Adobe CQ Help / Invoking Sling Servlets from AEM xtype widgets

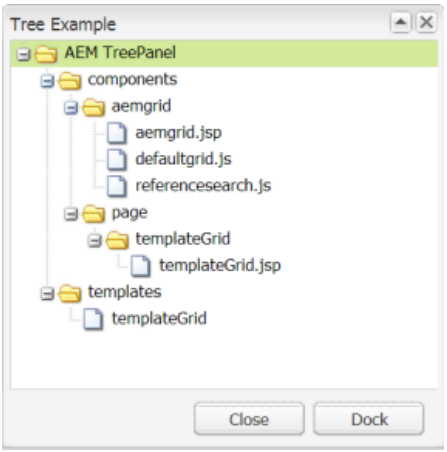
Article summary

Summary	Discusses the following points: • how to develop a CQ component that uses a TreePanel object based on a `CQ.Ext.tree.TreePanel`. A TreePanel lets AEM authors view tree stuctured data. • how to populate the TreePanel with JCR data by using a Sling Servlet. • how to invoke the Sling Servlet by using a `CQ.Ext.tree.TreeLoader` instance. • how to use an `org.apache.sling.commons.json.io.JSONWriter` within a Sling Servlet to encode JCR data as JSON data. • how to use a custom predicate that extends `com.day.cq.commons.predicate.AbstractNodePredicate` within a Sling Servlet that assits in searching JCR data. . This article uses an Adobe Maven Archetype project to build an OSGi bundle. If you are not familiar with an Adobe Maven Archetype project, it is recommended that you read the following article: Creating your first AEM Service using an Adobe Maven Archetype project.
Digital Marketing Solution(s)	Adobe Experience Manager (Adobe CQ)
Audience	Developer (intermediate)
Required Skills	JavaScript, HTML
Tested On	Adobe CQ 5.5, Adobe CQ 5.6

Introduction

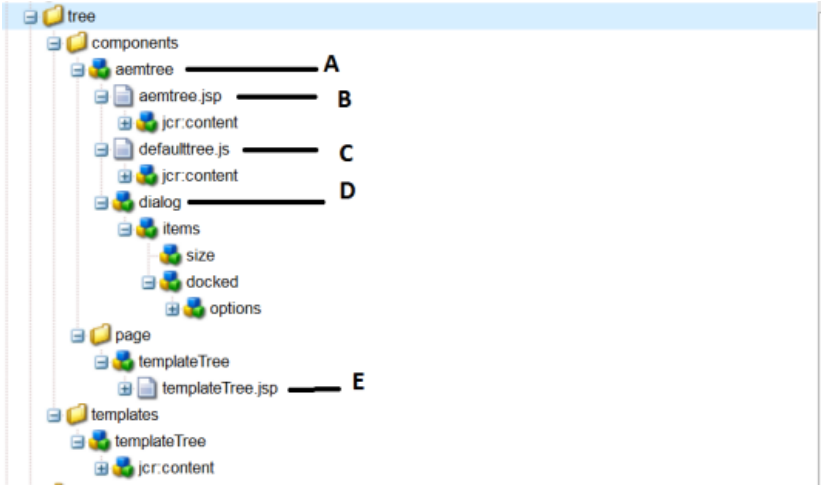
You can create an Adobe Experience Manager (AEM) custom component that contains a `TreePanel` xtype object that displays tree-structured data, such as the content of the AEM JCR. The data that is displayed within the `TreePanel` can be expanded or collapsed by clicking on it. The `TreePanel` is an instance of `CQ.Ext.tree.TreePanel`. For information, see CQ.Ext.tree.TreePanel.

You can develop the `TreePanel` component to invoke a CQ Sling Servlet. The Sling Servlet returns JSON encoded data that is used to populate the TreePanel with JCR data, as shown in the following illustration.



An AEM TreePanel component displaying AEM JCR data returned by a Sling Servlet

The following illustration represents the AEM project files that are created in this development article.



Project files created in this development article

The following table describes the project files.

Section	Description
A	The AEM component that uses the TreePanel xtype.
B	The component's main JSP file.
C	A JS file that contains application logic that creates a TreePanel object. This file also contains application logic that invokes the Sling Servlet that returns JSON data used to populate the TreePanel.
D	The dialog that the TreePanel component uses.
E	A page component that allows you to view the sidekick and drag the TreePanel widget onto the CQ page.

This article discusses how to use Maven to develop the Sling Servlet that retrieves JCR data and encodes it to JSON data that populates the `TreePanel`. It also discusses how to deploy the servlet and then how to create the `TreePanel` component that invokes the servlet.

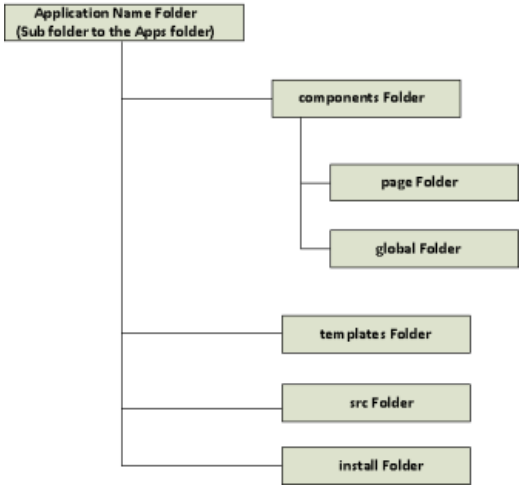
To create the `TreePanel` component that invokes the AEM Sling Servlet, perform these tasks:

1. Create an Adobe CQ application folder structure.
2. Create a template on which the page component is based.
3. Create a render component that uses the template.
4. Setup Maven in your development environment.
5. Create an Adobe CQ archetype project.
6. Add Java files that represent the Sling Servet to the Maven project.
7. Modify the Maven POM file.
8. Build the OSGi bundle using Maven.
9. Deploy the bundle to Adobe CQ.
10. Create a TreePanel component.
11. Add a dialog to the TreePanel component.
12. Add JavaScript code to the component files.
13. Create a CQ web page that uses the TreePanel component.

Create a CQ application folder structure

To the top

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



An AEM application file structure

The following describes each application folder:

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

To create an application folder structure:

1. To view the CQ welcome page, enter the URL http://[host name]:[port] into a web browser. For example, http://localhost:4502.
2. Select CRXDE Lite.
3. Right-click the apps folder (or the parent folder), select Create, Create Folder.
4. Enter the folder name into the Create Folder dialog box. Enter *tree*.
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.

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 *templateTree*.
- **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 *tree/components/page/templateTree*.
- **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 the page component based on 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.

3. Right-click /apps/tree/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 *templateTree*.
- **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 templateTree.jsp located at: /apps/tree/components/page/templateTree/templateTree.jsp.

8. Enter the following HTML code.

```
<html>
<%@include file="/libs/foundation/global.jsp" %>
<cq:include script="/libs/wcm/core/components/init/init.jsp"/>
<body>
<h1>Here is where the TreePanel component will go</h1>
<cq:include path="par" resourceType="foundation/components/parsys" />
</body>
</html>
```

Setup Maven in your development environment

To the top

You can use Maven to build an OSGi bundle that contains the Sling Servlet that is invoked by the TreePanel widget. 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 setti
   | the parameter in question.
   |
   | Default: true
  <interactiveMode>true</interactiveMode>
  -->

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

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

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

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

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

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

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

  <!-- profiles
   | This is a list of profiles which can be activated in a variety of ways, ar
   | the build process. Profiles provided in the settings.xml are intended to p
```

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

```

```
<profile>
                <id>adobe-public</id>
                <activation>
                    <activeByDefault>true</activeByDefault>
                </activation>
                <repositories>
                  <repository>
                    <id>adobe</id>
                    <name>Nexus Proxy Repository</name>
                    <url>http://repo.adobe.com/nexus/content/groups/public/</ur
                    <layout>default</layout>
                  </repository>
                </repositories>
                <pluginRepositories>
                  <pluginRepository>
                    <id>adobe</id>
                    <name>Nexus Proxy Repository</name>
                    <url>http://repo.adobe.com/nexus/content/groups/public/</ur
                    <layout>default</layout>
                  </pluginRepository>
                </pluginRepositories>
            </profile>
</profiles>
  <!-- activeProfiles
   | List of profiles that are active for all builds.
   |
  <activeProfiles>
    <activeProfile>alwaysActiveProfile</activeProfile>
    <activeProfile>anotherAlwaysActiveProfile</activeProfile>
  </activeProfiles>
  -->
</settings>
```

Create an Adobe CQ archetype project

To the top

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

Default files created by the Maven archetype plugin

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

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

2. Run the following Maven command:

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

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

- **groupId**: com.adobe.cq.sling.ds

- **artifactId**: jsonServlet
- **version**: 1.0-SNAPSHOT
- **package**: com.adobe.cq.sling.ds
- **appsFolderName**: jsonServlet-training
- **artifactName**: JSON 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\jsonServlet. Run the following Maven command:

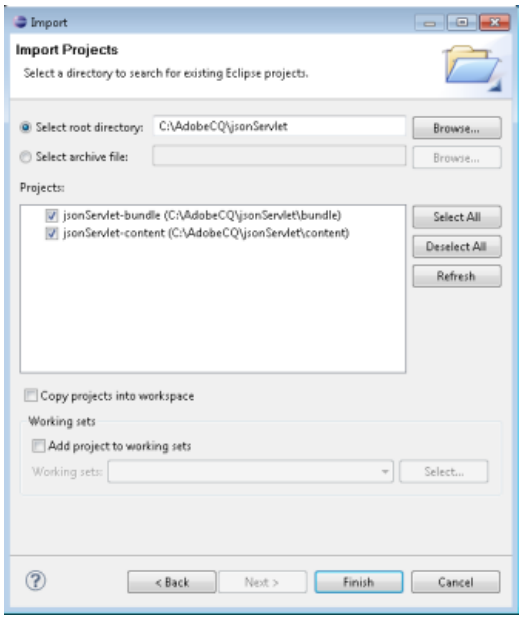
```
mvn eclipse:eclipse
```

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 `com.adobe.cq.sling.ds` package. The Java files that you create in this section use the Sling API to return a JCR data result set. The result set is encoded as JSON data and returned to the TreePanel widget. For information about the Sling API, see Sling API.

Add the following Java files to the `com.adobe.cq.sling.ds` package:

1. A Java servlet named `SiteAdminTreeServlet` that extends `SlingAllMethodsServlet`. This is the servlet that is invoked from the `TreePanel` widget by using a GET operation. For information, see SlingAllMethodsServlet.
2. A Java class named `IsSiteAdminPredicate` that implements the AbstractNodePredicate interface. This is a predicate that is used to filter JCR nodes. For information, see AbstractNodePredicate.

IsSiteAdminPredicate

The `IsSiteAdminPredicate` class represents extends `AbstractNodePredicate` and represents a custom predicate that is used by the `SiteAdmineTreeServlet`. This is an example of how you can use custom predicates within an AEM servlet. All this class does is check to see if the

JCR node is hidden. The following Java code represents this class.

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


import com.day.cq.commons.predicate.AbstractNodePredicate;
import com.day.cq.commons.jcr.JcrConstants;
import org.apache.commons.collections.Predicate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Properties;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Service;

import javax.jcr.Node;
import javax.jcr.RepositoryException;



public class IsSiteAdminPredicate extends AbstractNodePredicate implements Pred

    /** The is <code>hidden</code> property */
    private static final String HIDDEN = "hidden";
    /** Name of page content node */
    private static final String CONTENT_NODE = "jcr:content";

    /**
* {@inheritDoc}
*
* @return <code>true</code> if node must not be <code>hidden</code>
*/
    public boolean evaluate(Node node) throws RepositoryException {
        return !node.getName().startsWith(".") && node.isNodeType(JcrConstants.N
                !isHidden(node);
    }

    /**
* Check if node itself or its jcr:content node (if existent) is
* marked hidden.
*/
    private boolean isHidden(Node node) throws RepositoryException {
        boolean hidden = node.hasProperty(HIDDEN) && node.getProperty(HIDDEN).ge
        if(!hidden && node.hasNode(CONTENT_NODE)) {
            Node content = node.getNode(CONTENT_NODE);
            hidden = content.hasProperty(HIDDEN) && content.getProperty(HIDDEN)
        }
        return hidden;
    }
}
```

SiteAdminTreeServlet

The `SiteAdminTreeServlet` class uses the following Apache Felix SCR annotations:

- **@Component** – defines the class as a component.
- **@Service** - defines the service interface that is provided by the component.
- **@Properties** - defines properties used by the servlet. For example, notice that the URL of the servlet is specified: `/bin/tree` (this URL is specified in the `TreePanel` widget as shown later in this development article).
- **@Reference** - injects a `LiveRelationshipManager` into the component. For information, see LiveRelationshipManager.

Note: For information about these annotations, see http://felix.apache.org/documentation/subprojects/apache-felix-maven-scr-plugin/scr-annotations.html.

In this example, a `SiteAdminTreeJSONWriter` class is defined. This class is responsible for getting JCR data and encoding the data to JSON by using an `org.apache.sling.commons.json.io.JSONWriter` object. See JSONWriter.

The following Java code represents the servlet. Notice that the path parameter is passed to the servlet. A data set that represents the JCR node (and its children) is returned to the `TreePanel`. This data is displayed in the `TreePanel`.

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

import java.io.IOException;
import java.io.Writer;
import java.util.Calendar;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
```

```
import java.util.LinkedList;
import java.util.List;

import javax.jcr.Node;
import javax.jcr.RepositoryException;
import javax.servlet.ServletException;

import org.apache.commons.collections.Predicate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Properties;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.ReferenceCardinality;
import org.apache.felix.scr.annotations.ReferencePolicy;
import org.apache.felix.scr.annotations.Service;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.SlingHttpServletResponse;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.api.servlets.SlingAllMethodsServlet;
import org.apache.sling.commons.json.JSONException;
import org.apache.sling.commons.json.io.JSONWriter;

import com.day.cq.commons.LabeledResource;
import com.day.cq.dam.api.DamConstants;
import com.day.cq.replication.ReplicationStatus;
import com.adobe.cq.sling.ds.IsSiteAdminPredicate;
import com.day.cq.wcm.msm.api.LiveRelationshipManager;
import com.day.text.Text;

/**
 * <code>SiteAdminTreeServlet</code> implements a servlet that handles requests
 * from the SiteAdmin tree widget.
 */
@Component(metatype = false)
@Service
@Properties({
        @Property(name="sling.servlet.paths", value ={"/bin/tree"}, propertyPri
})
public class SiteAdminTreeServlet extends SlingAllMethodsServlet {

    private static final long serialVersionUID = -2600681470750906613L;

    /**
     * default path parameter name
     */
    public static final String PATH_PARAM = "path";

    /**
     * The parameter name that controls how many children are checked against
     * the predicate.
     */
    public static final String NUM_CHILDREN_CHECK = "ncc";

    @Reference (policy=ReferencePolicy.DYNAMIC, cardinality=ReferenceCardinali
    @SuppressWarnings({"UnusedDeclaration"})
    private LiveRelationshipManager relationshipManager;

    /**
     * {@inheritDoc}
     */
    @Override
    protected void doGet(SlingHttpServletRequest request,
                         SlingHttpServletResponse response)
            throws ServletException, IOException {

        if ("json".equals(request.getRequestPathInfo().getExtension())) {
            response.setContentType("application/json");
            response.setCharacterEncoding("utf-8");
            String path = request.getParameter(PATH_PARAM);

            int numChildrenCheck = 20;
            if (request.getParameter(NUM_CHILDREN_CHECK) != null) {
                try {
                    numChildrenCheck = Integer.parseInt(
                            request.getParameter(NUM_CHILDREN_CHECK));
                } catch (NumberFormatException e) {
                    // ignore and use default
                }
            }

            SiteAdminTreeJSONWriter w = new SiteAdminTreeJSONWriter(request.get

            w.write(response.getWriter(), path);
```

```
93          }
94      }
95
96      private static final class SiteAdminTreeJSONWriter {
97
98          private static final Predicate PREDICATE = new IsSiteAdminPredicate() {
99              @Override
100             public boolean evaluate(Node node) throws RepositoryException {
101                 return !node.isNodeType(DamConstants.NT_DAM_ASSET) && super.eva
102             }
103         };
104
105         private static final Predicate IS_SITE_ADMIN_PREDICATE = new IsSiteAdmi
106
107         /**
108          * internal resource resolver
109          */
110         private final ResourceResolver resolver;
111
112         /**
113          * The maximum number of children to check whether it matches the
114          * predicate.
115          */
116         private final int numChildrenCheck;
117
118         private LiveRelationshipManager relationshipManager;
119
120         /**
121          * Creates a new EXT tree json writer using the provided ResourceResol
122          *
123          * @param resolver            resource resolver
124          * @param numChildrenCheck    the maximum number of children to check
125          *                            whether it matches the predicate.
126          * @param relationshipManager Live copy relationship manager
127          */
128         public SiteAdminTreeJSONWriter(ResourceResolver resolver,
129                                        int numChildrenCheck,
130                                        LiveRelationshipManager relationshipMana
131             this.resolver = resolver;
132             this.numChildrenCheck = numChildrenCheck;
133             this.relationshipManager = relationshipManager;
134         }
135
136         /**
137          * Write the resource tree starting at <code>path</code> to the given w
138          * @param out  writer
139          * @param path start path
140          * @throws IOException if an I/O error occurs
141          */
142         public void write(Writer out, String path) throws IOException {
143             write(out, resolver.getResource(path));
144         }
145
146         /**
147          * Write the given resource tree to the given writer.
148          * @param out writer
149          * @param res start resource
150          * @throws IOException if an I/O error occurs
151          */
152         public void write(Writer out, Resource res) throws IOException {
153             if (res != null) {
154                 try {
155                     JSONWriter jw = new JSONWriter(out);
156                     boolean isOrderable = getIsOrderable(res);
157                     write(jw, getChildren(res), isOrderable, numChildrenCheck);
158                 } catch (JSONException e) {
159                     throw new IOException("Error while writing json " + e);
160                 }
161             } else {
162                 out.write("[]");
163             }
164         }
165
166         /**
167          * Write the given resource list as JSON array to the output.
168          *
169          * @param out              writer
170          * @param list             list of resources
171          * @param orderable        whether the list of resources is orderable.
172          * @param numChildrenCheck the maximum number of children to check
173          *                         whether it matches the predicate.
174          * @throws JSONException if a JSON error occurs
175          */
176         private void write(JSONWriter out,
```

```
                            List<Resource> list,
                            boolean orderable,
                            int numChildrenCheck)
                    throws JSONException {
                out.array();
                List<Resource> oList = orderable ? list : orderList(list);
                for (Resource resource : oList) {
                    out.object();

                    LabeledResource lr = resource.adaptTo(LabeledResource.class);
                    String name = Text.getName(resource.getPath());
                    out.key("name").value(name);
                    String text;

                    if (lr == null) {
                        text = name;
                    } else {
                        text = (lr.getTitle() == null)
                                ? name
                                : lr.getTitle();
                        if (lr.getDescription() != null) {
                            out.key("description").value(lr.getDescription());
                        }
                    }
                    if (text != null) {
                        text = text.replaceAll("<", "&lt;");
                    }
                    out.key("text").value(text);

                    out.key("type").value(resource.getResourceType());
                    int children = countChildren(resource, numChildrenCheck);
                    boolean hasChildren = children > 0;

                    // write CSS class information according to presence of childre
                    // this should probably be done via the 'type' attribute above
                    // the widget itself
                    out.key("cls").value(hasChildren ? "folder" : "file");
                    out.key("isLiveCopy").value(relationshipManager!=null
                            && relationshipManager.hasLiveRelationship(resource));

                    if (!hasChildren) {
                        out.key("leaf").value(true);
                    } else {
                        out.key("sub").value(children);
                    }

                    // if a user wants to delete an item from the tree, we must che
                    // thus, adding replication state information to the JSON resu
                    try {
                        writeReplicationState(out, resource);
                    } catch (Exception e) {
                        throw new JSONException("Unable to append replication state
                    }

                    out.endObject();
                }
                out.endArray();
            }

            /**
             * Returns a list of child resources that match {@link #PREDICATE}.
             *
             * @param res parent resource
             * @return list of child resources
             */
            private List<Resource> getChildren(Resource res) {
                List<Resource> children = new LinkedList<Resource>();
                for (Iterator<Resource> iter = resolver.listChildren(res); iter.has
                    Resource child = iter.next();
                    if (PREDICATE.evaluate(child)) {
                        children.add(child);
                    }
                }
                return children;
            }

            /**
             * Returns the number of children that have to be displayed in the tree
             * <p>If there are more than <code>numChildrenToCheck</code> children
             * (including the ones that will not be displayed) <code>numChildrenTo(
             * is returned to indicate that there could be more children.</p>
             *
             * @param res                parent resource
             * @param numChildrenToCheck The max number of children to check
```

```
261             * @return list of child resources
262             */
263            private int countChildren(Resource res, int numChildrenToCheck) {
264                int count = 0;
265                int totalCount = 0;
266                Iterator<Resource> iter = resolver.listChildren(res);
267                while (iter.hasNext() && totalCount <= numChildrenToCheck) {
268                    Resource child = iter.next();
269                    if (IS_SITE_ADMIN_PREDICATE.evaluate(child)) {
270                        // skip hidden (incl. "jcr:content") and non hierarchical r
271                        // see IsSiteAdminPredicate for the detailed conditions
272                        try {
273                            Node n = child.adaptTo(Node.class);
274                            if (!n.isNodeType(DamConstants.NT_DAM_ASSET)) {
275                                count++;
276                            }
277                            totalCount++;
278                        } catch (RepositoryException re) {
279                            // Ignored
280                        }
281                    }
282                }
283                if (totalCount == numChildrenToCheck + 1) {
284                    // avoid auto expand
285                    return totalCount;
286                }
287                return count;
288            }
289
290            private List<Resource> orderList(List<Resource> list) {
291                Collections.sort(list, new Comparator<Resource>() {
292                    public int compare(Resource r1, Resource r2) {
293                        return Text.getName(r1.getPath()).compareToIgnoreCase(Text.
294                    }
295                });
296                return list;
297            }
298
299            private boolean getIsOrderable(Resource resource) {
300                Node node = resource.adaptTo(Node.class);
301                if (node != null) {
302                    try {
303                        return node.getPrimaryNodeType().hasOrderableChildNodes();
304                    } catch (RepositoryException re) {
305                        // Ignored
306                    }
307                }
308                return false;
309            }
310
311            private void writeReplicationState(JSONWriter out, Resource resource) t
312                out.key("replication").object();
313                ReplicationStatus replicationStatus = resource.adaptTo(Replication
314                if (replicationStatus != null) {
315                    Calendar published = replicationStatus.getLastPublished();
316                    if (published != null) {
317                        out.key("published").value(published.getTimeInMillis());
318                    }
319                    if (replicationStatus.getLastReplicationAction() != null) {
320                        String action = replicationStatus.getLastReplicationAction(
321                        if (action != null && action.length() > 0) {
322                            out.key("action").value(action);
323                        }
324                    }
325                }
326                out.endObject();
327            }
328        }
329    }
```

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\jsonServlet\bundle, add the following dependencies.

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

The following XML represents this POM file.

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

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

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

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

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

```
85
86          <artifactId>org.osgi.core</artifactId>
87
88          <version>1.4.0</version>
89        </dependency>
90
91      <dependency>
92          <groupId>org.apache.sling</groupId>
93          <artifactId>org.apache.sling.commons.osgi</artifactId>
94          <version>2.2.0</version>
95      </dependency>
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>
106     <artifactId>jackrabbit-jcr-commons</artifactId>
107     <version>2.4.3</version>
108     </dependency>
109
110     <dependency>
111         <groupId>org.apache.sling</groupId>
112         <artifactId>org.apache.sling.jcr.api</artifactId>
113         <version>2.0.4</version>
114       </dependency>
115
116        <dependency>
117         <groupId>org.apache.sling</groupId>
118         <artifactId>org.apache.sling.api</artifactId>
119         <version>2.0.2-incubator</version>
120       </dependency>
121
122       <dependency>
123          <groupId>javax.jcr</groupId>
124          <artifactId>jcr</artifactId>
125          <version>2.0</version>
126       </dependency>
127
128 <dependency>
129     <groupId>javax.servlet</groupId>
130     <artifactId>servlet-api</artifactId>
131     <version>2.5</version>
132 </dependency>
133
134     <dependency>
135             <groupId>com.day.cq.wcm</groupId>
136             <artifactId>cq-wcm-api</artifactId>
137             <version>5.5.0</version>
138             <scope>provided</scope>
139         </dependency>
140
141         <dependency>
142             <groupId>com.day.cq</groupId>
143             <artifactId>cq-commons</artifactId>
144             <version>5.5.0</version>
145             <scope>provided</scope>
146         </dependency>
147
148        <dependency>
149             <groupId>com.day.commons</groupId>
150             <artifactId>day.commons.datasource.poolservice</artifactId>
151             <version>1.0.10</version>
152             <scope>provided</scope>
153            </dependency>
154
155            <dependency>
156             <groupId>com.day.cq.wcm</groupId>
157             <artifactId>cq-msm-api</artifactId>
158             <version>5.5.0</version>
159             <scope>provided</scope>
160         </dependency>
161
162         <dependency>
163             <groupId>com.day.commons</groupId>
164             <artifactId>day-commons-text</artifactId>
165             <version>1.1.8</version>
166             <scope>provided</scope>
167         </dependency>
168
```

```
        <dependency>
            <groupId>org.apache.sling</groupId>
            <artifactId>org.apache.sling.commons.json</artifactId>
            <version>2.0.6</version>
            <scope>provided</scope>
        </dependency>

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

        <dependency>
            <groupId>commons-collections</groupId>
            <artifactId>commons-collections</artifactId>
            <version>3.2.1</version>
            <scope>provided</scope>
        </dependency>

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

        <dependency>
            <groupId>com.adobe.granite</groupId>
            <artifactId>com.adobe.granite.replication.core</artifactId>
            <version>5.5.14</version>
            <scope>provided</scope>
        </dependency>


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



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

</project>
```

Build the OSGi bundle using Maven

To the top

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

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns:scr="http://www.osgi.org/xmlns/scr/v1.0.0">
    <scr:component enabled="true" name="com.adobe.cq.sling.ds.SimpleDSComponent'
        <implementation class="com.adobe.cq.sling.ds.SimpleDSComponent"/>
        <service servicefactory="false">
            <provide interface="java.lang.Runnable"/>
        </service>
        <property name="service.pid" value="com.adobe.cq.sling.ds.SimpleDSCompor
    </scr:component>
    <scr:component enabled="true" name="com.adobe.cq.sling.ds.SiteAdminTreeServ
        <implementation class="com.adobe.cq.sling.ds.SiteAdminTreeServlet"/>
```

```
        <service servicefactory="false">
            <provide interface="javax.servlet.Servlet"/>
            <provide interface="javax.servlet.ServletConfig"/>
            <provide interface="java.io.Serializable"/>
        </service>
        <property name="sling.servlet.paths" type="String" value="/bin/tree"/>
        <property name="service.pid" value="com.adobe.cq.sling.ds.SiteAdminTree
        <reference name="relationshipManager" interface="com.day.cq.wcm.msm.api
    </scr:component>
</components>
```

There are a couple of points to note about this XML file. First, notice that the implementation class element specifies `com.adobe.cq.sling.ds.SiteAdminTreeServlet`. 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 `relationshipManager`. Also notice that it's based on `com.day.cq.wcm.msm.api.LiveRelationshipManager`.

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

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

Deploy the bundle to Adobe CQ

To the top

Once you deploy the OSGi bundle, you can invoke the `SiteAdminTreeServlet` servlet from the xtype `TreePanel` widget (this is shown later in this development article). Also, you are able to see it in the Adobe CQ Apache Felix Web Conole.

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\jsonServlet\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.
9. If the status is not Active, check the CQ error.log for exceptions.

Create a component that uses a TreePanel xtype

To the top

Create the AEM component that uses a TreePanel. Perform these tasks using CRXDE Lite:

1. Right click on /apps/tree/components and then select New, Component.

2. Enter the following information into the Create Component dialog box:

- **Label**: The name of the component to create. Enter *aemtree*.
- **Title**: The title that is assigned to the component. Enter *AEM TreePanel component*.

- **Description**: The description that is assigned to the template. Enter *AEM TreePanel component*.
- **Super Resource Type**: Enter *foundation/components/parbase*.
- **Group**: The group in the side kick where the component appears. Enter *General*. (The aemtree component is located under the General heading in the sidekick.)
- **Allowed parents**: Enter **/*parsys*.

3. Click Ok.

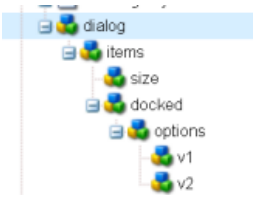
Note: The remaining part of this article talks about how to create the aemtree component that uses a `TreePanel` xtype. The aemtree.jsp file located at /apps/tree/components/aemtree.jsp is populated with JavaScript logic later in this development article.

Add a dialog to the TreePanel component

To the top

A dialog lets an author click on the component during design time and enter values that are used by the component. The component created in this development article lets a user enter values that influence the look of the TreePanel. For example, you can specify the width, in pixels, of the component.

The following illustration shows the JCR nodes that represent the dialog created in this section.



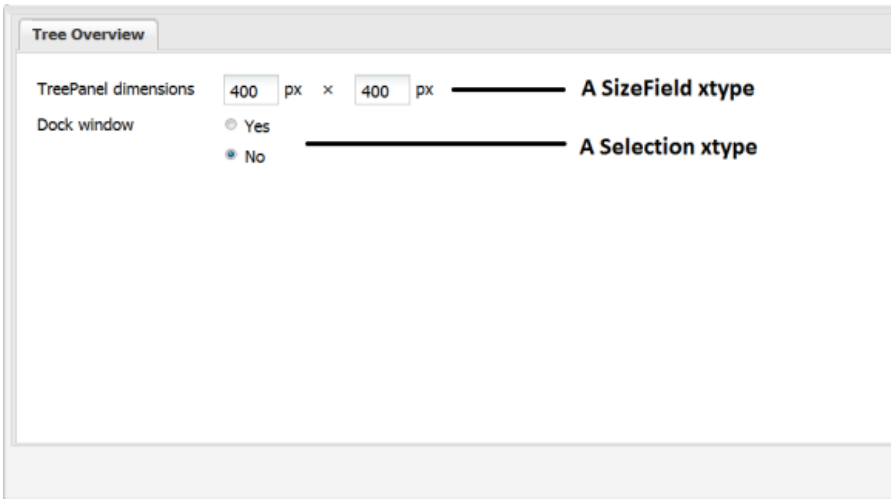
JCR nodes that represent the dialog for the TreePanel component

To add a dialog to the aemtree component, perform these tasks:

1. Select */apps/tree/components/aemtree*, right click and select Create, Create Dialog.
2. In the Title field, enter *aemtree*.
3. Click Ok.
4. Delete the tab1 node under */apps/tree/components/aemtree/dialog/items/items*.

Create the Overview tab

Create the first (and only) tab in the dialog titled *Tree Overview*. This dialog contains input controls that impact the TreePanel. The following illustration shows this tab in the CQ dialog.



The dialog that belongs to the aemgrid component

In the previous illustration, notice the TreePanel dimensions control. This control is based on a `sizetype` xtype control. A `sizetype` control lets the user enter the width and height for the grid. For information, see Class CQ.form.SizeField.

The Dock window control is based on a `selection` xtype. In this example, selecting the Yes option results in the TreePanel being docked. For information, see Class CQ.form.Selection.

To create the Overview tab, perform these tasks:

1. Click on the following node: */apps/tree/components/aemtree/dialog/items*.

2. Right click and select Create, Create Node. Enter the following values:

- **Name**: size
- **Type**: cq:Widget

3. Select the */apps/tree/components/aemtree/dialog/items/size* node.

4. Add the following properties to the `size` node.

Name	Type	Value	Description
fieldLabel	String	TreePanel dimensions	Specifies the label for the control.
xtype	String	sizefield	Specifies the data type for the control.

5. Click on the following node: */apps/tree/components/aemtree/dialog/items*.

6. Right click and select Create, Create Node. Enter the following values:

- **Name**: docked
- **Type**: cq:Widget

7. Select the */apps/tree/components/aemtree/dialog/items/docked* node.

8. Add the following properties to the `docked` node.

Name	Type	Value	Description
fieldLabel	String	Dock window	Specifies the label for the control.
defaultValue	String	false	Specfies which option is checked.
name	String	./name	Specifies the name of the control.
type	String	radio	Specifies the type of selection. The value radio specifies a radio button.
xtype	String	selection	Specifies the xtype of the control.

8. Click on the following node: */apps/tree/components/aemtree/dialog/items/docked*.

9. Right click and select Create, Create Node. Enter the following values:

- **Name**: options
- **Type**: cq:WidgetCollection

10. Click on the following node: */apps/tree/components/aemtree/dialog/items/docked/options*.

11. Right click and select Create, Create Node. Enter the following values:

- **Name**: v1
- **Type**: nt:unstructured.

12. Add the following properties to the `V1` node.

Name	Type	Value	Description
text	Sting	yes	The text that is displayed.
value	String	true	The value that corresponds to this option.

13. Click on the following node: /apps/tree/components/aemtree/dialog/items/docked/options.

14. Right click and select Create, Create Node. Enter the following values:

- **Name**: v2
- **Type**: nt:unstructured.

15. Add the following properties to the `V2` node.

Name	Type	Value	Description

text	Sting	no	The text that is displayed.
value	String	false	The value that corresponds to this option.

Add JavaScript code to the component files

To the top

To develop an AEM component that uses a TreePanel xtype, develop these files:

- **defaulttree.js**: contains JavaScript logic that creates a `CQ.Ext.tree.TreePanel` instance. Also contains application logic that invokes the Sling Servlet located at `/bin/tree`.
- **aemtree.jsp**: defines JavaScript logic that defines the behaviour of the `TreePanel` component.

defaulttree.js

The *defaulttree.js* file contains application logic that defines `TreePanel` object and invokes the Sling Servlet that populates the TreePanel with JCR data. When invoking a Sling Servlet from a `TreePanel` xtype widget, you assign the `loader` property with a valid `CQ.Ext.tree.TreeLoader` instance. A `CQ.Ext.tree.TreeLoader` provides for lazy loading of an `CQ.Ext.tree.TreeNode`'s child nodes from a specified URL. For more information, see CQ.Ext.tree.TreeLoader.

Assign a `CQ.HTTP.externalize` to the `TreeLoader` object's `dataUrl` property and specify the URL to the sling servlet. For example:

```
loader: {
    dataUrl:CQ.HTTP.externalize("/bin/tree.json"),
```

Another loader property that you set is the `requestMethod` property. This property specifies the HTTP request method for loading data. You can assign `GET` to this property. Also, define the `beforeload` event that is fired before a network request.

The following code shows the loader that populates the `TreePanel` instance.

```
loader: {
                dataUrl:CQ.HTTP.externalize("/bin/tree.json"),
                requestMethod:"GET",
                // request params
                baseParams: {
                    "_charset_": "utf-8"
                },
                // change request params before loading
                listeners: {
                    beforeload: function(loader, node) {

                        var myPath = node.getPath();
                        this.baseParams.path = myPath ;
                    }
                },
                // attributes for all nodes created by the loader
                baseAttrs: {
                    singleClickExpand:true
                    //iconCls:"folder"
                }
            },
```

The following JavaScript code represents the *defaulttree.js* file that creates a `CQ.Ext.tree.TreePanel` object and invokes the sling servlet.

```
//-----------------------------------------------------------------------------
// getTreePanel returns a TreePanel

function getTreePanel() {



    var treePanel = new CQ.Ext.tree.TreePanel({
           border:false,
           // CQ.Ext.tree.TreeLoader config
           loader: {
               dataUrl:CQ.HTTP.externalize("/bin/tree.json"),
               requestMethod:"GET",
               // request params
               baseParams: {
                   "_charset_": "utf-8"
               },
               // change request params before loading
               listeners: {
                   beforeload: function(loader, node) {
```

```
                        var myPath = node.getPath();
                        this.baseParams.path = myPath ;
                    }
                },
                // attributes for all nodes created by the loader
                baseAttrs: {
                    singleClickExpand:true
                    //iconCls:"folder"
                }
            },

            // CQ.Ext.tree.TreeNode config
            root: {
                nodeType:"async",
                text:CQ.I18n.getMessage("TreePanel Example"),
                name:"/apps/tree",
                expanded:true
            }
        });
    return treePanel ;
}
```

aemtree.jsp

The *aemtree.jsp* contains application logic that controls the behaviour of the aemtree component. First, the values defined in the dialog are obtained by using the `properties.get` method, as shown in the following code example.

```
// load properties defined by the aemgrid dialog
int width = properties.get("width", 600);
int height = properties.get("height", 300);
boolean docked = properties.get("docked", false);
```

These values control the behaviour of the TreePanel. The `width` and `height` values specify its size. The `docked` value specifies whether it's docked.

The following method, named `getTreePanel`, returns a `CQ.Ext.tree.TreePanel` instance to a variable named `treePanel`. This method is defined in the *defaulttree.js* file.

```
var treePanel = getTreePanel();
```

To ensure that the *defaulttree.js* file is referenced, the following `script` tag is included:

```
<script type="text/javascript"
src="/apps/tree/components/aemtree/defaulttree.js"></script>
```

To display a `CQ.Ext.tree.TreePanel` instance, create a `CQ.Ext.Window` instance. The `width`, `height`, and `docked` variables are used to create a `CQ.Ext.Window` instance. This is how the values specified in the component's dialog are hooked into the TreePanel component.

Aslo notice that the `treePanel` variable is used, as shown in the following code example.

```
tree= new CQ.Ext.Window({
            id:"<%= node.getName() %>-grid",
            title:"TreePanel Example",
            layout:"fit",
            hidden:true,
            collapsible:true,
            renderTo:"CQ",
            width:<%= width %>,
            height:<%= height %>,
            x:<%= docked ? 0 : 220 %>,
            y:<%= docked ? 0 : 200 %>,
            closeAction:'hide',
            items: treePanel,
            listeners: {
                beforeshow: function() {
                    gridPanel.getStore().load();
                }
            },
            buttons:[{
                text:"Close",
                handler: function() {
                    tree.hide();
                }
            },{
                text:"Dock",
                handler: function() {
                    tree.setPosition(0,0);
                }
            }]
        });
```

Notice that this code defines a window for the data grid. The `items` property is assigned the `treePanel`, which stores an instance of `CQ.Ext.tree.TreePanel`. This is how a TreePanel is associated with the window that is defined by using a `CQ.Ext.Window` data type.

Notice that two buttons are defined.

```
buttons:[{
text:"Close",
handler: function() {
tree.hide();
}
},{
text:"Dock",
handler: function() {
tree.setPosition(0,0);
}
}]
```

The first button closes the TreePanel when the button is clicked. Likewise, the Dock window sets the TreePanel to position 0. Using methods that belong to `CQ.Ext.tree.TreePanel`, you can further control the TreePanel's behaviour.

The following code represents the *aemtree.jsp* file.

```
<%@include file="/libs/foundation/global.jsp"%><%

    Node node22 = resource.adaptTo(Node.class);

    // load properties
    int width = properties.get("width", 200);
    int height = properties.get("height", 300);
    boolean docked = properties.get("docked", false);
%>
<h3>Exercise 4: Tree Overview</h3><%
%><p>Learn about:
    <ul>
        <li>The tree config</li>
        <li>The expected data format:<code><pre>
[{
    name: "lorem",
    text: 'A leaf Node',
    leaf: true
},{
    name: "ipsum",
    text: 'A folder Node',
    children: [{
        name: "dolor",
        text: 'A child Node'
    }]
}]
</pre></code>
</li>
    </ul>
</p>
<script type="text/javascript" src="/apps/tree/components/aemtree/defaulttree.js
<script type="text/javascript">


    var tree = CQ.Ext.getCmp("<%= node22.getName() %>-tree");
    if (!tree) {
        var treePanel = getTreePanel();

        tree = new CQ.Ext.Window({
            id:"<%= node22.getName() %>-tree",
            title:"Tree Example",
            hidden:true,
            layout:"fit",
            collapsible:true,
            renderTo:"CQ",
            width:<%= width %>,
            height:<%= height %>,
            x:<%= docked ? 0 : 500 %>,
            y:<%= docked ? 0 : 100 %>,
            closeAction:'hide',
            items: treePanel,
            buttons:[{
                text:"Close",
                handler: function() {
                    tree.hide();
                }
            },{
                text:"Dock",
```

```
                handler: function() {
                    tree.setPosition(0,0);
                }
            }]
        });
        tree.show();
    } else {
        tree.setWidth(<%= width %>);
        tree.setHeight(<%= height %>);
        tree.setPosition(<%= docked ? 0 : 500 %>,<%= docked ? 0 : 100 %>);
        tree.show();
    }

</script>
```

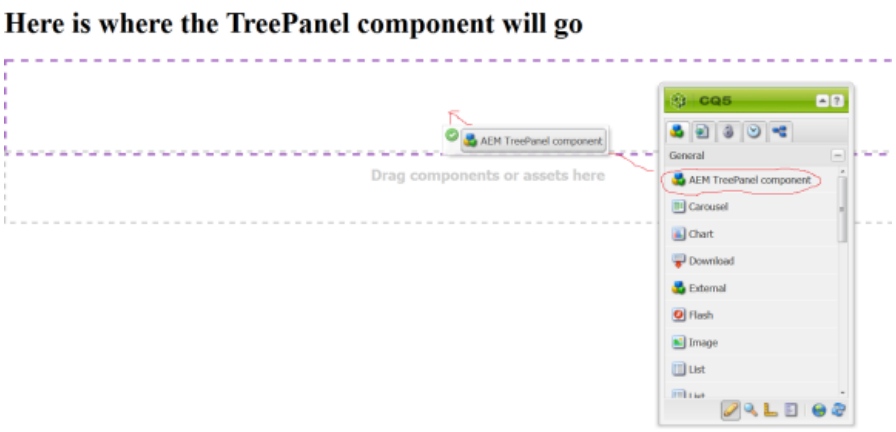
Add the files to the project

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/tree/components/aemtree/aemtree.jsp*.
4. Replace the JSP code with the new code shown in this section.
5. Select apps/tree/components/aemtree. Add a new file named *defaulttree.js*.
6. Add the code shown in this section to this file.
7. Click Save All.

Create a CQ web page that uses the aemtree component

To the top

The final task is to create a site that contains a page that is based on the templateTree (the template created earlier in this development article). This CQ page will let you select the aemtree that you just created from the CQ sidekick, as shown in the following illustration.



The CQ sidekick displaying the aemtree component that is created in this development article

1. Go to the CQ welcome page at http://[host name]:[port]; for example, http://localhost:4502.
2. Select Websites.
3. From the left hand pane, select Websites.
4. Select 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 templateTree 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.
8. Open the new page that you created by double-clicking it in the right pane. The new page opens in a web browser. Drag the aemtree component from the sidekick under the General category. (If the sidekick is empty, click the Design button near the bottom and click the Edit button. Select General from the list that appears. This will populate the sidekick.)
9. Double click on the aemtree component. Enter values into the dialog. Once done, the TreePanel appears with JCR data that is returned from the servlet.

See also

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