Adobe CQ Help / Creating Adobe CQ bundles that consume web services

Article summary

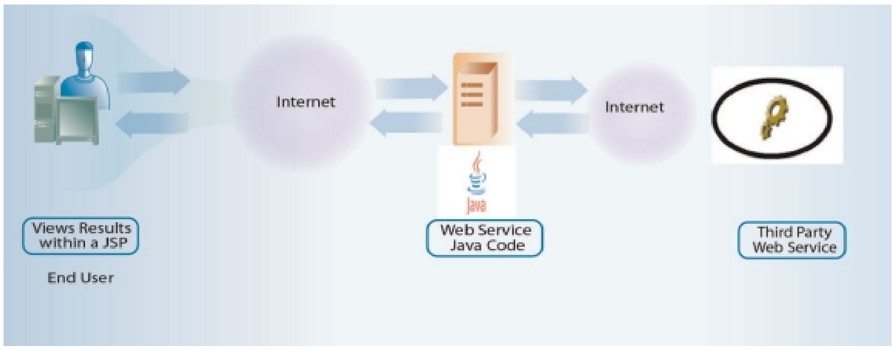
Summary	Discusses how to create and deploy an Adobe CQ OSGi bundle that consumes a third-party web service. In addition, discusses how to invoke an OSGi bundle operation from the client web page. This article talks about creating Java Proxy classes using JAX-WS. You can use Apache CXF to create Java proxy classes. For information, see Creating Adobe CQ bundles using Apache CXF that consume web services. This article discusses using a WSDL to consume a web service. To learn how to build an AEM application that consumes a Restful web service, see Creating Adobe Experience Manager services that invoke third party Restful web services.
Digital Marketing Solution(s)	Adobe Experience Manager (Adobe CQ)
Audience	Developer (intermediate)
Required Skills	Java, JQuery, web services, XML
Tested On	Adobe CQ 5.5, Adobe CQ 5.6

Note: This workflow works on Adobe CQ; however, you may encounter the following exception:

Caused by: java.lang.ClassNotFoundException: com.sun.xml.internal.ws.spi.ProviderImpl
at org.apache.sling.commons.classloader.impl.ClassLoaderFacade.loadClass(ClassLoaderFacade.java:127)
at java.lang.ClassLoader.loadClass(Unknown Source)
at javax.xml.ws.spi.FactoryFinder.safeLoadClass(Unknown Source)
... 107 more
Solution:
To fix this issue and ensure that you can create a bundle that consumes web services as described in this article, modify the sling.properties file located in the crx-quickstart\conf folder. Add the following line of code to this file: `sling.bootdelegation.com.sun=com.sun.*`*. Then restart the server. Once you perform this task, you can follow along with this article.*

Introduction

You can create an Adobe CQ bundle that consumes data from a third-party web service and displays the data in a web page. For example, assume that you use Adobe CQ to create a web site for a government department that tracks weather information. In this situation, you can create a CQ bundle that retrieves data from a third-party web service and displays the data within a form located in a web page. The following illustration shows data being retrieved from a third-party web service and displayed in a JSP.



A CQ application that consumes a third party web service

You can develop an OSGi bundle that contains Java proxy classes that were created by using JAX-WS or AXIS. That is, you can use a tool such as JAX-WS to generate the Java proxy classes that are based on the WSDL of an external web service. Then you can use these Java proxy classes within your OSGi bundle. The OSGi bundle that is created in this development article contains Java proxy classes that consume operations exposed by the following third-party WSDL:

http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL

An OSGi bundle lets you dynamically load, unload, configure, and control the Java module without restarting the server. The tool that is used in this development article to create the OSGi bundle is Eclipse. All of the instructions needed to build an OSGi bundle that contains Java web service proxy classes are covered in this development article.

Note: Before following along with this development article, make sure that you install Adobe CQ 5.5 and have it running. In addition, ensure that you have Eclipse 3.6.1 or higher. Eclipse is used to create the OSGi bundle that contains the Java proxy classes.

To create an Adobe CQ web application that uses an OSGi bundle that contains Java web service proxy classes, perform the following tasks:

1. Create an Adobe CQ application that contains the page that displays the data.
2. Create a template on which the page component is based.
3. Create a render component that uses the template. This component retrieves data from the OSGi bundle.
4. Create Java proxy classes that consume the soap stack of the external web service.
5. Create the OSGi bundle that contains the Java proxy classes.
6. Modify the JSP to call a Java method defined in the OSGi bundle. The Java methods return from the web service.
7. Create a new web page that displays data returned by the OSGi bundle.

Note: For information on how to place this application into a package (including the OSGi bundle), see Packaging Adobe AEM applications that contain an OSGi bundle.

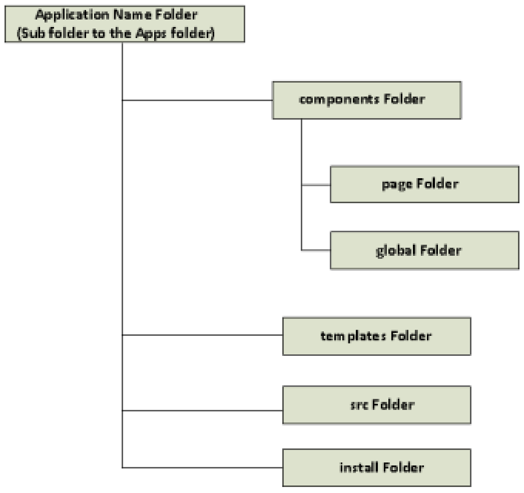
Note: For information on how to build a dynamic AEM JSP for this OSGi bundle, see Using JSONWriter objects to display Adobe CQ OSGi data.

Note: For information about how to invoke LiveCycle ES from Adobe CQ using web services, see Integrating LiveCycle into Adobe CQ applications.

Create a CQ 5.5 application folder structure

To the top

You can create an Adobe CQ application folder structure that contains templates, components, and pages. Create an application folder structure that displays data from a third-party web service. Before you create application assets such as templates, components, or pages, you create an application, which is essentially a specific folder. You can create this folder structure by using CRXDE Lite.



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 that you can base page components on

• **src**: contains source code that comprises an OSGi component (this development does not create an OSGi bundle using this folder or CRXDE. Rather Eclipse is used)
• **install**: contains a compiled OSGi bundles container

To create an application folder structure, perform these steps:

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 weatherapp.
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 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 CQ template, perform these steps:
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 weathertemplate.
• **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 `weatherapp/components/page/weathertemplate`.
• **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:
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/weatherapp/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 weathertemplate.
• **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 weatertemplate.jps located at: `/apps/weatherapp /components/page/weathertemplate /weathertemplate.jsp`.
8. Enter the following HTML code:

```
<html>
```

```
<head>
<title>Hello World !!!</title>
</head>
<body>
<h1>Hello Web Services</h1>
<h2>This page will contain data from a third-party web service</h2>
</body>
</html>
```

Create Java proxy classes that consume the soap stack of the external web service

To the top

You can use JAX-WS to convert a third-party WSDL to Java proxy classes. These Java classes enable you to invoke operations exposed by the soap stack. In this development article, the operations are exposed by the following WSDL:

http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL

Apache Ant lets you create a build script that generates Java proxy classes by referencing the WSDL.

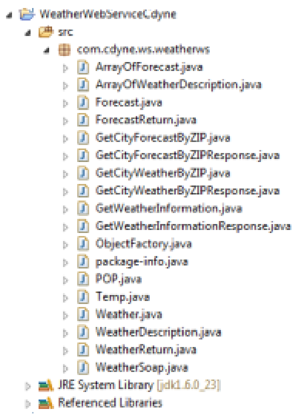
To create Java proxy classes using JAX-WS, perform these steps:
1. Install Apache Ant on the client computer. (See http://ant.apache.org/bindownload.cgi.)
• Add the bin directory to your class path.
• Set the ANT_HOME environment variable to the directory where you installed Ant.
2. Install JDK 1.6 or later.
• Add the JDK bin directory to your class path.
• Add the JRE bin directory to your class path. This bin is located in the [JDK_INSTALL_LOCATION]/jre directory.
• Set the JAVA_HOME environment variable to the directory where you installed the JDK. The JDK 1.6 includes the wsimport program used in the build.xml file. JDK 1.5 does not include that program.
3. Install JAX-WS on the client computer. See http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html. (Also see https://jax-ws.java.net.)
4. Use JAX-WS and Apache Ant to generate Java proxy classes. Create an Ant build script to accomplish this task. The following script is a sample Ant build script named build.xml:

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

<project basedir="." default="compile">


<target name="clean" >
<delete dir="classes" />
</target>

<target name="wsdl" depends="clean">
<mkdir dir="classes"/>
<exec executable="wsimport" failifexecutionfails="false" failonerror="true" resu
<arg line="-keep -d classes http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL"/>
</exec>
<fail unless="foundWSIMPORT">
!!! Failed to execute JDK's wsimport tool. Make sure that JDK 1.6 (or later) is
</fail>
</target>

<target name="compile" depends="clean, wsdl" >
<javac destdir="./classes" fork="true" debug="true">
<src path="./src"/>
</javac>
</target>

<target name="run">
<java classname="Client" fork="yes" failonerror="true" maxmemory="200M">
<classpath>
<pathelement location="./classes"/>
</classpath>
</java>
</target>
</project>
```

5. Create a BAT file to execute the Ant build script. The following command can be located within a BAT file that is responsible for executing the Ant build script:
ant -buildfile "build.xml" wsdl
6. Place the ANT build script in the C:\Program Files\Java\jaxws-ri\bin directory. The script writes the JAVA files to the ./classes folder. The script generates JAVA files that can invoke the operations

exposed by the WSDL.
7. Package the JAVA files into a JAR file by using Eclipse. Perform the following tasks:
• Create a new Java project that is used to package the proxy JAVA files into a JAR file.
• Create a source folder in the project.
• Create a cdyne.ws.weatherws package in the source folder. This is the package to which the Java proxy classes belong.
• Select the cdyne.ws.weatherws package and then import the JAVA files that Ant created (see the illustration after this list).
• Set the Java compiler's compliance level to 5.0 or greater.
• Build the project.
• Export the project as a JAR file named WeatherService.jar.

The following illustration shows an Eclipse project that contains all of the Java proxy classes located in a package named `cdyne.ws.weatherws`. Ensure that your project resembles the following illustration.



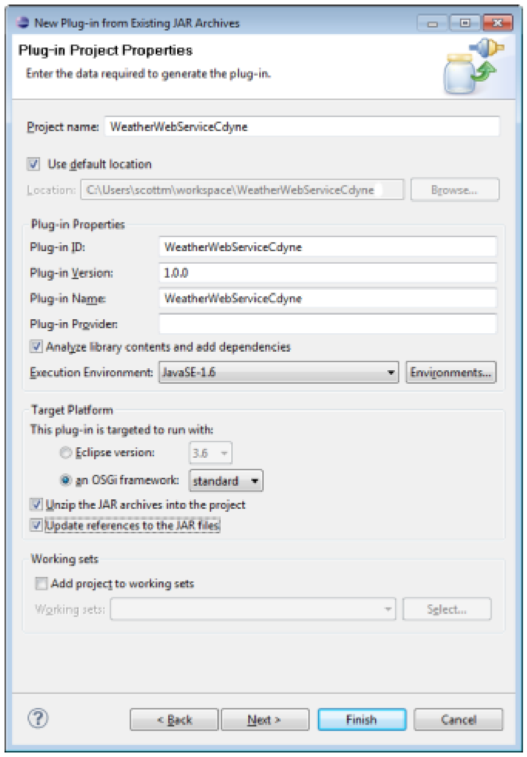
```
WeatherWebServiceCdyne
  src
    com.cdyne.ws.weatherws
      ArrayOfForecast.java
      ArrayOfWeatherDescription.java
      Forecast.java
      ForecastReturn.java
      GetCityForecastByZIP.java
      GetCityForecastByZIPResponse.java
      GetCityWeatherByZIP.java
      GetCityWeatherByZIPResponse.java
      GetWeatherInformation.java
      GetWeatherInformationResponse.java
      ObjectFactory.java
      package-info.java
      POP.java
      Temp.java
      Weather.java
      WeatherDescription.java
      WeatherReturn.java
      WeatherSoap.java
  JRE System Library [jdk1.6.0_23]
  Referenced Libraries
```

Create the OSGi bundle that contains the Java proxy classes

To the top

Create an OSGi bundle that contains WeatherService.jar file. Once you upload the OSGi bundle to Adobe CQ, you can call the methods located in Java files that are part of the `cdyne.ws.weatherws` package. That is, the OSGi bundle is able to invoke operations exposed by the WSDL. To create the OSGi bundle, you can use another Eclipse project. An OSGi bundle is essentially a collection of Java files and a MANIFEST.MF file.

To create an OSGi bundle that contains Java proxy classes using Eclipse, perform these steps:
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.
4. In the JAR selection dialog, click the Add external button, and browse to the WeatherService.jar file that you just created.
5. Click Next.
6. In the Plug-in Project properties dialog, ensure that you check the checkbox for Analyze library contents and add dependencies (see the following illustration).
7. Make sure that the Target Platform is the standard OSGi framework (see the following illustration).
8. Ensure the checkboxes for Unzip the JAR archives into the project and Update references to the JAR files are both checked (see the following illustration).



New Plug-in from Existing JAR Archives

Plug-in Project Properties

Enter the data required to generate the plug-in.

Project name: WeatherWebServiceCdyne

[✓] Use default location

Location: C:\Users\scottm\workspace\WeatherWebServiceCdyne

Plug-in Properties

Plug-in ID: WeatherWebServiceCdyne

Plug-in Version: 1.0.0

Plug-in Name: WeatherWebServiceCdyne

Plug-in Provider:

[✓] Analyze library contents and add dependencies

Execution Environment: JavaSE-1.6

Target Platform

This plug-in is targeted to run with:

( ) Eclipse version: 3.6

(●) an OSGi framework: standard

[✓] Unzip the JAR archives into the project

[✓] Update references to the JAR files

Working sets

[ ] Add project to working sets

Working sets:

9. Click Next, and then Finish.
10. Click on the Runtime tab.
11. Make sure that the Exported Packages list is populated
12. Make sure these 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. Ensure that the Import-Package header in MANIFEST.MF is also populated. The following code represents the MANIFEST.MF file.

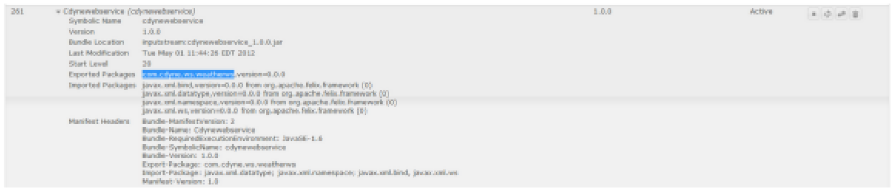
```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: WeatherWebServiceCdyne
Bundle-SymbolicName: WeatherWebServiceCdyne
Bundle-Version: 1.0.0
Export-Package: net.webservicex
Import-Package: javax.xml.namespace,
 javax.xml.bind,
 javax.xml.ws
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

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 with a name similar to cdynewebservice_1.0.0.jar.
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 look as follows: 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

32. If all goes well, the CQ error.log should have log entries such as follows:
INFO [FelixDispatchQueue] com.oracle.jdbc BundleEvent RESOLVED
INFO [FelixDispatchQueue] com.oracle.jdbc BundleEvent STARTED
INFO [FelixDispatchQueue] org.apache.felix.framework FrameworkEvent PACKAGES REFRESHED

Note: Now the Java proxy classes that know how to invoke operations exposed by the WSDL are part of the OSGI service container. You can create Java proxy objects and call their methods from a JSP.

The following illustration shows details about the OSGi bundle that is created in this article.



```
261    » Cdynewebservice (cdynewebservice)                                   1.0.0        Active
         Symbolic Name       cdynewebservice
         Version             1.0.0
         Bundle Location     inputstream:cdynewebservice_1.0.0.jar
         Last Modification   Tue May 01 11:44:26 EDT 2012
         Start Level         20
         Exported Packages   com.cdyne.ws.weatherws,version=0.0.0
         Imported Packages   javax.xml.bind,version=0.0.0 from org.apache.felix.framework (0)
                             javax.xml.datatype,version=0.0.0 from org.apache.felix.framework (0)
                             javax.xml.namespace,version=0.0.0 from org.apache.felix.framework (0)
                             javax.xml.ws,version=0.0.0 from org.apache.felix.framework (0)
         Manifest Headers    Bundle-ManifestVersion: 2
                             Bundle-Name: Cdynewebservice
                             Bundle-RequiredExecutionEnvironment: JavaSE-1.6
                             Bundle-SymbolicName: cdynewebservice
                             Bundle-Version: 1.0.0
                             Export-Package: com.cdyne.ws.weatherws
                             Import-Package: javax.xml.datatype; javax.xml.namespace; javax.xml.bind, javax.xml.ws
                             Manifest-Version: 1.0
```

You can view details, such as the Java packages that are part of the OSGi bundle. For example, notice that the `com.cdyne.ws.weatherws` package is specified in the previous illustration. (Step 18 in the previous procedure described how to login to Adobe CQ's Apache Felix Web Console.)

Modify the weathertemplate JSP to call the methods specified in the OSGi bundle

To the top

After you build the OSGi bundle and confirm that it has been deployed, you can call its methods from a JSP that is part of a page component. In this development article, the weathertemplate.jsp calls methods defined in the OSGi bundle that invoke operations exposed by the WSDL. Modify this JSP so that it resembles the following JSP code.

```
<%@include file="/libs/foundation/global.jsp"%>
<h1><%= properties.get("title", currentPage.getTitle()) %></h1>
<%

com.cdyne.ws.weatherws.Weather ww = new com.cdyne.ws.weatherws.Weather();

com.cdyne.ws.weatherws.WeatherSoap ws = ww.getWeatherSoap();

com.cdyne.ws.weatherws.WeatherReturn wr = ws.getCityWeatherByZIP("95101");

%>

<h2>The following describes the weather for 95101 ZIP Code</h2>
<h3><%= "The city is " +wr.getCity()%></h3>
<h3><%= "The state is " +wr.getState()%></h3>
<h3><%= "The description is " +wr.getDescription()%></h3>
<h3><%= "The wind is " +wr.getWind()%></h3>
<h3><%= "The current temp is " +wr.getTemperature()%></h3>
<h3><%= "The current Humidity is " +wr.getRelativeHumidity()%></h3>
```

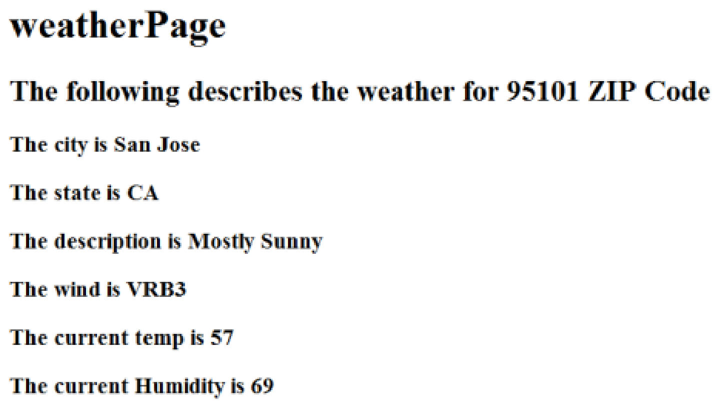
Notice how a `com.cdyne.ws.weatherws.Weather` object can be created and its methods are called. When a method, such as `getCity` is invoked, an operation exposed by the WSDL is called. The data returned by the web service is displayed in the JSP.

Create a page that displays data retrieved from the OSGi bundle

To the top

The final task that you perform in order to see a web page that displays data retrieved from an OSGi bundle is to create a site that contains a page that is based on the weathertemplate (the template created earlier in this development article).

The following illustration shows the data returned by the OSGi bundle being displayed in the web page. The weather information was returned by the web service.



weatherPage

The following describes the weather for 95101 ZIP Code

The city is San Jose

The state is CA

The description is Mostly Sunny

The wind is VRB3

The current temp is 57

The current Humidity is 69

To create a web page that displays data retrieved from an OSGi bundle, 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 Websites.
3. From the left hand pane, select Websites.
4. Select New, New Page.
5. Specify the title of the page in the Title field.
6. Specify the name of the page in the Name field.
7. Select weathertemplate 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 when creating the template, the template will not show up in the New Page dialog box.
8. 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 displaying data similar to the previous illustration.