Maqetta: Visual Authoring of HTML5 User Interfaces

Log in to participate

Adding Your Own JavaScript to Maqetta Prototypes

tonyerwin | *July 20 2012* | *Comments (11)* | *Visits (5952)*

Updated Sept 10, 2012 for Release 7.

1 Like 0 Share Tweet 0

Maqetta allows you to create very rich HTML5 prototypes without writing a line of code. However, there might come a time when you'd like to provide some of your own custom JavaScript to take your prototypes to the next level. Or, as alluded to in a previous blog post (see *Maqetta to RAD 8.5 Workflow*), you might be done with prototyping and ready to move onto the next phase of building a fully functional, production-ready application. Whichever camp you fall in, this article will use several examples to walk you through the process of starting to write JavaScript to go with your Maqetta prototypes.

Creating a New Desktop Application

Let's get started by creating a new HTML file and looking at its contents:

1. Open the **Create** menu in the main menu bar and choose the **Desktop Application...** option. This will bring up a dialog asking for a filename and location. You can just accept the defaults by clicking the **Create** button.
2. At this point, you should see a new editor with an empty canvas. Click the **Source** button in the editor's toolbar to look at the HTML source.

The source should look like the listing below. In particular, note the following line which causes a JavaScript file called *app.js* to be loaded: `<script type="text/javascript" src="app.js"></script>` . We'll look at this JavaScript file more closely in the next section.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>Untitled</title>
<script type="text/javascript" src="lib/dojo/dojo/dojo.js" data-dojo-config="'parse
OnLoad':true,'async':true,'packages':[{'name':'gridx','location':'../gridx'},{'name'
:'clipart','location':'../../clipart'},{'name':'maqettaSamples','location':'../../..
/samples'},{'name':'maqetta','location':'../../maqetta'},{'name':'shapes','location'
:'../../shapes'},{'name':'zazl','location':'../../zazl'},{'name':'widgets','location
':'../../custom'}]"></script>
<script type="text/javascript">
require([
  "dijit/dijit",
  "dojo/parser",
  "maqetta/space",
  "maqetta/AppStates"
]);
</script>
<style>@import "themes/claro/document.css";@import "themes/claro/claro.css";@import
"app.css";
</style>
<script type="text/javascript" src="app.js"></script>
</head>
<body class="claro" data-maq-flow-layout="true" data-maq-ws="collapse" id="myapp"
data-maq-appstates="{}">
</body>
</html>
```

Now that we've looked the the source, you can click the **Design** button in the editor's toolbar to go back to the page designer.

Exploring the Default *app.js* File

The *app.js* file loaded by all Maqetta-generated HTML files is included in Maqetta projects by default. So, it provides a

About this blog

In this blog, Tony Erwin covers a range of topics related to Maqetta. Maqetta is an open source project that provides WYSIWYG visual authoring of HTML5 user interfaces (desktop and mobile).

Related posts

Registry Services on...		
UpdatedJune 19	0	0
連載：ちょっとディープなXPages　第...		
UpdatedJune 18	1	0
Insights from Innova...		
UpdatedJune 12	1	1
Maqetta Means Mock-u...		
UpdatedJune 12	1	0
We're on a mission t...		
UpdatedMay 30	6	0

Tags

Find a Tag

css design dijit dojo gridx html5 javascript maqetta mobile phonegap prototype rational_application_developer rational-application-developer welcome workflow

Cloud | List

Recent tweets

Follow @tonyerwin

convenient place to insert your own JavaScript.

Let's look at the *app.js* file by double-clicking on it in the Files palette in the lower left of the Maqetta workbench. You should see a new editor with a listing like the one below:

```
/*
 * This file is provided for custom JavaScript logic that your HTML files might need.
 * Maqetta includes this JavaScript file by default within HTML pages authored in
Maqetta.
 */
require(["dojo/ready"], function(ready){
     ready(function(){
         // logic that requires that Dojo is fully initialized should go here

     });
});
```

The *app.js* file works by making use of dojo/ready. Any code placed within the `ready` function is guaranteed to run only after the page and necessary Dojo resources are fully loaded.

Adding An Alert to *app.js*

At this point, let's write some very simple JavaScript to give a hint as to what's possible. In the editor you have opened for *app.js* add an `alert` statement as shown below:

```
/*
 * This file is provided for custom JavaScript logic that your HTML files might need.
 * Maqetta includes this JavaScript file by default within HTML pages authored in
Maqetta.
 */
require(["dojo/ready"], function(ready){
     ready(function(){
         // logic that requires that Dojo is fully initialized should go here

        //Add a temporary alert just to make sure we're working
        alert("Code from app.js is running!");
     });
});
```

Click **Save** to save your updates. Then, switch back to the editor for your HTML file and click the **Preview** button in the toolbar. If everything is working correctly, then when your page is previewed you should see an alert box that says "Code from app.js is running!" .

[*HELPFUL HINT:* If you've done any previous JavaScript development, you probably already know this. But, if you change JavaScript files, it's typically necessary to clear your browser's cache before the changes will be loaded. Please keep this in mind as we modify *app.js* and preview the HTML in the rest of this article.]

Modifying DOM Elements

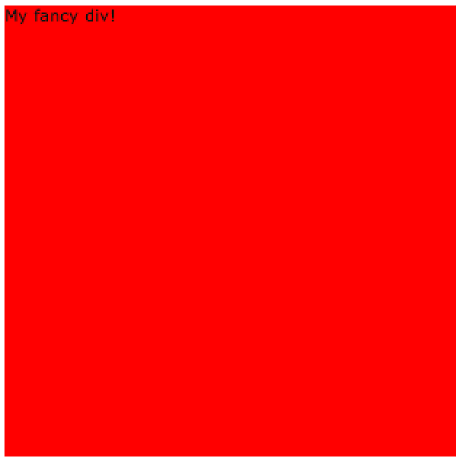
We'll now look at some slightly more advanced JavaScript that will modify DOM elements to cause dynamic changes on your page. First, let's use the Maqetta page editor to add some content to your HTML file:

1. From the HTML folder in the *Widgets* palette, drag and drop a <div> element onto your page.
2. Double click on the element. This will bring up an input dialog where you can set the <div>'s contents. Enter whatever you like (such as, "This is my fancy div!").
3. Set the ID of the <div> to "myFancyDiv" using the *Properties* palette.
4. Save your HTML file.

Now, switch back to the editor for *app.js* and replace the contents with the JavaScript below:

```
/*
 * This file is provided for custom JavaScript logic that your HTML files might need.
 * Maqetta includes this JavaScript file by default within HTML pages authored in
Maqetta.
 */
require(["dojo/ready", "dojo/dom", "dojo/dom-style", "dijit/registry"],
function(ready, dom, domStyle, registry){
     ready(function(){
         // logic that requires that Dojo is fully initialized should go here

        //Get a reference to our div
        var myDiv = dom.byId("myFancyDiv");

        //Make sure we successfully got a reference
        if (myDiv) {
                // Change the style of the div. We could
                // have done this directly in Maqetta, but
                // done here just to show one of the simplest
                // things you can do with dojo style manipulation
                domStyle.set(myDiv, {
                        backgroundColor: "red"
                });

                //Change the size of the div on a repeating
                //timer. We could *not* have done this with
                //Maqetta alone.
                var size = 150;
                setInterval(function() {
                        //set the style on the div to change the size
                        domStyle.set(myDiv, {
                                width: size + "px",
                                height: size + "px"
                        });

                        //Increment size
                        size += 40;

                        //if size gets too big, then start over
                        if (size > 500) {
                                size = 150;
                        }
                }, 250);
        }
     });
});
```

After saving *app.js*, go back to the editor for your HTML file and click the preview button in the toolbar. When you do so, you should see that your <div> element is now red and it grows and shrinks repeatedly.


My fancy div!

This article isn't by any means intended to be a full JavaScript (or Dojo) tutorial, but the important things to note in the JavaScript code that made this happen are:

- `dom.byId` is used to get a reference to the <div> that we had added to our HTML page using the page editor. The `dom` variable refers to `dojo/dom` by virtue of the addition of `"dojo/dom"` to the list of requires, and `dojo/dom` provides the `byId` function.
- `domStyle.set` is then used to modify the styling of the <div> element. The `domStyle` variable refers to `dojo/dom-style` by virtue of the addition of `"dojo/dom-style"` to the list of requires, and `dojo/dom-style` provides the `set` function. Our first use of `domStyle.set` changes the background color of the <div> to red (which, of course, we could have done in the Maqetta page editor using the *Properties* palette).
- We then start a timer that changes the size of the <div> every 150 ms (again leveraging `domStyle.set`). This may not seem terribly useful, but is just meant to show a simple example of DOM manipulation that you can't do with the Maqetta page editor alone.

Working with Dojo Widgets

In our next example, we'll explore how to manipulate Dojo widgets via JavaScript to build an extremely poor man's "stopwatch." First, using the Maqetta page editor:

1. Delete the <div> element that you previously added.
2. From the HTML folder in the *Widgets* palette, drag and drop a <label> element and enter "Seconds:" for its content.
3. From the Dojo folder in the *Widgets* palette, drag and drop a TextBox element next to the <label>
4. Set the ID of the TextBox element to "myTextBox" using the *Properties* palette.

Now, switch back to the editor for *app.js* and replace the contents with the JavaScript below:

```
/*
 * This file is provided for custom JavaScript logic that your HTML files might need.
 * Maqetta includes this JavaScript file by default within HTML pages authored in
Maqetta.
 */
require(["dojo/ready", "dojo/dom", "dojo/dom-style", "dijit/registry"],
function(ready, dom, domStyle, registry){
     ready(function(){
         // logic that requires that Dojo is fully initialized should go here

        //Get reference to our TextBox using dijit
        var textBox = registry.byId("myTextBox");
        if (textBox) {
                //Inititialize a counter
                var seconds = 0;

                //Start a timer that fires every second
                setInterval(function() {
                        //Update the TextBox's value (and the counter)
                        textBox.set("value", seconds++);
                }, 1000);
         }
     });
});
```

After saving *app.js*, preview your HTML page. In the preview, you should see a text box with a number that increments once every second like shown in the screen shot below:

Important things to note from the JavaScript:

- `registry.byId` is used to get a reference to the widget. The `registry` variable refers to `dijit/registry` by virtue of the addition of `"dijit/registry"` to the list of requires, and `dijit/registry` provides the `byId` function.
    - Note that `registry.byId` serves a very different purpose than `dom.byId` used in the previous sample. `registry.byId` is specifically used to work with Dojo widgets; whereas, `dom.byId` works only with HTML DOM elements.
- The object we get back from `registry.byId` is an instance dijit.form.TextBox which provides many functions that can be called for a variety of purposes. In this case, we make use of the `set` function (common to many Dojo widgets) for changing attributes. We call `set` within a timer to change the "value" attribute of the TextBox once every second.

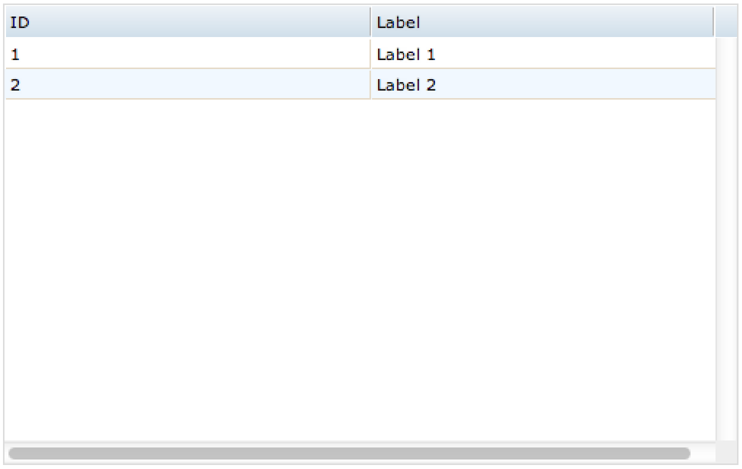
Working with Advanced Dojo Widgets

We'll now go through a final sample working with more advanced Dojo widgets. This final sample is based on a real-world scenario presented by a Maqetta user in our Google Groups forum. We will create a GridX widget (a widget used for displaying tabular data) in the Maqetta page editor and then use JavaScript to modify the data it displays at runtime.

First, using the Maqetta page editor:

1. Delete the <label> and <div> elements that you added for the last example.
2. From the Dojo folder in the *Widgets* palette, drag and drop a GridX element onto your page.
3. Click **Finish** on the first panel of GridX Configuration Wizard. NOTE: Much more detailed information on how to use the wizard to do advanced configuration of GridX can be found in one of my earlier blog posts (see *Maqetta GridX Configuration Wizard*).
4. Set the ID of the GridX element to "myGridX" using the *Properties* palette.

Your GridX will contain some simple default data and should look something like this in the Maqetta page editor:

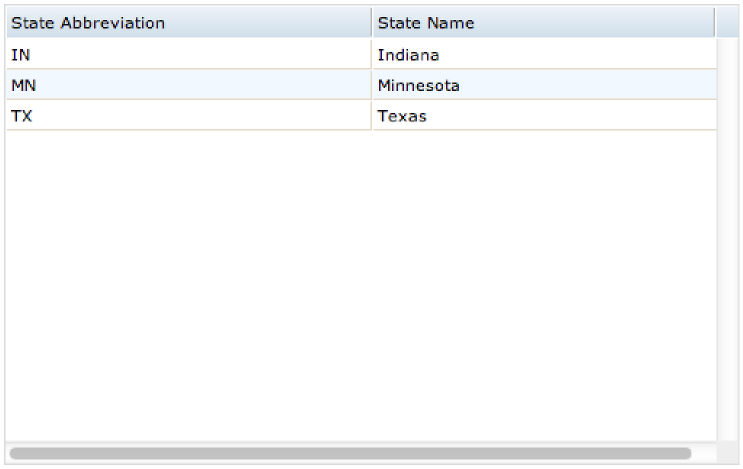


ID	Label
1	Label 1
2	Label 2

Now, switch back to the editor for *app.js* and replace the contents with the JavaScript below:

```
/*
 * This file is provided for custom JavaScript logic that your HTML files might need.
 * Maqetta includes this JavaScript file by default within HTML pages authored in
Maqetta.
 */
require(["dojo/ready", "dojo/dom", "dojo/dom-style", "dijit/registry"],
function(ready, dom, domStyle, registry){
     ready(function(){
         // logic that requires that Dojo is fully initialized should go here

        // Create a local reference to our data store (an
        // instance of dojo/data/ItemFileReadStore) for
        // convenience. The data store was created by Maqetta when
        // we created our GridX widget. And, Maqetta gave it an
        // id of "ItemFileReadStore_1". In addition, Maqetta gave it
        // a jsId of "ItemFileReadStore_1", so Dojo created a
        // global variable we can point at.
        var store = ItemFileReadStore_1;

        // Create new data for data store
        var myData =
                {
                        identifier: "unique_id",
                        items: [
                                {unique_id:1, id: "IN", label: "Indiana"},
                                {unique_id:2, id: "MN", label: "Minnesota"},
                                {unique_id:3, id: "TX", label: "Texas"}
                        ]
                };

        // Set new data on data store (the store has jsId set, so there's
        // a global variable we can reference)
        store.clearOnClose = true;
        store.data = myData;
        store.close();

        // Get reference to our grid object. I set the id to "GridX" using
        // the Maqetta properties palette.
        var myGridX = registry.byId("myGridX");
        if (myGridX) {
                // Tell our grid to reset itself
                myGridX.store = null;
                myGridX.setStore(store);

                // Optionally change column structure on the grid
                myGridX.setColumns([
                        {field:"id", name: "State Abbreviation"},
                        {field:"label", name: "State Name"},
                ]);
        }
     });
});
```

Now, when you preview your HTML, you should see something like this in your browser (with the data from Maqetta replaced and the column labels modified):



State Abbreviation	State Name
IN	Indiana
MN	Minnesota
TX	Texas

Important things to note from the JavaScript:

- We first work with the data store that Maqetta created for us when we clicked Finish on the GridX wizard. Dojo data stores provide a standardized interface used by a wide variety of widgets for data access. To change the data displayed by GridX, we need to change the data stored in the data store. The data store used in this case is an instance of `dojo.data.ItemFileReadStore`, and we provide it a new array of items.
- Once we modify the data store, we get a reference to the GridX widget using `registry.byId` (the same mechanism we used to get a reference to the TextBox in the previous example). The widget we get is an instance of `gridx.GridX` and it provides many useful functions. We use the `setStore` function for reloading the store, and `setColumns` to change the column labels.

Tags: *dojo dijit prototype gridx design maqetta javascript*

Add a Comment | More Actions

Comments (11)

Add a Comment | *More Actions*

1 ***justin-m*** *commented Jan 13* *Permalink*

Hi Tony, <div>&nbsp;</div> Thanks for this tutorial. I'm not a developer, but I could still follow along. Some of the references to dojo is uncompromisable to me since I don't know javascript. I enjoy using Maqetta drag and drop HTML5 canvas to make interactive prototypes. <div>&nbsp;</div> Can you help me or point me in the right direction about how I can use other javascript libraries within Maqetta. I like Twiiter Bootstrap and they're using some jQuery plugins. Since the bootstrap.js requires jQuery would it be as simple as putting all the javascript into app.js ? <div>&nbsp;</div> Thanks for your help. <div>&nbsp;</div> Regards, <br /> Justin

2 ***JonFerraiolo*** *commented Jan 14* *Permalink*

Justin, <br /> Thanks for comments and questions. <div>&nbsp;</div> The good news is that we designed Maqetta with a plug-in approach to JavaScript libraries, widget libraries and CSS themes. However, with the current release, you need to be a programmer to take advantage of these extensibility points. You need to have good knowledge of JavaScript coding (along with some knowledge of Java and OSGi bundles), and you have to create a custom build. <div>&nbsp;</div> I expect we will add alternate plug-in approaches before too long that will allow non-programmers to add custom widget libraries and themes to an existing implementation of Maqetta, without requiring any JavaScript or Java coding skills, and not requiring a custom build. I have no concrete estimate on when this would be available, but it is a top requirement for lots of users. <div>&nbsp;</div> Jon

3 ***MaqettaUser*** *commented Feb 10* *Permalink*

Hi Tony, <br /> I am currently trying to create a UI that has an event logger and from what I can understand a GridX widget would be good for this!? As usual, I am relatively new to Javascript, Maqetta &amp; Dojo but I found your above tutorial on GridX really helpful. Unfortunately, using the example above I am not able to append new lines to the bottom of the grid? I tried: <br /> var myNewItem = {unique_id: 4, id: "NY", label: "New York"}; <br /> store.newItem(myNewItem); <div>&nbsp;</div> but it didn't work. Any help that you could give would be much appreciated? <div>&nbsp;</div> Best regards, <div>&nbsp;</div> Matt <br />

4 ***tonyerwin*** *commented Feb 11* *Permalink*

Hi, Matt, <div>&nbsp;</div> Thanks for your question. The key here is that Maqetta creates the data store for GridX as an ItemFileReadStore, but if you want to dynamically add/remove rows, an ItemFileWriteStore needs to be used. <div>&nbsp;</div> So, I modified the sample JavaScript to create an ItemFileWriteStore for the GridX. And, then I can successfully make the store.newItem call you wanted to make. <div>&nbsp;</div> I've included the updated JS below. I hope this helps. <div>&nbsp;</div> Good luck, <br /> Tony <div>&nbsp;</div> <div>&nbsp;</div> /* <br /> * This file is provided for custom JavaScript logic that your HTML files might need. <br /> * Maqetta includes this JavaScript file by default within HTML pages authored in Maqetta. <br /> */ <br /> require(["dojo/ready", "dojo/dom", "dojo/dom-style", "dijit/registry", "dojo/data/ItemFileWriteStore"], function(ready, dom, domStyle, registry, ItemFileWriteStore){ <br /> ready(function(){ <br /> // logic that requires that Dojo is fully initialized should go here <div>&nbsp;</div> // Create new data for data store <br /> var myData = <br /> { <br /> identifier: "unique_id", <br /> items: [<br /> {unique_id:1, id: "IN", label: "Indiana"}, <br /> {unique_id:2, id: "MN", label: "Minnesota"}, <br /> {unique_id:3, id: "TX", label: "Texas"} <br />] <br /> }; <div>&nbsp;</div> // Need to change our store to ItemFileWriteStore because we want to dynamically <br /> // add rows <br /> var store = new ItemFileWriteStore({ <br /> data: myData <br /> }); <div>&nbsp;</div> // Get reference to our grid object. I set the id to "GridX" using <br /> // the Maqetta properties palette. <br /> var myGridX = registry.byId("myGridX"); <br /> if (myGridX) { <br /> // Tell our grid to reset itself <br /> myGridX.store = null; <br /> myGridX.setStore(store); <div>&nbsp;</div> // Optionally change column structure on the grid <br /> myGridX.setColumns([<br /> {field:"id", name: "State Abbreviation"}, <br /> {field:"label", name: "State Name"}, <br />]); <div>&nbsp;</div> // After 1 seconds add a row to the grid <br /> setTimeout(function() { <br /> var myNewItem = {unique_id: 4, id: "NY", label: "New York"}; <br /> store.newItem(myNewItem); <br /> }, 1000); <br /> } <br /> }); <br /> }); <br />

5 ***MaqettaUser*** *commented Feb 11* *Permalink*

Hi Tony, <div>&nbsp;</div> your suggestion worked really well! Thanks a lot for the help, it is much appreciated! <div>&nbsp;</div> Best regards, <div>&nbsp;</div> Matt <br />

6 ***triessner*** *commented Feb 20* *Permalink*

Hi Tony, <br /> thanks for you effort, to make maqetta more usable. Nevertheless I'm stuck in the following problem now: I want to Design a database application with maqetta using gridx - tables to show the data. A lot of it already works. I can get the data from a json store (I wrote a small php-script, that accesses my postgresql database and sends json data back to the dojo.store.JsonRest) and the gridx shows the data. Even the sort module works meanwhile :-). But I didn't overcome the problem, that the table shows just the number of rows, that I put in the csv-table during configuring the gridx. As the number of rows in my database are quite big for some tables, or changes from time to time, it is quite uncomfortable to put in so many rows in the csv table of the configuring wizard. Is there any possibility that the gridx reads the number of rows from my json-store at runtime? How can I add VirtualVScrolling? It would be really great if I could get some hints for that. <br /> Regards, <br /> Thilo

7 ***tonyerwin*** *commented Feb 21* *Permalink*

Hi, Thio, <div>&nbsp;</div> Thanks for your question. It sounds like you've successfully set a different store on the GridX and are retrieving some of your data. It also sounds like you've been able to attach GridX modules for sorting, etc. So, all that's good. <div>&nbsp;</div> But, offhand, It doesn't really make sense to me why it would only retrieve data up to the count of rows your happened to enter in the GridX wizard. <div>&nbsp;</div> I've not done a lot with GridX and very large data sets, but I was looking at their sample gallery in the GridX toolkit (see gridx/gallery/gallery.html) and they have a sample called "GridX with a huge store(1,000,000 rows)" that might help you (see gridx/tests/test_grid_huge_data.html). <div>&nbsp;</div> It loads in a gridx/tests/test_grid_huge_data.js file which creates the GridX and appears to set the store to an XQueryReadStore. This store is a custom subclass of ojox/data/QueryReadStore they appear to have written to better deal with paging. <div>&nbsp;</div> It might also be useful if you to look at the modules and attributes they set on their GridX instance. <div>&nbsp;</div> So, I'm not sure if I'm being much help here and if this will allow you get further or not... GridX is actually a separate open source project from Maqetta (see https://github.com/oria/gridx) and the folks there might also have some hints to help you. <div>&nbsp;</div> Thanks, <br /> Tony

8 ***triessner*** *commented Feb 21* *Permalink*

Hi Tony, <br /> thanks for your answer. Yes I replace the default store, created by the wizard (ItemFileReadStore_2) by a Cache store, which combines a Memory store and a JsonRest store, which points to my Json-Database php-script. <br /> Here is the code: <div>&nbsp;</div> var userMemoryStore = new dojo.store.Memory(); <br /> var userJsonRestStore = new dojo.store.JsonRest({target: "http://localhost/php/controller-pgsql.php/some_table/", <br /> sortParam: "sortBy"}); <br /> var store = new dojo.store.Cache(userJsonRestStore, userMemoryStore); <div>&nbsp;</div> // Get reference to our grid object. I set the id to "myGridX" using <br /> // the Maqetta properties palette. <br /> var myGridX = registry.byId("myGridX"); <br /> if (myGridX) { <br /> // Tell our grid to reset itself <br /> myGridX.store = null; <br /> myGridX.cacheClass=Cache; <br /> myGridX.setStore(store); <br /> ..... <div>&nbsp;</div> One funny thing is, that if I use just the JsonRestStore (and not the Cache store), the table shows only the last row delivered by the php-script, repeated as often as there are rows in my initial store (ItemFileReadStore_2). <br /> But I'll follow your hint, and put the question to the GridX folks. <br /> Thanks for your help. <br /> Thilo

9 ***MaqettaUser*** *commented Mar 9* *Permalink*

Hi Tony, <div>&nbsp;</div> would it be possible to extend this article to add an example of working with a dynamic dijit tree? I have been trying in Maqetta to populate and update (add, delete, refresh) a tree from javascript using json data for a while now but it never seems to work. <div>&nbsp;</div> As before with the GridX, any help would be much appreciated. <div>&nbsp;</div> Best regards, <div>&nbsp;</div> Matt <br />

10 ***tonyerwin*** *commented Mar 11* *Permalink*

Hi, Matt, <div>&nbsp;</div> Thanks for your question on dijit trees. Unfortunately, at the moment, I'm spending the bulk of my time on another project and won't have any time in the short-term to develop a more tree-oriented example. <div>&nbsp;</div> It's possible the other guys on the Maqetta team could help you on the Maqetta Users Google Group (especially if you have more specifics on what you tried and what didn't work): <div>&nbsp;</div> https://groups.google.com/forum/?fromgroups#!forum/maqetta-users <div>&nbsp;</div> Thanks, <br /> Tony <br />

11 ***MaqettaUser*** *commented Mar 11* *Permalink*

Hi Tony, <div>&nbsp;</div> don't worry about it, I understand about your other commitments. I finally managed to make a break through anyway using: <br /> http://dojotoolkit.org/reference-guide/1.8/dijit/Tree.html#dijit-tree <div>&nbsp;</div> Good luck with the other project and best regards, <div>&nbsp;</div> Matt <br />

Show: 10 | 20 | 30 items per page 1 Previous | Next

Previous Entry | Main | Next Entry