Adobe CQ Help / Creating a custom CQ component that uses a dialog grid

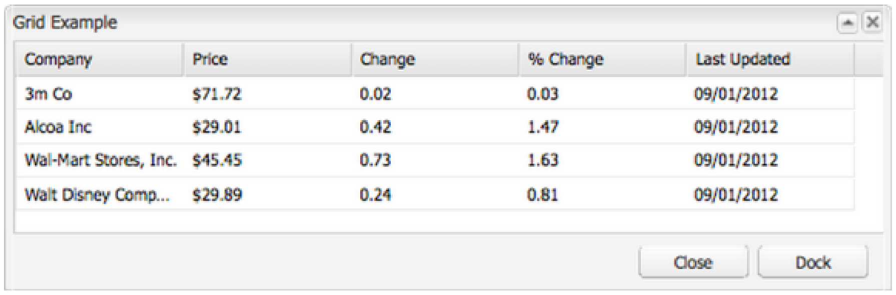
Article summary

Summary	Discusses how to develop a CQ component that uses a data grid based on a `CQ.Ext.grid.GridPanel`. Also discusses how to populate the grid by using a `CQ.Ext.data.Store` instance.
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 grid control that displays data. The custom component contains a dialog that renders a grid. The data grid is an instance of `CQ.Ext.grid.GridPanel`. For information, see CQ.Ext.grid.GridPanel.

The grid lets an AEM author open the component's dialog and view data, as shown in the following illustration.

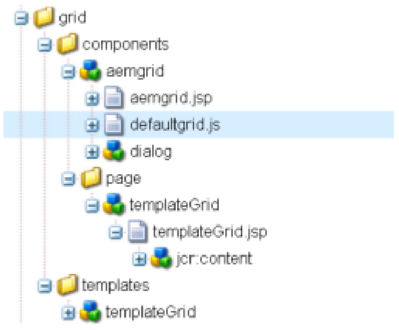

Grid Example

Company	Price	Change	% Change	Last Updated
3m Co	$71.72	0.02	0.03	09/01/2012
Alcoa Inc	$29.01	0.42	1.47	09/01/2012
Wal-Mart Stores, Inc.	$45.45	0.73	1.63	09/01/2012
Walt Disney Comp...	$29.89	0.24	0.81	09/01/2012

A grid control located in an AEM component's dialog

You can populate the grid by using a `CQ.Ext.data.Store` instance. You can define a JavaScript method that dynamically creates a `CQ.Ext.data.Store` instance to populate the grid with data obtained at run-time. (The data values in this example are hard coded for simplicity.)

The following illustration shows the project files created in this development article.


```
grid
  components
    aemgrid
      aemgrid.jsp
      defaultgrid.js
      dialog
    page
      templateGrid
        templateGrid.jsp
          jcr:content
  templates
    templateGrid
```

Project files created in this development article

The aemgrid is the name of the AEM component that contains a data grid. The *aemgrid.jsp* and *defaultgrid.js* files contain JavaScript application logic that is responsible for defining a `CQ.Ext.grid.GridPanel` instance. In addition, a `CQ.Ext.data.Store` instance is also created to populate the grid with data. These files are created later in this development article.

This development article steps you through how to build an AEM component that uses a `CQ.Ext.grid.GridPanel` instance. Perform these steps:

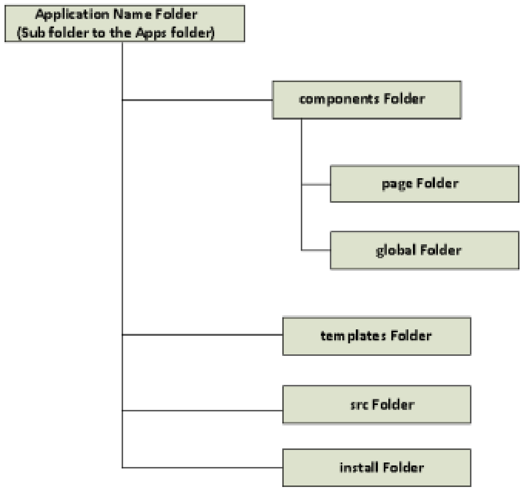
1. Create a CQ application folder structure.

2. Create a template.
3. Create the page component based on the template.
4. Create a component that uses a grid.
5. Add a dialog to the grid component.
6. Add JavaScript code to the component files.
7. Create a CQ web page that uses the new 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 *grid*.
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 *templateGrid*.
- **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 *grid/components/page/templateGrid*.
- **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/grid/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 *templateGrid*.
- **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 templateHook.jsp located at: /apps/grid/components/page/templateGrid/templateGrid.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 component will go</h1>
<cq:include path="par" resourceType="foundation/components/parsys" />
</body>
</html>
```

Create a component that uses a grid

To the top

After you setup the AEM folder structure, create the AEM component that uses a grid. Perform these tasks using CRXDE Lite:

1. Right click on /apps/grid/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 *aemgrid*.
- **Title**: The title that is assigned to the component. Enter *AEM Grid component*.
- **Description**: The description that is assigned to the template. Enter *AEM Grid component*.
- **Super Resource Type**: Enter *foundation/components/parbase*.

- **Group**: The group in the side kick where the component appears. Enter *General*. (The aemgrid component is located under the General heading in the sidekick.)
- **Allowed parents**: Enter **/*parsys*.

3. Click Ok.

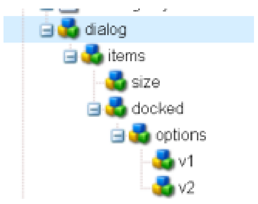
Note: The remaining part of this article talks about how to create the aemgrid component that uses a grid. The aemgrid.jsp file located at /apps/grid/components/aemgrid.jsp is populated with JavaScript logic later in this development article.

Add a dialog to the AEM grid 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 grid. For example, the user can specify the width, in pixels, of the grid.

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



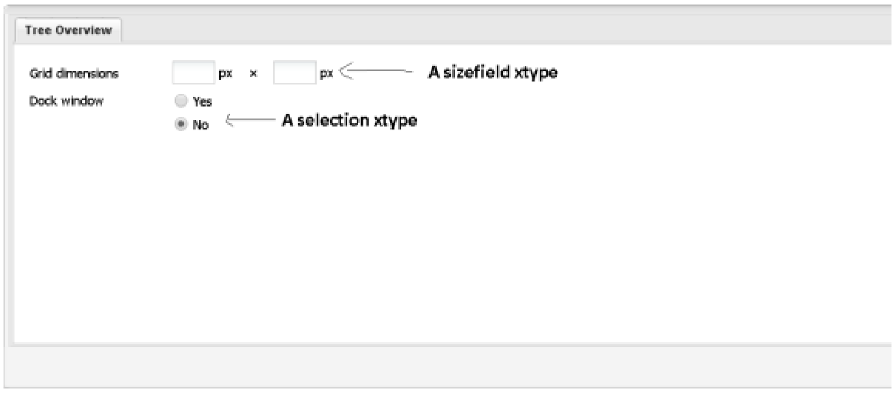
JCR nodes that represent the dialog for the aemgrid component

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

1. Select */apps/grid/components/aemgrid*, right click and select Create, Create Dialog.
2. In the Title field, enter *aemgrid*.
3. Click Ok.
4. Delete the tab1 node under */apps/grid/components/aemgrid/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 grid. The following illustration shows this tab in the CQ dialog.



The dialog that belongs to the aemgrid component

In the previous illustration, notice the Grid 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 grid being docked. For information, see Class CQ.form.Selection.

To create the Overview tab, perform these tasks:

1. Click on the following node: */apps/grid/components/aemgrid/dialog/items*.

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

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

3. Select the */apps/grid/components/aemgrid/dialog/items/size* node.

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

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

5. Click on the following node: */apps/grid/components/aemgrid/dialog/items*.
6. Right click and select Create, Create Node. Enter the following values:

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

7. Select the */apps/grid/components/aemgrid/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/grid/components/aemgrid/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/grid/components/aemgrid/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/grid/components/aemgrid/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 grid, you develop these files:

- **defaultgrid.js**: contains JavaScript logic that creates a `CQ.Ext.grid.GridPanel` instance.

- **aemgrid.jsp**: defines JavaScript logic that defines the behaviour of the grid.

defaultgrid.js

The *defaultgrid.js* file contains application logic that defines both the data grid and the data source that populates the grid with data. The following JavaScript code examples defines a `CQ.Ext.grid.GridPanel` instance.

```
var gridPanel = new CQ.Ext.grid.GridPanel({
        border:true,
        store: store,
        columns: [
            {
                id:'company',
                header: "Company",
                dataIndex: 'company'
            },{
                header: "Price",
                renderer: CQ.Ext.util.Format.usMoney,
                dataIndex: 'price'
            },{
                header: "Change",
                dataIndex: 'change'
            },{
                header: "% Change",
                dataIndex: 'pctChange'
            },{
                header: "Last Updated",
                renderer: CQ.Ext.util.Format.dateRenderer('m/d/Y'),
                dataIndex: 'lastChange'
            }
        ],
        viewConfig: {
            forceFit: true
        },
        sm: new CQ.Ext.grid.RowSelectionModel({singleSelect:true}),
        iconCls:'icon-grid'
    });
```

Notice the `store` populates the grid with data. You set the `store` property with a `CQ.Ext.data.Store` instance. The following JavaScript code example creates a `Store` instance.

```
//myData defines data that is displayed in the data grid
var myData = [
      ['3m Co',71.72,0.02,0.03,'9/1 12:00am'],
      ['Alcoa Inc',29.01,0.42,1.47,'9/1 12:00am'],
      ['Wal-Mart Stores, Inc.',45.45,0.73,1.63,'9/1 12:00am'],
      ['Walt Disney Company (The) (Holding Company)',29.89,0.24,0.81,'9/1 12:00
    ];

//The Data model that is used to populate the grid. It uses myData as the data
    var store = new CQ.Ext.data.Store({
        proxy: new CQ.Ext.data.MemoryProxy(myData),
        reader: new CQ.Ext.data.ArrayReader({}, [
            {name: 'company'},
            {name: 'price', type: 'float'},
            {name: 'change', type: 'float'},
            {name: 'pctChange', type: 'float'},
            {name: 'lastChange', type: 'date', dateFormat: 'n/j h:ia'}
        ])
    });
```

The following code represents the *defaultgrid.js* that defines the data grid that is used in the AEM component.

```
//-------------------------------------------------------------------------------
// default grid showing basic config

function getGridPanel() {
    var myData = [
      ['3m Co',71.72,0.02,0.03,'9/1 12:00am'],
      ['Alcoa Inc',29.01,0.42,1.47,'9/1 12:00am'],
      ['Wal-Mart Stores, Inc.',45.45,0.73,1.63,'9/1 12:00am'],
      ['Walt Disney Company (The) (Holding Company)',29.89,0.24,0.81,'9/1 12:00
    ];


    //The Data model that is used to populate the grid
    var store = new CQ.Ext.data.Store({
        proxy: new CQ.Ext.data.MemoryProxy(myData),
        reader: new CQ.Ext.data.ArrayReader({}, [
            {name: 'company'},
            {name: 'price', type: 'float'},
            {name: 'change', type: 'float'},
            {name: 'pctChange', type: 'float'},
            {name: 'lastChange', type: 'date', dateFormat: 'n/j h:ia'}
        ])
    });

    //load the data
    store.load();

    var gridPanel = new CQ.Ext.grid.GridPanel({
        border:true,
        store: store,
        columns: [
            {
                id:'company',
                header: "Company",
                dataIndex: 'company'
            },{
                header: "Price",
                renderer: CQ.Ext.util.Format.usMoney,
                dataIndex: 'price'
            },{
                header: "Change",
                dataIndex: 'change'
            },{
                header: "% Change",
                dataIndex: 'pctChange'
            },{
                header: "Last Updated",
                renderer: CQ.Ext.util.Format.dateRenderer('m/d/Y'),
                dataIndex: 'lastChange'
            }
        ],
        viewConfig: {
            forceFit: true
        },
        sm: new CQ.Ext.grid.RowSelectionModel({singleSelect:true}),
        iconCls:'icon-grid'
    });
    gridPanel.getColumnModel().defaultSortable = true;
    return gridPanel;
}
```

aemgrid.jsp

The *aemgrid.jsp* contains application logic that controls the behaviour of the aemgrid component, including the grid. 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 data grid. The `width` and `height` values specify its size. The `docked` value specifies whether it's docked.

The following method, named `getGridPanel`, returns a `CQ.Ext.grid.GridPanel` instance to a variable named `gridPanel`. This method is defined in the *defaultgrid.js* file.

```
var gridPanel = getGridPanel();
```

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

```
<script type="text/javascript"
src="/apps/grid/components/aemgrid/defaultgrid.js"></script>
```

To display a `CQ.Ext.grid.GridPanel` 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 data grid.

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

```
grid = new CQ.Ext.Window({
            id:"<%= node.getName() %>-grid",
            title:"Grid Example 24",
            layout:"fit",
            hidden:true,
            collapsible:true,
            renderTo:"CQ",
            width:<%= width %>,
            height:<%= height %>,
            x:<%= docked ? 0 : 220 %>,
            y:<%= docked ? 0 : 200 %>,
            closeAction:'hide',
            items: gridPanel,
            listeners: {
                beforeshow: function() {
                    gridPanel.getStore().load();
                }
            },
            buttons:[{
                text:"Close",
                handler: function() {
                    grid.hide();
                }
            },{
                text:"Dock",
                handler: function() {
                    grid.setPosition(0,0);
                }
            }]
        });
```

Notice that this code defines a window for the data grid. The `items` property is assigned the `gridPanel`, which stores an instance of `CQ.Ext.grid.GridPanel`. This is how a data grid 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() {
grid.hide();
}
},{
text:"Dock",
handler: function() {
grid.setPosition(0,0);
}
}]
```

The first button closes the data grid when the button is clicked. Likewise, the Dock window sets the data grid to positition 0. Using methods that belong to `CQ.Ext.grid.GridPanel`, you can further control the behaviour of the grid.

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

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

    Node node = resource.adaptTo(Node.class);
    // load properties
    int width = properties.get("width", 600);
    int height = properties.get("height", 300);
    boolean docked = properties.get("docked", false);

%>
<h3>Exercise 5: Grid Overview</h3><%
%><p>Learn about:
    <ul>
        <li>The grid config</li>
        <li>The store config</li>
        <li>The expected data format
<code><pre>
{
    results: 3,
    root: [
        { 'id': 1, 'firstname': 'Bill', occupation: 'Gardener' },
        { 'id': 2, 'firstname': 'Frank', occupation: 'Programmer' },
        { 'id': 3, 'firstname': 'Ben' , occupation: 'Horticulturalist' }
    ]
}
</pre></code>
        The above data would require the following store config to consume:
<code><pre>
var store = new CQ.Ext.data.JsonStore({
    reader: {
        totalProperty:"results",
        root:"root",
        id:"id"
    },
    url:"/content/test.json"
});
</pre></code>
        </li>
    </ul>
</p>
 <script type="text/javascript" src="/apps/grid/components/aemgrid/defaultgrid.
 <script type="text/javascript">

    var grid = CQ.Ext.getCmp("<%= node.getName() %>-grid");
    if (!grid) {

        var gridPanel = getGridPanel();

        grid = new CQ.Ext.Window({
            id:"<%= node.getName() %>-grid",
            title:"Grid Example 24",
            layout:"fit",
            hidden:true,
            collapsible:true,
            renderTo:"CQ",
            width:<%= width %>,
            height:<%= height %>,
            x:<%= docked ? 0 : 220 %>,
            y:<%= docked ? 0 : 200 %>,
            closeAction:'hide',
            items: gridPanel,
            listeners: {
                beforeshow: function() {
                    gridPanel.getStore().load();
                }
            },
            buttons:[{
                text:"Close",
                handler: function() {
                    grid.hide();
                }
            },{
                text:"Dock",
                handler: function() {
                    grid.setPosition(0,0);
                }
            }]
        });
        grid.show();
    } else {
        grid.setWidth(<%= width %>);
        grid.setHeight(<%= height %>);
        grid.setPosition(<%= docked ? 0 : 500 %>,<%= docked ? 0 : 100 %>);
        grid.show();
    }

</script>
```

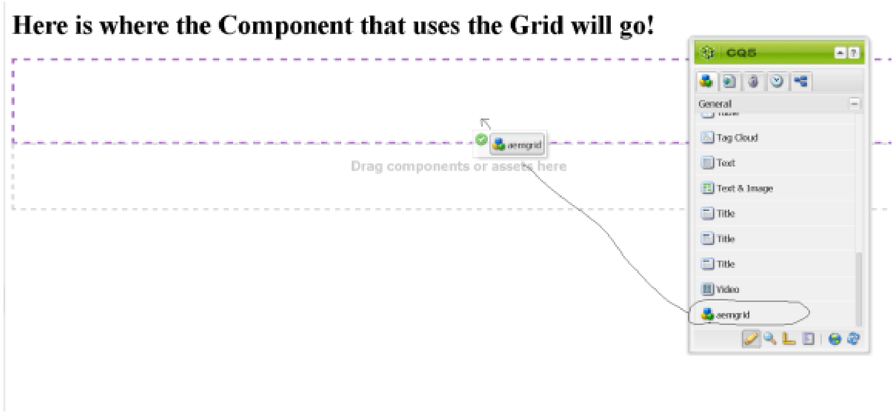
Add the files to the project

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

Create a CQ web page that uses the aemgrid component

To the top

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


Here is where the Component that uses the Grid will go!
Drag components or assets here
aemgrid
CQ5
General
Tag Cloud
Text
Text & Image
Title
Title
Title
Video
aemgrid

The CQ sidekick displaying the aemgrid 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 templateGrid 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 aemgrid component from the sidekick under the General category.
9. Double click on the aemgrid component. Enter values into the dialog. Once done, the data grid is displayed.