

Adobe CQ Help /

Creating a custom CQ component that uses an editable dialog grid

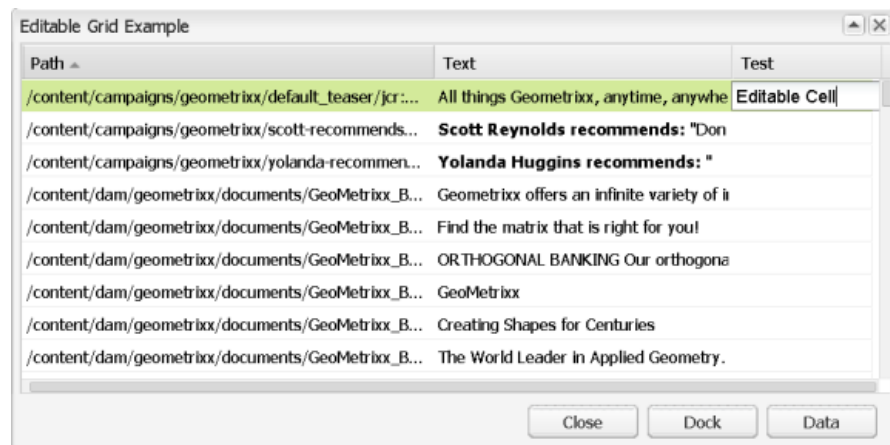
Article summary

Summary	<p>Discusses the following points:</p> <ul style="list-style-type: none"> • how to develop a CQ component that uses an editable data grid based on a <code>CQ.Ext.grid.EditorGridPanel</code>. An editable grid lets AEM authors modify data located in the grid's cells. • how to setup the grid's column by using a <code>CQ.Ext.grid.ColumnModel</code> instance. • how to populate the grid by using an <code>Ext.data.GroupingStore</code> instance. • how to use a <code>CQ.Ext.data.HttpProxy</code> instance to retrieve data from CQ. • how to use a <code>CQ.Ext.data.JsonReader</code> to store grid data. • how to use grid event handlers such as the method that is fired when a change to a grid cell occurs. • how to get grid data by iterating through the grid and retrieving <code>CQ.Ext.data.Record</code> objects.
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 an editable grid control that displays data. The data within the grid can be edited by clicking on a cell. The data grid is an instance of `CQ.Ext.grid.EditorGridPanel`. For information, see [CQ.Ext.grid.EditorGridPanel](#).

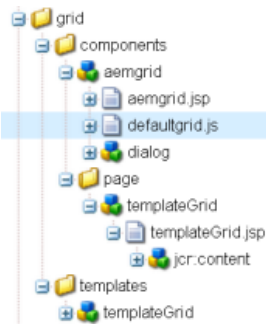
The grid lets an AEM author open the component's dialog and view data, as shown in the following illustration.



A grid control located in an AEM component's dialog with editable cells

You can configure the grid's columns by using a `CQ.Ext.grid.ColumnModel` instance. You define a JavaScript object of this type and use that to set the columns that are displayed in the grid. (This is shown later in this development article.)

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



Project files created in this development article

The `aemgrid` is the name of the AEM component that contains an editable data grid. The `aemgrid.jsp` and `defaultgrid.js` files contain JavaScript application logic that is responsible for defining a `CQ.Ext.grid.EditorGridPanel` instance (these files are created later in this development article).

When an AEM author modifies a cell's content, you can programmatically access the modified value. That is, you have access to the modified value as well as the original value. For example, assume that the value of a cell is *Cell1*. Next assume that the author changes the cell value to *Cell2*. You have access to both the original value (*Cell1*) and the modified value (*Cell2*) (this is shown later in this development article).

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

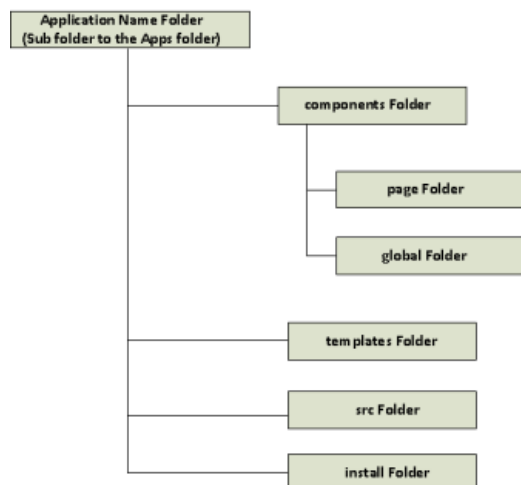
1. Create a CQ application folder structure.
2. Create a template.
3. Create the page component based on the template.
4. Create a component that uses a grid.
5. Add a dialog to the grid component.
6. Add JavaScript code to the component files.
7. Create a CQ web page that uses the new component.

Note: You can create an AEM component that uses a non-editable grid. This type of grid is based on `CQ.Ext.grid.GridPanel`. For information, see [Creating a custom CQ component that uses a dialog grid](#).

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 templateGrid.jsp located at:
/apps/grid/components/page/templateGrid/templateGrid.jsp.
 8. Enter the following HTML code.

```

1 <html>
2 <%@include file="/libs/foundation/global.jsp" %>
3 <cq:include script="/libs/wcm/core/components/init/init.jsp"/>
4 <body>
5 <h1>Here is where the component will go</h1>
6 <cq:include path="par" resourceType="foundation/components/parsys" />
7 </body>
8 </html>

```

Create a component that uses an editable grid

[To the top](#)

After you setup the AEM folder structure, create the AEM component that uses an editable 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.

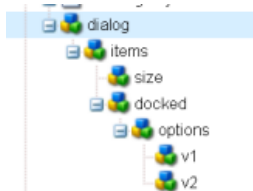
Note: The remaining part of this article talks about how to create the aemgrid component that uses a grid. The aemgrid.jsp file located at /apps/grid/components/aemgrid.jsp is populated with JavaScript logic later in this development article.

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	Specifies 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	String	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	String	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 an editable grid, develop these files:

- **defaultgrid.js:** contains JavaScript logic that creates a `CQ.Ext.grid.EditorGridPanel` 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 `CQ.Ext.grid.ColumnModel` that defines the columns for the grid. For information, see [CQ.Ext.grid.ColumnModel](#).

The following JavaScript code example defines a `CQ.Ext.grid.ColumnModel` instance.

```

1  var cm = new CQ.Ext.grid.ColumnModel([
2      {
3          id: 'path',
4          header: CQ.I18n.getMessage("Path"),
5          dataIndex: 'jcr:path',
6          width: 220
7      }, {
8          header: CQ.I18n.getMessage("Text"),
9          dataIndex: 'text',
10         width: 200,
11         renderer: function(v) {
12             if (v.length > 50) {
13                 v = v.substring(0, 46) + "...";
14             }
15             return v;
16         }
17     }, {
18         header: CQ.I18n.getMessage("Test"),
19         dataIndex: "test",
20         width: 100,
21         editor: new CQ.Ext.form.TextField({})
22     }
23 ]);

```

In this example, there are three columns defined: *Path*, *Text*, and *Test*. Each column corresponds to a `CQ.Ext.grid.Column` instance. For information, see [CQ.Ext.grid.Column](#).

The text value that appears in each column header is defined by invoking the `CQ.I18n.getMessage` method and passing a string value that specifies the text value. For example:

```
header: CQ.I18n.getMessage("Test")
```

Notice the third column defined in this code example:

```
header: CQ.I18n.getMessage("Test"),
dataIndex:"test",
width: 100,
editor: new CQ.Ext.form.TextField({})
```

The `editor` field specifies the `CQ.Ext.form.Field` to use when editing values in this column. In this example, a `CQ.Ext.form.TextField` is used. This results in the *Test* column being editable. The other two columns in the grid are not editable.

In this grid example, a `CQ.Ext.data.JsonReader` instance is used to store the grid data. A `CQ.Ext.data.JsonReader` object lets you retrieve grid data by using JSON (see below in the sub section named *Get Grid Data*).

The following JavaScript code creates a `CQ.Ext.data.JsonReader` instance.

```
1 //Create a JsonReader instance used to populate the editable grid
2 var reader = new CQ.Ext.data.JsonReader({
3     root: 'hits',
4     totalProperty: 'results',
5     fields: [
6         {name: 'jcr:path', type: 'string'},
7         {name: 'text', type: 'string'},
8         {name: 'test', type: 'string'}
9     ]
10 });
```

Notice that the `fields` property specifies three values that correspond to the `dataIndex` property defined in the data grid columns. In this example, the values `jcr:path`, `text`, and `test` correspond to `dataIndex` values defined in each `CQ.Ext.grid.ColumnModel`. These values must match with the values defined in the columns.

To populate the grid with data in this example, a `CQ.Ext.data.GroupingStore` is used. This data type is a specialized store for grouping data records. For more information, see [CQ.Ext.data.GroupingStore](#).

The following JavaScript code example defines a `CQ.Ext.data.GroupingStore`.

```
1 var store = new CQ.Ext.data.GroupingStore({
2     proxy: new CQ.Ext.data.HttpProxy({
3         url: CQ.HTTP.externalize("/bin/querybuilder.json"),
4         method: "GET"
5     }),
6     baseParams: {
7         "p.limit":0,
8         "p.hits":"full",
9         "path":"/content",
10        "property":"sling:resourceType",
11        "property.value":"foundation/components/text"
12    },
13     reader: reader,
14     sortInfo:{field: 'jcr:path', direction: "ASC"},
15     groupField: 'jcr:path'
16 });
```

The property that is responsible for populating the grid with data is `proxy`. This property provides access to a data object. In this example, a `CQ.Ext.data.HttpProxy` is assigned to the `proxy` property. The `CQ.Ext.data.HttpProxy` object retrieves data from a CQ servlet that corresponds to `/bin/querybuilder.json`. For more information, see [CQ.Ext.data.HttpProxy](#).

In this example, the data returned by the servlet is stored in the `store` variable (an instance of `CQ.Ext.data.GroupingStore`). The `store` variable is used when defining a `CQ.Ext.grid.EditorGridPanel` object. It is assigned to the `CQ.Ext.grid.EditorGridPanel` object's `store` property (as shown below). This is how the grid is populated with data returned by a CQ servlet.

The following JavaScript code creates a `CQ.Ext.grid.EditorGridPanel` object.

```
1 var gridPanel = new CQ.Ext.grid.EditorGridPanel({
2     store: store,
3     stateful: false,
4     cm: cm,
5     clicksToEdit:2,
6     sm: new CQ.Ext.grid.RowSelectionModel({singleSelect:true}),
7     frame:false,
8     listeners: {
9         beforeedit: function(params) {
10         },
11         afteredit: function(params) {
12             var postParams = {};
13             postParams["_charset_"] = "utf-8";
14             postParams["./test"] = params.value;
15             var response = CQ.HTTP.post(params.record.get("jcr:path"), null,
```

```

16         if (CQ.HTTP.isOk(response)) {
17             params.record.commit();
18         } else {
19             params.record.reject();
20         }
21     },
22     },
23     autoExpandColumn: 'path'
24 });

```

In this example, notice that the `cm` property is assigned the `CQ.Ext.grid.ColumnModel` instance. This is how you define the columns that appear in the grid.

Another important piece of code to note is the `afteredit` function.

```

afteredit: function(params) {
    var postParams = {};
    postParams["_charset_"] = "utf-8";
    postParams["./test"] = params.value;
    var response = CQ.HTTP.post(params.record.get("jcr:path"), null,
postParams);
    if (CQ.HTTP.isOk(response)) {
        params.record.commit();
    } else {
        params.record.reject();
    }
}

```

This method is invoked by CQ after a cell's data value modified. You can add application logic to meet your business requirements. In this example, the value `params.value` returns the value of the modified cell value. The modified value is posted back to CQ using `CQ.HTTP.post` method. If the response is OK, the record is committed by invoking the `params.record.commit` method. If the response is not OK, then the change is rejected by invoking the `params.record.reject` method.

Get Grid Data

You can get the data located within the grid by calling the grid object's `getStore` method. This method returns a `CQ.Ext.data.Store` instance. You can call methods of this object to get data. For information, see [CQ.Ext.data.Store](#).

For example, the following code shows how to get a `CQ.Ext.data.Store` object and then get the number of records within that store (each row in the grid is considered a record).

```

var store = grid.getStore();
var countRecs=store.getCount();

```

You can get the value of each grid cell by iterating through the grid and getting a `CQ.Ext.data.Record` for each row in the grid. For information, see [CQ.Ext.data.Record](#).

The following JavaScript code is a method named `getGridData` that iterates through the grid and gets the data value of the second and third columns.

```

function getGridData(grid)
{
    var store = grid.getStore();
    var tt=store.getCount();
    var iiu=0;

    //Iterate through the grid and get back data
    for (var z=0; z < tt; z++)
    {
        var myRec = store.getAt(z);
        var myJSON = myRec.json ;

        //Get the values of the 2nd and 3rd columns
        var TextVal = myJSON.text;
        var TestVal = myJSON.test;
    }
}

```

Notice that for each iteration, the `CQ.Ext.grid.EditorGridPanel` object's `getAt` method is invoked. This method returns a `CQ.Ext.data.Record` instance. You can get the value of the `CQ.Ext.data.Record` object's `json` property. This property returns JSON data and is only valid if this record was created by an `ArrayReader` or a `JsonReader`. (In this example, `JsonReader` is used).

Now you can get the value of each column by referencing the name of the column. Remember that

the column name is specified when the `CQ.Ext.data.JsonReader` instance is created (this is shown earlier in this section).

The following JavaScript represents the entire `defaultgrid.js` file. Notice that this file contains application logic for a method named `getGridPanel`. This method returns an instance of `CQ.Ext.grid.EditorGridPanel`.

```

1  //-----
2  // example grid showing a reference search
3  // and how cells can be edited
4
5  //Get the grid data
6  function getGridData(grid)
7  {
8
9      var store = grid.getStore();
10     var tt=store.getCount();
11
12     //Iterate through the grid and get back data
13     for (var z=0; z < tt; z++)
14     {
15         var myRec = store.getAt(z);
16         var myJSON = myRec.json ;
17
18         //Get the Values of grid columns - 2nd and 3rd
19         var TextVal = myJSON.text;
20         var TestVal = myJSON.test;
21
22     }
23
24 }
25
26 function getGridPanel() {
27     var reader = new CQ.Ext.data.JsonReader({
28         root: 'hits',
29         totalProperty: 'results',
30         fields: [
31             {name: 'jcr:path', type: 'string'},
32             {name: 'text', type: 'string'},
33             {name: 'test', type: 'string'}
34         ]
35     });
36
37     var cm = new CQ.Ext.grid.ColumnModel([
38         {
39             id: 'path',
40             header: CQ.I18n.getMessage("Path"),
41             dataIndex: 'jcr:path',
42             width: 220
43         }, {
44             header: CQ.I18n.getMessage("Text"),
45             dataIndex: 'text',
46             width: 200,
47             renderer: function(v) {
48                 if (v.length > 50) {
49                     v = v.substring(0, 46) + "...";
50                 }
51                 return v;
52             }
53         }, {
54             header: CQ.I18n.getMessage("Test"),
55             dataIndex: "test",
56             width: 100,
57             editor: new CQ.Ext.form.TextField({})
58         }
59     ]);
60     // by default columns are sortable
61     cm.defaultSortable = true;
62
63     var store = new CQ.Ext.data.GroupingStore({
64         proxy: new CQ.Ext.data.HttpProxy({
65             url: CQ.HTTP.externalize("/bin/querybuilder.json"),
66             method: "GET"
67         }),
68         baseParams: {
69             "p.limit": 0,
70             "p.hits": "full",
71             "path": "/content",
72             "property": "sling:resourceType",
73             "property.value": "foundation/components/text"
74         },
75         reader: reader,
76         sortInfo: {field: 'jcr:path', direction: "ASC"},
77         groupField: 'jcr:path'

```

```

78     });
79
80     var gridPanel = new CQ.Ext.grid.EditorGridPanel({
81         store: store,
82         stateful: false,
83         cm: cm,
84         clicksToEdit: 2,
85         sm: new CQ.Ext.grid.RowSelectionModel({singleSelect:true}),
86         frame:false,
87         listeners: {
88             beforeedit: function(params) {
89             },
90             afteredit: function(params) {
91                 var postParams = {};
92                 postParams["_charset_"] = "utf-8";
93                 postParams["./test"] = params.value;
94                 var response = CQ.HTTP.post(params.record.get("jcr:path"), null);
95                 if (CQ.HTTP.isOk(response)) {
96                     params.record.commit();
97                 } else {
98                     params.record.reject();
99                 }
100             }
101         },
102         autoExpandColumn: 'path'
103     });
104     return gridPanel;
105 }

```

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.EditorGridPanel` 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.EditorGridPanel` 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.

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

```

1  grid = new CQ.Ext.Window({
2      id:"<%= node.getName() %>-grid",
3      title:"Grid Example 24",
4      layout:"fit",
5      hidden:true,
6      collapsible:true,
7      renderTo:"CQ",
8      width:<%= width %>,
9      height:<%= height %>,
10     x:<%= docked ? 0 : 220 %>,
11     y:<%= docked ? 0 : 200 %>,
12     closeAction:'hide',
13     items: gridPanel,
14     listeners: {
15         beforeshow: function() {
16             gridPanel.getStore().load();
17         }
18     },
19     buttons:[{
20         text:"Close",
21         handler: function() {
22             grid.hide();
23         }
24     },{
25         text:"Dock",

```

```

26         handler: function() {
27             grid.setPosition(0,0);
28         }
29     }]]
30 });

```

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.EditorGridPanel`. This is how a data grid is associated with the window that is defined by using a `CQ.Ext.Window` data type.

Notice that three buttons are defined.

```

buttons:[{
    text:"Close",
    handler: function() {
        grid.hide();
    }
},{
    text:"Dock",
    handler: function() {
        grid.setPosition(0,0);
    }
},{
    text:"Data",
    handler: function() {
        getGridData(gridPanel);
        var tt=00;
    }
}
]

```

The first button closes the data grid when the button is clicked. Likewise, the Dock window sets the data grid to position 0. Using methods that belong to `CQ.Ext.grid.EditorGridPanel`, you can further control the behaviour of the grid. The third button calls the `getGridData` method (defined in `defaultgrid.js`) that gets grid data.

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

```

1  <%@include file="/libs/foundation/global.jsp"%><%
2
3      Node node = resource.adaptTo(Node.class);
4      // load properties
5      int width = properties.get("width", 600);
6      int height = properties.get("height", 300);
7      boolean docked = properties.get("docked", false);
8
9      %>
10     <h3>Exercise 5: Grid Overview</h3><%
11     %><p>Learn about:
12         <ul>
13             <li>The grid config</li>
14             <li>The store config</li>
15             <li>The expected data format
16         </ul>
17         <code><pre>
18         {
19             results: 3,
20             root: [
21                 { 'id': 1, 'firstname': 'Bill', occupation: 'Gardener' },
22                 { 'id': 2, 'firstname': 'Frank', occupation: 'Programmer' },
23                 { 'id': 3, 'firstname': 'Ben', occupation: 'Horticulturalist' }
24             ]
25         }
26         </pre></code>
27         The above data would require the following store config to consume:
28         <code><pre>
29         var store = new CQ.Ext.data.JsonStore({
30             reader: {
31                 totalProperty:"results",
32                 root:"root",
33                 id:"id"
34             },
35             url:"/content/test.json"
36         });
37         </pre></code>
38         </li>
39     </ul>
40     </p>
41     <script type="text/javascript" src="/apps/grid/components/aemgrid/reference.js">

```

```

41
42 <script type="text/javascript">
43
44     var grid = CQ.Ext.getCmp("<%= node.getName() %>-grid");
45     if (!grid) {
46
47         var gridPanel = getGridPanel();
48
49         grid = new CQ.Ext.Window({
50             id:"<%= node.getName() %>-grid",
51             title:"Editable Grid Example",
52             layout:"fit",
53             hidden:true,
54             collapsible:true,
55             renderTo:"CQ",
56             width:<%= width %>,
57             height:<%= height %>,
58             x:<%= docked ? 0 : 220 %>,
59             y:<%= docked ? 0 : 200 %>,
60             closeAction:'hide',
61             items: gridPanel,
62             listeners: {
63                 beforeshow: function() {
64                     gridPanel.getStore().load();
65                 }
66             },
67             buttons:[{
68                 text:"Close",
69                 handler: function() {
70                     grid.hide();
71                 }
72             },{
73                 text:"Dock",
74                 handler: function() {
75                     grid.setPosition(0,0);
76                 }
77             },{
78                 text:"Data",
79                 handler: function() {
80                     getGridData(gridPanel);
81                     var tt=00;
82                 }
83             }
84         ]
85     });
86     grid.show();
87 } else {
88     grid.setWidth(<%= width %>);
89     grid.setHeight(<%= height %>);
90     grid.setPosition(<%= docked ? 0 : 500 %>,<%= docked ? 0 : 100 %>);
91     grid.show();
92 }
93
94
95
96 </script>

```

Add the files to the project

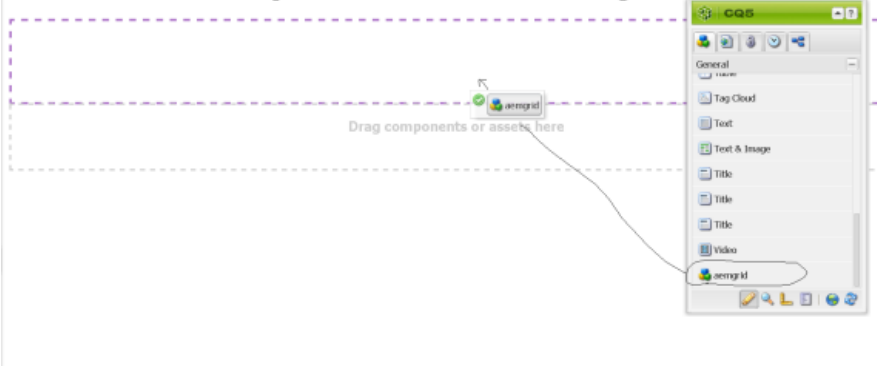
1. To view the CQ welcome page, enter the URL: `http://[host name]:[port]` into a web browser.
For example, `http://localhost:4502`.
2. Select CRXDE Lite.
3. Double-click `/apps/grid/components/aemgrid/aemgrid.jsp`.
4. Replace the JSP code with the new code shown in this section.
5. Select `apps/grid/components/aemgrid`. 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!

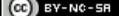


The CQ sidekick displaying the aemgrid component that is created in this development article

1. Go to the CQ welcome page at [http://\[host name\]:\[port\]](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.

See also

Congratulations, you have just created an AEM xtype component. Please refer to the [AEM community page](#) for other articles that discuss how to build AEM services/applications.

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

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