# Adobe CQ Help / Creating Adobe CQ OSGi bundles that use the Query Builder API

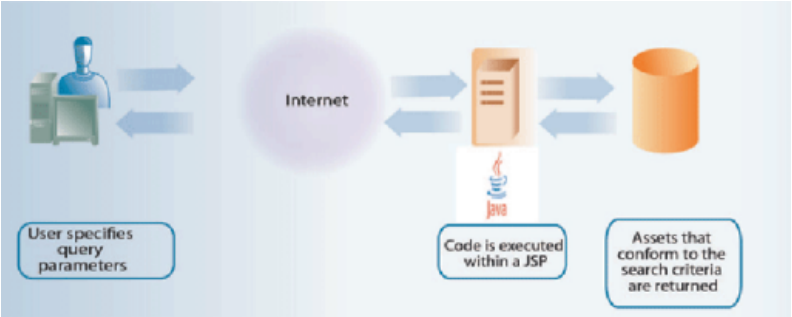
### Article Summary

| | |
|---|---|
| **Summary** | Discusses how to create an OSGi bundle that contains the AEM Query Builder API. This OSGi operation contains application logic to search the AEM repository. This article also discusses how to create a web page that invokes an operation exposed by the OSGi bundle and display the results in a grid control.<br>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.<br>This article has been updated to replace the use of a `SlingRepository` instance with a `ResourceResolverFactory` instance. The `ResourceResolverFactory` is used to create a `Session` instance that is required to use the AEM Query Builder API. Now a `Session` instance is created by using the `adaptTo` method:<br>`resourceResolver.adaptTo(Session.class);` |
| **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

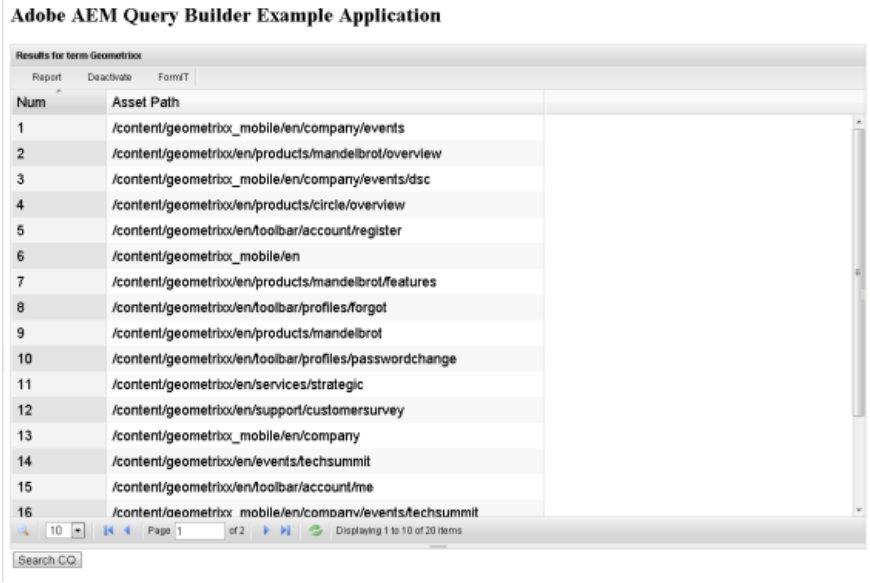
You can create an AEM application that searches the CQ repository for assets and displays results to the end user. For example, you can search CQ pages under a specific repository node (for example, nodes under /content) and look for a specific search term. All content that satisfy the search criteria are included in the search results. To search the Adobe CQ repository, you use the AEM Query Builder API. This API requires that you define search parameters, and an optional filter. After you execute the query, the results are stored in a result set. You can display the result set in an Adobe CQ web page.

When working with the Query Builder API, you can use a Java API or a Restful API. This development article uses the Adobe CQ Query Builder Java API within an OSGi bundle to perform searches.




A user specifies search criteria and the CQ repository is searched using the defined criteria

The server-side query builder (QueryBuilder) accepts a query description, creates and runs a query. For example, you can search for CQ pages located under /content using the search term: "Geometrixx". The following illustration shows the result set for this query.




Results produced by using the Query Builder API

The query description is simply a set of predicates (Predicate). Examples include a full-text predicate, which corresponds to the `jcr:contains()` function in XPath, and an image size predicate that looks for width and height properties in the DAM asset subtree.

For each predicate type, there is an evaluator component (`PredicateEvaluator`) that knows how to handle that specific predicate for XPath, filtering, and facet extraction. It is very easy to create custom evaluators, which are plugged-in through the OSGi component runtime.

The REST API provides access to exactly the same features through HTTP with responses being sent in JSON.

For more information about the Adobe CQ Query Builder API, see http://dev.day.com/docs/en/cq/current/dam/customizing_and_extendingcq5dam/query_builder.html.

This development article walks you through how to build an OSGi bundle that uses the Query Builder API and searches the Adobe CQ repository. To create an AEM web application that queries assets from the JCR by using the Query Builder API, perform these tasks:

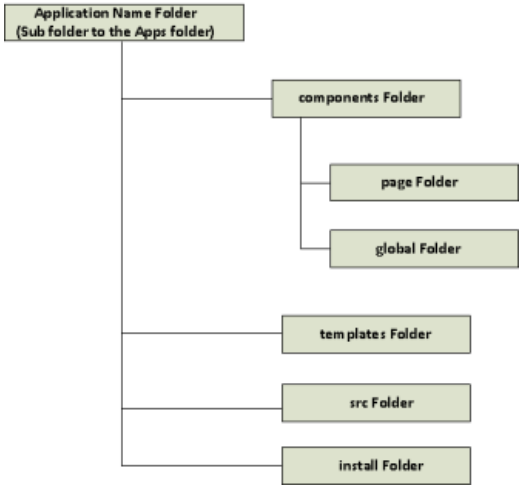
1. Create an Adobe CQ application folder structure.
2. Create a template on which the page component is based.
3. Create a render component that uses the template.
4. Setup Maven in your development environment.
5. Create an Adobe CQ archetype project.
6. Add Java files that use the Query Builder API 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. Add CSS and JQuery files to a `cq:ClientLibraryFolder` node.
11. Modify the render component to invoke an OSGi operation that queries the JCR.
12. Create a site that contains a page that lets a user search the AEM JCQ.

*Note: Instead of using the Query Builder API to search the AEM JCR, you can also use the JCR API. For details, see Querying Adobe Experience Manager Data using the JCR 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 *queryBuilder*.
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 *templateQueryBuilder*.
- **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 *queryBuilder/components/page/templateQueryBuilder*.
- **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/queryBuilder/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 *templateQueryBuilder*.
- **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 templateQueryjsp located at:
/apps/queryBuilder/components/page/templateQueryBuilder/templateQueryBuilder.jsp.
8. Enter the following JSP code.

```
<html>
<head>
<title>Hello World !!!</title>
</head>
<body>
<h1>Hello Query Builder API!!!</h1>
<h2>This page will query the AEM JCR using the Query Builder API</h2>
</body>
</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/.

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 sett
   | the parameter in question.
   |
   | Default: true
  <interactiveMode>true</interactiveMode>
  -->

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

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

```
      <pluginGroup>com.your.plugins</pluginGroup>
      -->
    </pluginGroups>

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

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

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

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

    <!-- profiles
     | This is a list of profiles which can be activated in a variety of ways, an
     | the build process. Profiles provided in the settings.xml are intended to p
     | specific paths and repository locations which allow the build to work in t
```

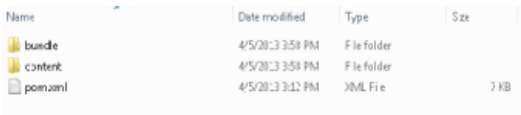
```
  |
  | For example, if you have an integration testing plugin - like cactus - tha
  | your Tomcat instance is installed, you can provide a variable here such th
  | dereferenced during the build process to configure the cactus plugin.
  |
  | As noted above, profiles can be activated in a variety of ways. One way -
  | section of this document (settings.xml) - will be discussed later. Another
  | relies on the detection of a system property, either matching a particular
  | or merely testing its existence. Profiles can also be activated by JDK ver
  | value of '1.4' might activate a profile when the build is executed on a JD
  | Finally, the list of active profiles can be specified directly from the co
  |
  | NOTE: For profiles defined in the settings.xml, you are restricted to spec
  |       repositories, plugin repositories, and free-form properties to be us
  |       variables for plugins in the POM.
  |
  |-->
<profiles>
  <!-- profile
    | Specifies a set of introductions to the build process, to be activated u
    | mechanisms described above. For inheritance purposes, and to activate pr
    | or the command line, profiles have to have an ID that is unique.
    |
    | An encouraged best practice for profile identification is to use a consi
    | for profiles, such as 'env-dev', 'env-test', 'env-production', 'user-jdo
    | 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>
  -->
```

```
249 <profile>
250
251                 <id>adobe-public</id>
252
253                 <activation>
254
255                     <activeByDefault>true</activeByDefault>
256
257                 </activation>
258
259                 <repositories>
260
261                   <repository>
262
263                     <id>adobe</id>
264
265                     <name>Nexus Proxy Repository</name>
266
267                     <url>http://repo.adobe.com/nexus/content/groups/public/</ur
268
269                     <layout>default</layout>
270
271                   </repository>
272
273                 </repositories>
274
275                 <pluginRepositories>
276
277                   <pluginRepository>
278
279                     <id>adobe</id>
280
281                     <name>Nexus Proxy Repository</name>
282
283                     <url>http://repo.adobe.com/nexus/content/groups/public/</ur
284
285                     <layout>default</layout>
286
287                   </pluginRepository>
288
289                 </pluginRepositories>
290
291             </profile>
292
293 </profiles>
294
295   <!-- activeProfiles
296    | List of profiles that are active for all builds.
297    |
298   <activeProfiles>
299     <activeProfile>alwaysActiveProfile</activeProfile>
300     <activeProfile>anotherAlwaysActiveProfile</activeProfile>
301   </activeProfiles>
302   -->
303 </settings>
```

## Create an Adobe CQ archetype project

To the top

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




Default files created by the Maven archetype plugin

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

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

2. Run the following Maven command:

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

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

- **groupId**: custom.querybuilder
- **artifactId**: querybuilder

- **version**: 1.0-SNAPSHOT
- **package**: custom.querybuilder
- **appsFolderName**: querybuilder-training
- **artifactName**: QueryBuilderTraining 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\querybuilder. Run the following Maven command:

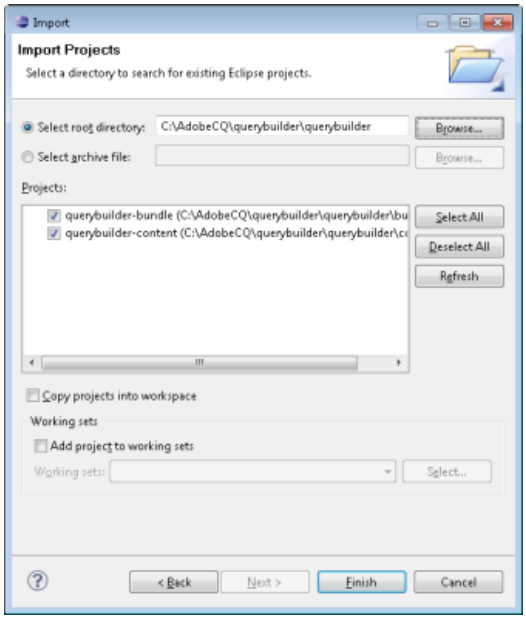
```
mvn eclipse:eclipse
```

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

## Add Java files to the Maven project using Eclipse

To the top

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




The Eclipse Import Project dialog

The next step is to add Java files to the `custom.querybuilder` package. The Java files that you create in this section use the Adobe CQ QueryBuilder API. For information, see http://dev.day.com/docs/en/cq/current/javadoc/com/day/cq/search/QueryBuilder.html.

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

- A Java interface named `SearchService`.
- A Java class named `SearchServiceImpl` that implements the `SearchService` interface.

#### SearchService interface

The following code represents the `SearchService` interface that contains a method named `SearchCQForContent`. The implementation logic for this method is located in the `SearchServiceImpl` class. This method uses the QueryBuilder API to search the AEM JCR.

```
package custom.querybuilder;

public interface SearchService {

    public String SearchCQForContent();

}
```

**SearchServiceImpl class**

The `SearchServiceImp` 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 `QueryBuilder` instance is injected into the `SearchCQForContent` method. This instance is required to perform Query Builder operations from within an OSGi bundle. To inject a `QueryBuilder` instance, you use the `@Reference` annotation to define a class member, as shown in the following example.

```
@Reference
    private QueryBuilder builder;
```

Within the `SearchCQForContent` method, a `ResourceResolverFactory` instance is injected into the `SearchCQForContent` method. This instance is required to create a Session instance that lets you create a `Query` instance. To inject a `ResourceResolverFactory` instance, you use the `@Reference` annotation to define a class member, as shown in the following example.

```
@Reference
private ResourceResolverFactory resolverFactory;


@Override
public String SearchCQForContent() {
try {

    //Create a Session
    ResourceResolver resourceResolver = resolverFactory.getAdministrativeResour
    session = resourceResolver.adaptTo(Session.class);
```

The `SearchCQForContent` method returns an XML schema that contains data that conforms to the QueryBuilder search. In this development article, the following QueryBuilder search is used.

```
//Define the search term
String fulltextSearchTerm = "Geometrixx";

// create query description as hash map (simplest way, same as form post)
Map<String, String> map = new HashMap<String, String>();

 // create query description as hash map (simplest way, same as form post)
map.put("path", "/content");
map.put("type", "cq:Page");
map.put("group.p.or", "true"); // combine this group with OR
map.put("group.1_fulltext", fulltextSearchTerm);
map.put("group.1_fulltext.relPath", "jcr:content");
map.put("group.2_fulltext", fulltextSearchTerm);
map.put("group.2_fulltext.relPath", "jcr:content/@cq:tags");

// can be done in map or with Query methods
map.put("p.offset", "0"); // same as query.setStart(0) below
map.put("p.limit", "20"); // same as query.setHitsPerPage(20) below

//Create a Query instance
Query query = builder.createQuery(PredicateGroup.create(map), session);
```

The following Java code shows how to process the result set of the query. Notice that the results are placed into XML. The XML is passed back to the client web page and displayed in the client (this is shown later in this development article).

```
//Get the query results
SearchResult result = query.getResult();

// paging metadata
int hitsPerPage = result.getHits().size(); // 20 (set above) or lower
long totalMatches = result.getTotalMatches();
long offset = result.getStartIndex();
long numberOfPages = totalMatches / 20;

//Place the results in XML to return to client
DocumentBuilderFactory factory =     DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();

//Start building the XML to pass back to the AEM client
```

```
Element root = doc.createElement( "results" );
doc.appendChild( root );

// iterating over the results
for (Hit hit : result.getHits()) {
   String path = hit.getPath();

    //Create a result element
    Element resultel = doc.createElement( "result" );
    root.appendChild( resultel );

    Element pathel = doc.createElement( "path" );
    pathel.appendChild( doc.createTextNode(path ) );
    resultel.appendChild( pathel );
}
```

The following Java code represents the `SearchServiceImpl` class.

```
package custom.search;

import org.w3c.dom.Document;
import org.w3c.dom.Element;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.StringWriter;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;

import java.util.HashMap;
import java.util.Map;

import javax.jcr.Repository;
import javax.jcr.SimpleCredentials;
import javax.jcr.Node;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.jackrabbit.commons.JcrUtils;

import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Service;
import javax.jcr.RepositoryException;
import org.apache.felix.scr.annotations.Reference;
import org.apache.jackrabbit.commons.JcrUtils;

import javax.jcr.Session;
import javax.jcr.Node;


//Sling Imports
import org.apache.sling.api.resource.ResourceResolverFactory ;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.api.resource.Resource;


//QUeryBuilder APIs
import com.day.cq.search.QueryBuilder;
import com.day.cq.search.Query;
import com.day.cq.search.PredicateGroup;
import com.day.cq.search.result.SearchResult;
import com.day.cq.search.result.Hit;


//This is a component so it can provide or consume services
@Component

@Service
public class SearchServiceImpl implements SearchService {

/** Default log. */
protected final Logger log = LoggerFactory.getLogger(this.getClass());

private Session session;

//Inject a Sling ResourceResolverFactory
```

```
67  @Reference
68  private ResourceResolverFactory resolverFactory;
69
70  @Reference
71  private QueryBuilder builder;
72
73  @Override
74  public String SearchCQForContent() {
75  try {
76
77   //Invoke the adaptTo method to create a Session
78      ResourceResolver resourceResolver = resolverFactory.getAdministrativeResour
79      session = resourceResolver.adaptTo(Session.class);
80
81      String fulltextSearchTerm = "Geometrixx";
82
83      // create query description as hash map (simplest way, same as form post)
84      Map<String, String> map = new HashMap<String, String>();
85
86   // create query description as hash map (simplest way, same as form post)
87
88      map.put("path", "/content");
89      map.put("type", "cq:Page");
90      map.put("group.p.or", "true"); // combine this group with OR
91      map.put("group.1_fulltext", fulltextSearchTerm);
92      map.put("group.1_fulltext.relPath", "jcr:content");
93      map.put("group.2_fulltext", fulltextSearchTerm);
94      map.put("group.2_fulltext.relPath", "jcr:content/@cq:tags");
95      // can be done in map or with Query methods
96      map.put("p.offset", "0"); // same as query.setStart(0) below
97      map.put("p.limit", "20"); // same as query.setHitsPerPage(20) below
98
99      Query query = builder.createQuery(PredicateGroup.create(map), session);
100
101     query.setStart(0);
102     query.setHitsPerPage(20);
103
104     SearchResult result = query.getResult();
105
106     // paging metadata
107     int hitsPerPage = result.getHits().size(); // 20 (set above) or lower
108     long totalMatches = result.getTotalMatches();
109     long offset = result.getStartIndex();
110     long numberOfPages = totalMatches / 20;
111
112     //Place the results in XML to return to client
113     DocumentBuilderFactory factory =         DocumentBuilderFactory.newInstance
114 DocumentBuilder builder = factory.newDocumentBuilder();
115 Document doc = builder.newDocument();
116
117     //Start building the XML to pass back to the AEM client
118     Element root = doc.createElement( "results" );
119     doc.appendChild( root );
120
121
122     // iterating over the results
123     for (Hit hit : result.getHits()) {
124            String path = hit.getPath();
125             //Create a result element
126             Element resultel = doc.createElement( "result" );
127             root.appendChild( resultel );
128
129             Element pathel = doc.createElement( "path" );
130             pathel.appendChild( doc.createTextNode(path ) );
131             resultel.appendChild( pathel );
132
133     }
134
135     //close the session
136     session.logout();
137     return convertToString(doc);  // Convert the XML to a string to return to t
138
139 }
140  catch(Exception e){
141      log.info(e.getMessage());
142   }
143  return null;
144 }
145
146 private String convertToString(Document xml)
147 {
148 try {
149    Transformer transformer = TransformerFactory.newInstance().newTransformer();
150   StreamResult result = new StreamResult(new StringWriter());
```

```
    DOMSource source = new DOMSource(xml);
    transformer.transform(source, result);
    return result.getWriter().toString();
  } catch(Exception ex) {
            ex.printStackTrace();
  }
    return null;
       }
   }

```

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

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

Because the QueryBuilder API is used, a Maven dependency on the repository that contains the QueryBuilder API exists. 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 QueryBuilder API in your Java code.

```
<dependency>
    <groupId>com.day.cq</groupId>
    <artifactId>cq-search</artifactId>
    <version>5.5.4</version>
    <scope>provided</scope>
</dependency>
```

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

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
    <modelVersion>4.0.0</modelVersion>
    <!-- ====================================================================
    <!-- P A R E N T P R O J E C T D E S C R I P T I O N -->
    <!-- ====================================================================
    <parent>
        <groupId>custom.search</groupId>
        <artifactId>search</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>search-bundle</artifactId>
    <packaging>bundle</packaging>
    <name>search package Bundle</name>

```

```
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.search.search-bundle</Bund
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/search/insta
55                      <usePut>true</usePut>
56                  </configuration>
57              </plugin>
58          </plugins>
59      </build>
60
61  <dependencies>
62
63      <dependency>
64              <groupId>com.day.cq</groupId>
65              <artifactId>cq-search</artifactId>
66              <version>5.5.4</version>
67              <scope>provided</scope>
68          </dependency>
69
70          <dependency>
71              <groupId>org.osgi</groupId>
72              <artifactId>org.osgi.compendium</artifactId>
73          </dependency>
74          <dependency>
75              <groupId>org.osgi</groupId>
76              <artifactId>org.osgi.core</artifactId>
77          </dependency>
78          <dependency>
79              <groupId>org.apache.felix</groupId>
80              <artifactId>org.apache.felix.scr.annotations</artifactId>
81          </dependency>
82          <dependency>
83              <groupId>org.slf4j</groupId>
84              <artifactId>slf4j-api</artifactId>
85          </dependency>
86
87
88          <dependency>
89           <groupId>org.apache.felix</groupId>
90
91           <artifactId>org.osgi.core</artifactId>
92
93           <version>1.4.0</version>
94        </dependency>
95
96
97
98      <dependency>
99      <groupId>org.apache.jackrabbit</groupId>
100     <artifactId>jackrabbit-core</artifactId>
101     <version>2.4.3</version>
102     </dependency>
103
104     <dependency>
105     <groupId>org.apache.jackrabbit</groupId>
```

```
    <artifactId>jackrabbit-jcr-commons</artifactId>
```

## 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 `SearchServiceImp` 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="custom.search.SearchServiceImpl">
        <implementation class="custom.search.SearchServiceImpl"/>
        <service servicefactory="false">
            <provide interface="custom.search.SearchService"/>
        </service>
        <property name="service.pid" value="custom.search.SearchServiceImpl"/>
        <reference name="resolverFactory" interface="org.apache.sling.api.resour
        <reference name="builder" interface="com.day.cq.search.QueryBuilder" car
    </scr:component>
</components>
```

There are a couple of points to note about this XML file. First, notice that the implementation class element specifies `custom.querybuilder.SearchServiceImpl`. The service element contains an interface attribute that specifies `custom.querybuilder.SearchService`. 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 `builder`. Also notice that it's based on `com.day.cq.search.QueryBuilder`, that is part of the QueryBuilder API.

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

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

## Deploy the bundle to Adobe CQ

To the top

Once you deploy the OSGi bundle, you are able to invoke the `SearchCQForContent` method defined in the `SearchServiceImpl` 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 Conole.

Apache Felix Web Console Bundles view

You will also be able to view the service defined in the OSGi bundle by using the Service tab in the Apache Felix Web Console.

Apache Felix Web Console Service 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\querybuilder\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.

## Add the data grid library to a cq:ClientLibraryFolder node

To the top

You add CSS files and JQuery framework files 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.

In addition to the JQuery framework file, a data grid plugin named flexigrid is used. This plugin is used to display search results in a tabular format. Download the flexigrid plugin from the following URL:

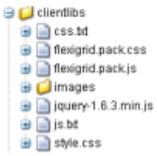
http://flexigrid.info/

*Note: You can use other data grid plugins in an AEM application as well. For example, the following AEM article uses another data grid control named DataTables. For information, see Querying Adobe Experience Manager Data using the JCR API.*

Download and extract the flexigrid archive file. The AEM application uses these files from the archive file:

- flexigrid.pack.css
- flexigrid.pack.js

In addition, copy the images folder to the `cq:ClientLibraryFolder` node. The following illustration displays the files that you must add to this node.




The ClientLibs folder that contains flexigrid plugin files

To add CSS files and JQuery framework files 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. 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 files and JQuery libraries in the page. The categories property informs CQ which clientlibs must be included.

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

#### Text files

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

flexigrid.pack.css. Likewise, the js.txt file contains the JS file names jquery-1.6.3.min.js and flexigrid.pack.js.

#### Add the files to the ClientLibs folder

1. Right-click /apps/users/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 files are located. Drag and drop the JS files to the clientlibs node by using CRXDE.
5. On your file system, navigate where you placed the CSS files. Drag and drop the CSS files to the clientlibs folder by using CRXDE.
6. Add a TXT file to the clientlibs folder named js.txt. Add the content specified in this section.
7. Add a TXT file to the clientlibs node named css.txt. Add the content specified in this section.

---

### Modify the render component to invoke SearchService operations

To the top

To create the AEM client that searches the JCR by using the OSGi bundle that contains the Query Builder API, create these files:

- **query.json.jsp**: contains application logic that calls the OSGi bundle's `SearchCQForContent` method.
- **templateQueryBuilder.jsp**: contains application logic that calls the query.json.jsp and displays the search results in a grid control.

#### Create the query.json.jsp

Add a new JSP file named query.json.jsp to the following CQ path:

apps/querybuilder/components/page/templateQueryBuilder/

In query.json.jsp, you create a `SearchService` instance by using the `sling.getService` method, as shown in the following example:

```
custom.querybuilder.SearchService cs =
sling.getService(custom.querybuilder.SearchService.class);
```

Pass the fully qualified name of the service to `sling.getService` method. Because the OSGi bundle is a managed component that injects a `QueryBuilder` instance into the service, you must use the `sling.getService` method to create a `QueryBuilder` object.

If you attempt to create a `SearchService` using the new operation, the OSGi bundle is not considered a managed component and will not successfully inject a `QueryBuilder` instance. A Java null pointer exception is thrown.

After you create a `SearchService` object by using `sling.getService`, you can invoke the `SearchCQForContent` method exposed by the service. This method retrurns XML that contains the results of the search.

The following code represents the query.json.jsp file.

```
<%@page session="false" %>
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="org.apache.sling.jcr.api.SlingRepository" %>
<%@ page import="custom.search.SearchServiceImpl" %>
<%@ page import="com.day.cq.security.UserManagerFactory" %>
<%@ page import="com.day.cq.security.User" %>
<%@ page import="com.day.cq.security.Authorizable" %>
<%@ page import="com.day.cq.security.profile.Profile" %>
<%@ page import="java.util.Iterator" %>
<%@ page import="java.util.List" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="org.apache.sling.commons.json.io.JSONWriter" %>

<%@page import="com.day.cq.dam.api.Asset"%>
<%

String filter = request.getParameter("filter");

//create a SeachService instance
custom.querybuilder.SearchService queryBuilder = sling.getService(custom.querybu

String XML = queryBuilder.SearchCQForContent() ;

```

```
//Send the data back to the client
JSONWriter writer = new JSONWriter(response.getWriter());
writer.object();
writer.key("xml");
writer.value(XML);

writer.endObject();
%>
```

**Modify the templateQueryBuilder.jsp**

Modify the templateQueryBuilder.jsp file to call the query.json.jsp and write the search results to the Flexigrid control. In this example, a JQuery Ajax HTTP request is used to invoke the code in the query.json.jsp. This code shows the submit method that is called when the user clicks the Search CQ button.

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

          };

 var url = location.pathname.replace(".html", "/_jcr_content.query.json");

  $.ajax(url, {
      dataType: "text",
      success: function(rawData, status, xhr) {
          var data;
          try {
              data = $.parseJSON(rawData);
               var val = data.xml  ;

              //Display the results in the Grid control
              var jsonObj = []    ;
              var index = 1 ;

              $(val).find('result').each(function(){

                  var Template = {};
                  var field = $(this);

                  Template["id"] =  index;
                  Template["name"] =  $(field).find('path').text();

                  //Push JSON
                  jsonObj.push(Template)   ;
                  index++;
              });

            //Populate the Data Grid Control
            var gridData =  formatCustomerResults(jsonObj)  ;
            $("#flex1").flexAddData(eval(gridData));

 }  catch(err) {
          failure(err);
      }
  },
  error: function(xhr, status, err) {
          failure(err);
      }
  });
});
```

Notice that for each result in the XML, the data is placed into a JSON data structure that is used to populate the data grid control.

The following code represents the entire TemplateQueryBuilder.jsp file.

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


jQuery(function ($) {


$("#flex1").flexigrid(
{
        dataType: 'json',
        colModel : [
        {display: 'Num', name : 'id', width : 100, sortable : true, align: 'le
        {display: 'Asset Path', name : 'name', width : 500, sortable : true, a
```

```
15
16      ],
17      buttons : [
18      {name: 'Report', bclass: 'report', onpress : test},
19      {name: 'Deactivate', bclass: 'delete', onpress : test},
20      {name: 'FormIT', bclass: 'view', onpress : test},
21      {separator: true}
22      ],
23          searchitems : [
24              {display: 'First Name', name : 'first_name'},
25              {display: 'Surname', name : 'surname', isdefault: true},
26              {display: 'Position', name : 'position'}
27          ],
28      sortname: "id",
29      sortorder: "asc",
30      usepager: true,
31      title: "Results for term Geometrixx",
32      useRp: true,
33      rp: 10,
34      showTableToggleBtn: false,
35      resizable: true,
36      width: 1000,
37      height: 470,
38      singleSelect: true
39      }
40  );
41
42
43  $('#submit').click(function() {
44  var failure = function(err) {
45
46    alert("Unable to retrive data "+err);
47    };
48
49
50    var url = location.pathname.replace(".html", "/_jcr_content.query.json");
51
52    $.ajax(url, {
53            dataType: "text",
54            success: function(rawData, status, xhr) {
55                var data;
56                try {
57                    data = $.parseJSON(rawData);
58
59                    var val = data.xml  ;
60
61                  //Display the results in the Grid control
62                var jsonObj = []    ;
63                var index = 1 ;
64
65                $(val).find('result').each(function(){
66
67                        var Template = {};
68                        var field = $(this);
69
70                        Template["id"] =  index;
71                        Template["name"] =  $(field).find('path').text();
72
73                        //Push JSON
74                        jsonObj.push(Template)   ;
75                        index++;
76
77            });
78
79      //Populate the Flexigrid control
80       var gridData =  formatCustomerResults(jsonObj)  ;
81       $("#flex1").flexAddData(eval(gridData));
82
83        } catch(err) {
84            failure(err);
85            }
86        },
87        error: function(xhr, status, err) {
88                failure(err);
89        }
90    });
91     });
92
93
94  });
95
96  function test() {
97      alert("Not implemented yet.");
98  }
```

```

function formatCustomerResults(Templates){

    var rows = Array();
    var temp =      Templates ;

    for (i = 0; i <temp.length; i++) {
        var item = temp[i];

        rows.push({ cell: [item.id,
            item.name
        ]
        });
    }

    var len =  temp.length;

    return {
        total: len,
        page: 1,
        rows: rows
    }

};


</script>

<body>
<h2>Adobe AEM Query Builder Example Application</h2>

 <table id="flex1" width="1050">

 </table>

 <input type="button" value="Search CQ"  name="submit" id="submit" value="Subm

</body>
</html>
```

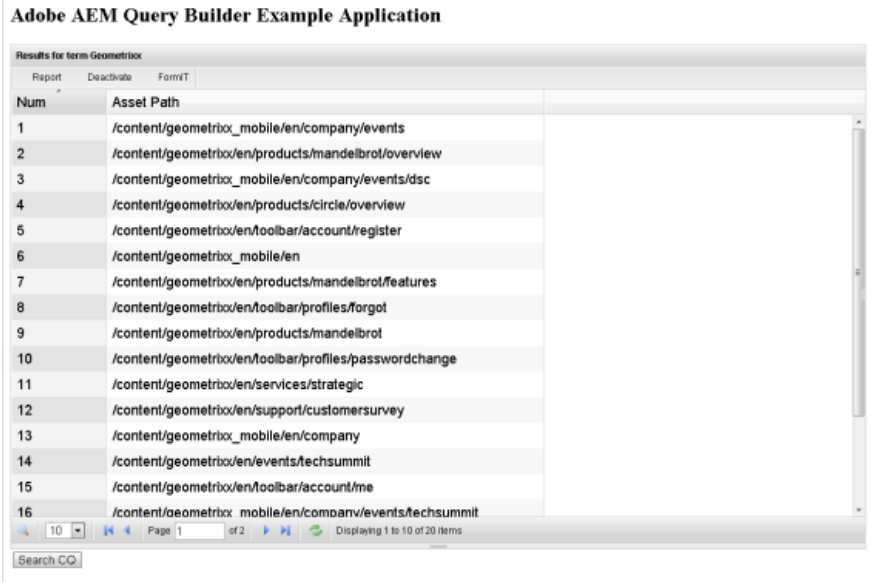
Modify the templateQueryBuilder file:

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. (If you are using AEM 5.6, click Tool in the left menu)
3. Double-click apps/querybuilder/components/page/templateQueryBuilder/templateQueryBuilder.jsp.
4. Replace the JSP code with the new code shown in this section.
5. Click Save All.

## Create an AEM web page that searches the JCR

To the top

The final task is to create a site that contains a page that is based on the templateQueryBuilder (the template created earlier in this development article). When the user clicks the Search CQ button, the search results are displayed in the data grid control.




The AEM application that is created in this development article

Create an AEM web page that queries data from the AEM JCR:

1. Go to the CQ Websites page at http://localhost:4502/siteadmin#/content.
2. Select New Page.
3. Specify the title of the page in the Title field.
4. Specify the name of the page in the Name field.
5. Select *templateQueryBuilder* 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. You should see a page similar to the previous illustration.

**See also**

Congratulations, you have just created an AEM OSGi bundle 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.