 @Override
54 public String getJCRData(String location) {
55 try
56 {
57 //Get the title of the AEM web page at this specific location - assume its a
58 ResourceResolver resourceResolver = resolverFactory.getAdministrativeResourceResolver();
59 Resource res = resourceResolver.getResource(location);
60
61 //Adapts the resource to another type - in this example to a com.day.cq.wcm.api.page
62 Page page = res.adaptTo(Page.class);
63 String title = page.getTitle(); // Get the title of the web page
64 return title ;
65 }
66 catch (Exception e)
67 {
68 e.printStackTrace() ;
69 }
```

```

70 }
71
72 return null;
73 }
74
75 }

```

## 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\querysling\bundle, add the following dependencies.

- org.apache.felix.scr
- org.apache.felix.scr.annotations
- org.apache.jackrabbit
- org.apache.sling
- com.day.cq.wcm.api

Because the Sling API is used, a Maven dependency for that API exists.

```

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

```

Also because the Java class contains a dependency to the `com.day.cq.wcm.api.Page` API, you have to add the following `<repositories>` element to your POM file.

Add the following `<repositories>` element to your POM file.

```

<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>

```

Once you add this repository element to your POM file, you can add the following dependency to your POM file, that lets you use the `com.day.cq.wcm.api.Page` API in your Java code.

```

<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>

```

The following XML represents the POM file to build the OSGi bundle that contains the Sling API.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
3    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
4    <modelVersion>4.0.0</modelVersion>
5    <!-- =====
6    <!-- P A R E N T P R O J E C T D E S C R I P T I O N -->
7    <!-- =====
8    <parent>

```

```

9      <groupId>custom.sling</groupId>
10     <artifactId>querysling</artifactId>
11     <version>1.0-SNAPSHOT</version>
12 </parent>
13
14 <!-- =====
15 <!-- P R O J E C T D E S C R I P T I O N -->
16 <!-- =====
17
18 <artifactId>querysling-bundle</artifactId>
19 <packaging>bundle</packaging>
20 <name>Training Bundle</name>
21
22 <!-- =====
23 <!-- B U I L D D E F I N I T I O N -->
24 <!-- =====
25 <build>
26
27     <plugins>
28         <plugin>
29             <groupId>org.apache.felix</groupId>
30             <artifactId>maven-scr-plugin</artifactId>
31             <executions>
32                 <execution>
33                     <id>generate-scr-descriptor</id>
34                     <goals>
35                         <goal>scr</goal>
36                     </goals>
37                 </execution>
38             </executions>
39         </plugin>
40         <plugin>
41             <groupId>org.apache.felix</groupId>
42             <artifactId>maven-bundle-plugin</artifactId>
43             <extensions>true</extensions>
44             <configuration>
45                 <instructions>
46                     <Bundle-SymbolicName>custom.sling.querysling-bundle</B
47                 </instructions>
48             </configuration>
49         </plugin>
50         <plugin>
51             <groupId>org.apache.sling</groupId>
52             <artifactId>maven-sling-plugin</artifactId>
53             <configuration>
54                 <slingUrl>http://${crx.host}:${crx.port}/apps/query-trainin
55                 <usePut>true</usePut>
56             </configuration>
57         </plugin>
58     </plugins>
59 </build>
60
61 <dependencies>
62     <dependency>
63         <groupId>org.osgi</groupId>
64         <artifactId>org.osgi.compendium</artifactId>
65     </dependency>
66     <dependency>
67         <groupId>org.osgi</groupId>
68         <artifactId>org.osgi.core</artifactId>
69     </dependency>
70     <dependency>
71         <groupId>org.apache.felix</groupId>
72         <artifactId>org.apache.felix.scr.annotations</artifactId>
73     </dependency>
74     <dependency>
75         <groupId>org.slf4j</groupId>
76         <artifactId>slf4j-api</artifactId>
77     </dependency>
78
79
80     <dependency>
81         <groupId>org.apache.felix</groupId>
82
83         <artifactId>org.osgi.core</artifactId>
84
85         <version>1.4.0</version>
86     </dependency>
87
88
89
90     <dependency>
91         <groupId>org.apache.jackrabbit</groupId>
92         <artifactId>jackrabbit-core</artifactId>

```



```

93     <version>2.4.3</version>
94   </dependency>
95
96   <dependency>
97     <groupId>org.apache.jackrabbit</groupId>
98     <artifactId>jackrabbit-jcr-commons</artifactId>
99     <version>2.4.3</version>
100   </dependency>
101   <dependency>
102     <groupId>junit</groupId>
103     <artifactId>junit</artifactId>
104   </dependency>
105
106   <dependency>
107     <groupId>org.apache.sling</groupId>
108     <artifactId>org.apache.sling.api</artifactId>
109     <version>2.2.4</version>
110     <scope>provided</scope>
111   </dependency>
112
113   <dependency>
114     <groupId>javax.jcr</groupId>
115     <artifactId>jcr</artifactId>
116     <version>2.0</version>
117   </dependency>
118
119   <dependency>
120     <groupId>com.day.cq.wcm</groupId>
121     <artifactId>cq-wcm-api</artifactId>
122     <version>5.5.0</version>
123     <scope>provided</scope>
124   </dependency>
125
126   <dependency>
127     <groupId>com.day.cq</groupId>
128     <artifactId>cq-commons</artifactId>
129     <version>5.5.0</version>
130     <scope>provided</scope>
131   </dependency>
132
133 </dependencies>
134
135 <repositories>
136   <repository>
137     <id>adobe</id>
138     <name>Adobe Public Repository</name>
139     <url>http://repo.adobe.com/nexus/content/groups/public</url>
140     <layout>default</layout>
141   </repository>
142 </repositories>
143 <pluginRepositories>
144   <pluginRepository>
145     <id>adobe</id>
146     <name>Adobe Public Repository</name>
147     <url>http://repo.adobe.com/nexus/content/groups/public</url>
148     <layout>default</layout>
149   </pluginRepository>
150 </pluginRepositories>
151
152 </project>

```

## Build the OSGi bundle using Maven

[To the top](#)

Build the OSGi bundle by using Maven. When you build the OSGi bundle, Maven creates the required serviceComponents.xml file based on the annotations that are included in the QueryImp class. The following XML represents this file.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <components xmlns:scr="http://www.osgi.org/xmlns/scr/v1.0.0">
3    <scr:component enabled="true" name="custom.sling.QueryImp">
4      <implementation class="custom.sling.QueryImp"/>
5      <service servicefactory="false">
6        <provide interface="custom.sling.Query"/>
7      </service>
8      <property name="service.pid" value="custom.sling.QueryImp"/>
9      <reference name="resolverFactory" interface="org.apache.sling.api.resou
10    </scr:component>
11    <scr:component enabled="true" name="custom.sling.SimpleDSComponent">
12      <implementation class="custom.sling.SimpleDSComponent"/>
13      <service servicefactory="false">
14        <provide interface="java.lang.Runnable"/>
15      </service>

```

```

16     <property name="service.pid" value="custom.sling.SimpleDSComponent"/>
17   </scr:component>
18 </components>

```

There are a couple of points to note about this XML file. First, notice that the implementation class element specifies `custom.sling.QueryImp`. The service element contains an interface attribute that specifies `custom.sling.Query`. This corresponds to the Java interface that was created in an earlier step.

In order for the service injection to work, the reference element must be configured correctly. In this example, notice that name of the reference is `resolverFactory`. Also notice that it's based on `org.apache.sling.api.resource.ResourceResolverFactory`, that is part of the Sling API.

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

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

## Deploy the bundle to Adobe CQ

[To the top](#)

Once you deploy the OSGi bundle, you are able to invoke the `getJCRData` method defined in the `QueryImp` class (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 Console.

Id	Name	Version	Category	Status	Actions
252	Training Bundle (custom.sling.quersling-bundle)	1.0.0-SNAPSHOT		Active	
	Symbolic Name	custom.sling.quersling-bundle			
	Version	1.0.0-SNAPSHOT			
	Bundle Location	inputstream\quersling-bundle-1.0-SNAPSHOT.jar			
	Last Modification	Tue Oct 01 13:29:46 EDT 2013			
	Description	Maven Multimodule project for Training.			
	Start Level	20			
	Exported Packages	custom.sling, version=1.0.0-SNAPSHOT			
	Imported Packages	com.day.cq.wcm.api, version=6.0.0 from com.day.cq.wcm.cq-wcm-api (214); org.apache.sling.api.resource, version=2.1.0 from org.apache.sling.api (92); org.osgi.framework, version=1.5.0 from org.apache.felix.framework (0); org.osgi.service.component, version=1.1.0 from org.apache.felix.scr (41); org.sling, version=1.6.4 from org.sling.api (11)			
	Service ID 1151	Types: custom.sling.Query Service PID: custom.sling.QueryImp Component Name: custom.sling.QueryImp Component ID: 1151			
	Service ID 1152	Types: java.lang.Runnable Service PID: custom.sling.SimpleDSComponent Component Name: custom.sling.SimpleDSComponent Component ID: 1152			
	Manifest Headers	Bundle-LastModified: 1380648522821 Bundle-Id: 1.0.0-SNAPSHOT Bundle-Name: Training Bundle Bundle-SymbolicName: custom.sling.quersling-bundle Bundle-Version: 1.0.0-SNAPSHOT Created-By: Apache Maven Bundle Plugin Export-Packages: custom.sling; uses="com.day.cq.wcm.api, org.apache.sling.api.resource, org.osgi.framework, org.osgi.service.component, org.sling, version=1.0.0-SNAPSHOT"; Import-Packages: com.day.cq.wcm.api, org.apache.sling.api.resource, version="2.1.0", org.osgi.framework, version="1.5.0", org.osgi.service.component, version="1.1.0", org.sling, version="1.6.4"; Manifest-Version: 1.0 Service-Component: OSGI-INF/serviceComponents.xml Tool: Bund-1.30.0			

### Apache Felix Web Console Bundles view

Deploy the OSGi bundle 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\quersling\bundle\target`).
5. Click Install.
6. Click the Refresh Packages button.
7. Check the bundle with the highest Id.
8. Click Active. Your new bundle should now be listed with the status Active.  
If the status is not Active, check the CQ error.log for exceptions.

## Modify the render component to invoke getJCRData method

[To the top](#)

Modify the `templateSling.jsp` file to invoke the `getJCRData` method that is defined by the `QueryImpl` class that uses the Sling API. The argument of this method is a string value that specifies the location of the JCR content. For example, `/content/geometrix/en/services`.

To invoke this service, you call `sling.getService` method as shown in the following example.

```
//create a custom.sling.Query instance
custom.sling.Query wfService =
sling.getService(custom.sling.Query.class);
```

The following code represents the templateSling JSP page.

```
1 <%@include file="/libs/foundation/global.jsp"%>
2 <%@taglib prefix="cq" uri="http://www.day.com/taglibs/cq/1.0" %>
3 <h1>TemplateSling Page</h1>
4 <%
5 //create a custom.sling.Query instance
6 custom.sling.Query wfService = sling.getService(custom.sling.Query.class);
7
8 %>
9 <h2>Use the Sling API to get title of the resource at /content/geometrixx/en/service
10
11 <h3>The title of the page is: <%= wfService.getJCRData("/content/geometrixx/en,
```

Modify the templateSling 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/slingApp/components/page/templateSling/templateSling.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 resource values

[To the top](#)

The final task is to create a site that contains a page that is based on the templateSling (the template created earlier in this development article). This web page invokes the `getJCRData` method as shown in the following illustration.

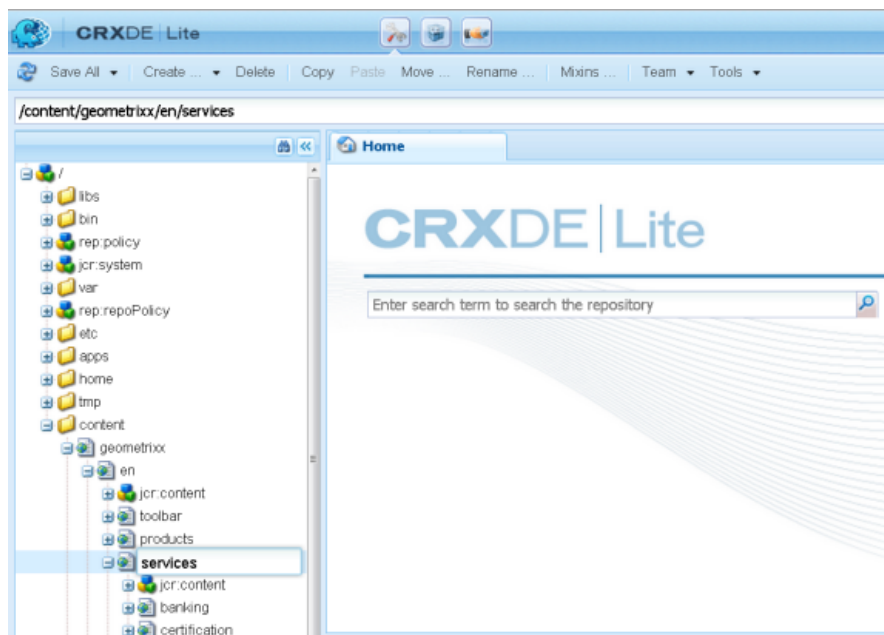
### TemplateSling Page

Use the Sling API to get title of the resource at `/content/geometrixx/en/service`

The title of the page is: Services

An AEM web page based on the templateSling template

The following illustration shows the resource located in the AEM JCR that was retrieved in this example.



CRXDE lite displaying a resource located in the JCR AEM

Create a CQ web page that is based on the templateSling template.

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 templateSling 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.



Twitter™ and Facebook posts are not covered under the terms of Creative Commons.

[Legal Notices](#) | [Online Privacy Policy](#)