

Article

[Vitaliy Serdtsev](#) · Dec 22, 2017 10m read

\$(REST - CSP - (-HyperEvents) + EasyUI + File Upload). Part 5

Let 's continue our series of articles about creating a web application using REST only.

Integration with jQuery EasyUI (using datagrid and tree components as examples)

Anyone interested in seeing offline demos without the server part can find them in *demo* and *demo-mobile* folders of the downloaded archive.

IE users may want to enable the “ Allow active content to run in files on My Computer ” option in the browser 's settings to save themselves the trouble of having to deal with a popup window all the time.

You can also replace the *jquery.min.js* file with its newer version: specifically, 1.11.x for local test in IE11, since local AJAX doesn 't work in version 2.1.x for IE11.

Additionally, you may want to check out an [online demo](#), as well as the [tutorial](#) and [documentation](#).

So, the datagrid comes first.

Datagrid

To begin with, let 's create a persistent class and fill it with data:

```
Class my.a Extends %Persistent [ Final ]
{
    Property idp As %Integer;
    Property name As %String;

    /// d ##class(my.a).Fill()
    ClassMethod Fill(count = 10)
    {
        d ..%KillExtent()
        f i=1:1:count s ^my.aD(i)=$lb(, "name_"_i)
        s ^my.aD=count
        s ^my.aD(6)=$lb(, "<Hello> from <br><span style='color:red;'>Caché</span>")

        s ^my.aD(1)=$lb(, "ROOT")
        s ^my.aD(2)=$lb(1, "node2")
        s ^my.aD(3)=$lb(2, "node3")
        s ^my.aD(4)=$lb(1, "node4")
        s ^my.aD(5)=$lb(4, "node5")
    }
}
```

```
USER>d ##class(my.a).Fill()
```

Let 's recreate our REST class to only keep our eye on relevant topics going forward:

\$(REST - CSP - (-HyperEvents) + EasyUI + File Upload). Part 5

Published on InterSystems Developer Community (<https://community.intersystems.com>)

```
Class my.rest Extends %CSP.REST
{

Parameter CHARSET = "UTF-8";

Parameter CONTENTTYPE = {..#CONTENTTYPEJSON};

Parameter UseSession As Integer = 1;

XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
<Routes>
  <Route Url="/" Method="GET" Call="MainPage"/>
  <Route Url="/(.*)" Method="GET" Call="StaticFiles"/>
</Routes>
}

ClassMethod StaticFiles(url) As %Status
{
  k %request.Data
  s %request.Data("FILE",1)=%request.URL

  d ##class(%CSP.StreamServer).OnPreHTTP()
  d ##class(%CSP.StreamServer).OnPage()
  q $$$OK
}

ClassMethod MainPage() As %Status
{
  s %response.ContentType="text/html"
  &html<
<!DOCTYPE html>
<html>
  <head>
    <meta charset="#(..#CHARSET)#">
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0,maximum-
scale=1, user-scalable=no"/>
    <link rel="stylesheet" type="text/css" href="easyui/themes/default/easyui.css">
    <link rel="stylesheet" type="text/css" href="easyui/themes/icon.css">
    <script type="text/javascript" src="easyui/js/jquery.min.js"></script>
    <script type="text/javascript" src="easyui/js/jquery.json.min.js"></script>
    <script type="text/javascript" src="easyui/js/jquery.easyui.min.js"></script>
    <script type="text/javascript" src=
"easyui/locale/easyui-lang-#(%session.Language)#.js"></script>
    <title>Main page</title>
    <script type="text/javascript">

    var MyApp={

    };

    (function($){

      $.ajaxSetup({
        type: 'POST',
        dataType: 'json',
        cache: false,
        async: true,
```

```

        contentType: 'application/json; charset=UTF-8',
        processData: false,
        beforeSend: function(jqXHR, settings) {
            settings.data=$.toJSON(settings.data);
        },
        statusCode: {
            401: function() {alert('You are not authorized')},
            404: function() {alert('Page not found')},
            405: function() {alert('Method forbidden')},
            500: function
    (jqXHR, textStatus, errorThrown) {alert($.toJSON(jqXHR.responseJSON, null, 2))}
        },
        error: function(jqXHR, textStatus, errorThrown){
            if (textStatus==='timeout') {alert('Too late :(')}
        }
    });

})(jQuery);

$(function(){

});

</script>
</head>
<body>
</body>
</html>>
q $$$OK
}

}

```

Let 's add a table and a corresponding method returning JSON data:

```

...
<body>
    <table id="dg" class="easyui-datagrid" title="DataGrid" style=
"width:500px;height:350px" data-options="url:'getsqljson'">
        <thead>
            <tr>
                <th data-options="field:'ID',width:60,align:'center'">ID</th>
                <th data-options="field:'idp',width:60,align:'center'">Parent</th>
                <th data-options="field:'name',width:160,align:'right'">Name</th>
            </tr>
        </thead>
    </table>
</body>
...

ClassMethod SrvGetSQLJSON() As %Status
{
    d ##class(%ZEN.Auxiliary.jsonSQLProvider).%WriteJSONFromSQL("select * from my.a")
    q $$$OK
}

...
<Routes>

```

```
<Route Url="/getsqljson" Method="POST" Call="SrvGetSQLJSON"/>
...
```

If we try refreshing the page, along with our datagrid, in the debugging mode now, we will see an error in the console.

This happens because JSON data is returned to the “ children ” field that jeasyui is not aware of.

So what do we do with it?

We ’ ll rewrite the default loader that we will place immediately after **\$.ajaxSetup**:

```
...
function getJsonLoader(pluginName){
    return function(param, success, error){
        var opts = $(this)[pluginName]('options');
        if (!opts.url) return false;
        $.ajax({
            url: opts.url,
            data: param,
            success: function(data){
                success(data.children);
            },
            error: function(){
                error.apply(this, arguments);
            }
        });
    }
};

$.fn.datagrid.defaults.loader = getJsonLoader('datagrid');

})(jQuery);
```

Now, the data is both loaded and shown.

That ’ s good, but there are situation when we need to load partial data or update it from the server upon the user ’ s request.

Let ’ s create a button that will partially update our data. To do this, we will need to rewrite the request for invoking input parameters and add the button itself. But it ’ s not going to be your regular button...

```
ClassMethod SrvGetSQLJSON() As %Status
{
    s p1=##class(%ZEN.Auxiliary.parameter).%New()
    s p2=##class(%ZEN.Auxiliary.parameter).%New()
    s p1.value=1
    s p2.value=10

    if $IsObject(%request.Content) {
        d ##class(%ZEN.Auxiliary.jsonProvider).%ConvertJSONToObject(%request.
Content,,.params)

        s:params.p1'="" p1.value=params.p1
        s:params.p2'="" p2.value=params.p2
    }
}
```

```
s p=##class(%ZEN.Auxiliary.jsonSQLProvider).%New()
s p.sql="select %ID,idp,name from my.a where %ID between ? and ?"
d p.parameters.SetAt(p1,1)
d p.parameters.SetAt(p2,2)
s p.%Format=""
s p.maxRows=0
d p.%DrawJSON()

q $$$OK
}

var MyApp={
  toolbar: [{
    id:'tbReload',
    text:'Refresh',
    iconCls:'icon-reload',
    handler:function(){$('#dg').datagrid('load', {p1:2,p2:6})}
  ]
};

<table id="dg" class="easyui-datagrid" title="DataGrid" style=
"width:500px;height:350px" data-options="url:'getsqljson',toolbar:MyApp.toolbar">
```

From now on, when the page is loaded, it will display records with id [1,10]. When the button is pressed, it will display records with id [2,6].

Important:

Always use parametrized requests, not a merge of the request text, or the the code will be vulnerable to SQL injections.
--

Tree

Many things are identical here to what was described above, although some differences do exist.

To begin with, let 's add a tree to our page and redefine the loader:

```
...
</table>
<div title="Tree" style="width:300px;height:350px">
  <ul id="tree" class="easyui-tree" data-options="url:'getsqljson'"></ul>
</div>
</body>
...

...
$.fn.datagrid.defaults.loader = getJsonLoader('datagrid');
$.fn.tree.defaults.loader = getJsonLoader('tree');
...
```

We already see something, but it 's not quite what we were expecting.

This happened because the **makeTree** method wasn 't invoked in the loader. Let 's fix it (it is assumed that it 's found in the MyApp object):

```
...
success: ((pluginName=='tree')||(pluginName=='treegrid'))?
  function(data){
```

```
    success(MyApp._makeTree({q:data.children, id:'ID', parentid:'idp'}));
  }:
  function(data){
    success(data.children);
  },
  ...
```

The structure of the tree is displayed correctly now, but the node text isn't.

According to the [tree documentation](#), we are supposed to return the following fields:

- Every node contains following properties:
- id: node id, which is important to load remote data
- text: node text to show
- state: node state, 'open' or 'closed', default is 'open'. When set to 'closed', the node has children nodes and will load them from remote site
- checked: Indicate whether the node is checked selected.
- attributes: custom attributes can be added to a node
- children: an array of nodes defines some children nodes

That is, we need to replace the "name" field by "text" and "ID" by "id".

That's easy:

```
...
success: ((pluginName=='tree')||(pluginName=='treegrid'))?
  function(data){
    success(JS
ON.parse($.toJSON(csprest._m
akeTree({q:data.children, id:'ID', parentid:'idp'})), function(k, v) {
  if (k === "ID")
    this.id = v;
  else if (k === "name")
    this.text = v.replace('<br>', '&nbsp;');
  else
    return v;
}));
  }:
  function(data){
    success(data.children);
  },
  ...
```

The tree is now displayed exactly the way it is supposed to be.

Let's make our button also load data for the tree upon our request:

```
...
handler:function(){
  $('#dg').datagrid('load', {p1:2,p2:6});
  $('#tree').tree({queryParams:{p1:1,p2:7}});
}
...
```

[myrestfull.zip](#) with Easter eggs (see comments)

\$(REST - CSP - (-HyperEvents) + EasyUI + File Upload). Part 5

Published on InterSystems Developer Community (<https://community.intersystems.com>)

[#REST API](#) [#UI Development](#) [#Frontend](#) [#Caché](#)

Source URL: <https://community.intersystems.com/post/rest-csp-hyperevents-easyui-file-upload-part-5>