Using the JQuery dataTables plugin to display dynamic data in tables: part 1

Posted At : 25 February 2010 12:26 | Posted By : Shaun McCran
Related Categories: JQuery, Coldfusion, RIA, Json, AJAX

Rather than writing out long winded table code to display your data in a tabulated fashion why not use the dataTables JQuery plugin to do it for you?

In this blog entry I'll be generating tables using a JQuery plugin, but I will also be generating the JQuery code from an XML document.

The theory behind the dataTables JQuery plugin is that when the template loads it makes an AJAX request to a remotely specified template. That template returns a JSON object of data which is formatted and used in a tabular display. This means that you can perfom filtering and sort functions inline, and the JQuery simply re submits the AJAX request, receiving new JSON each time. So no refreshing.

I'll be dealing with the auto generation of the JSON back end in the second part of this article. Here is how I setup the tabular display.

Build a standard html template, including the CSS and JQuery plugins.

```
PE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/transitiona
                                <html lang="en-GB">
                                    <head>
validTag http-equiv="Content-Type" content="text/html; charset=iso-8859-1" lang="e
                <InvalidTag name="language" content="en-GB">

                <style type="text/css" title="currentStyle">
                                @import "demo_page.css";
                                @import "demo_table.css";
                                    </style>

'text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.mi
ript type="text/javascript" language="javascript" src="jquery.dataTables.js"></scr
```

Next we build our JQuery function. I wont go into massive detail about all the parameters and values being passed in here, but it is well documented on http://www.datatables.net/ .

```
            <s/cript type="text/javascript" charset="utf-8">
                    $(document).ready(function() {
                                    $('#example').dataTable( {
                                            "bProcessing": true,
                                            "bServerSide": true,
                                       "sAjaxSource": "content.cfm",
                                               "aoColumns": [
          { "sName": "Edit", "sTitle": "Edit", "sWidth":"10%"} ,
                  { "sName": "Band", "sTitle": "Band"} ,
                 { "sName": "Genre","sTitle": "Genre"} ,
                        { "sTitle": "Fake Column"}
                                    ],
                                        "sPaginationType": "full_numbers",
                                            "aaSorting": [[1,'asc']],
                                                 "oLanguage": {
                                       "sLengthMenu": "Page length: _MENU_
                                                "sSearch": "Filter:"
                                                      }
                                                } );
                                            } );
                                </script>
```

The really important column here is the "aoColumns" JSON data block. This specifies what fields are returned from your AJAX call, and parameters they must adhere to.

In this example we are anticipating that we will receive four columns of data back (Edit,Band,Genre and Fake Column).

Lastly we create a table with an ID of "example", as this is what the JQuery is looking for. This table must be formatted in a certain way, as the JQuery plugin will re write the specified elements.

```
<table cellpadding="0" cellspacing="0" border="0" class="display" id="example">
                                    <thead>
                                      <tr>
```

```
                                    <th>Edit</th>
                                    <th>Band</th>
                                    <th>Genre</th>
                                <th>Fake Column</th>
                                    </tr>
                                  </thead>
                                    <tbody>
                                      <tr>
      <td colspan="3" class="dataTables_empty">Loading data from server</td>
                                    </tr>
                                  </tbody>
                                </table>
```

The code also contains the header elements that will match the returned column values from the AJAX request. The last part of the table is displayed when the data is loading.

This all works well, but to extend it further I have altered the code to read from an XML document. The XML document Is loaded when the template starts, and the data fields and attributes are read, and looped over to create the JQuery code and table headers. In this way it is a generic table display template, driven from an XML document.

The XML doc:

```
                            <?xml version="1.0"?>
                                    <form>
    <field sName="Edit" source="data" sTitle="Edit" sWidth="10%">Edit</field>
          <field sName="Band" source="data" sTitle="Band">Band</field>
        <field sName="Genre" source="data" sTitle="Genre">Genre</field>
  <field sName="fake column" source="" sTitle="Fake Column">Fake column</field
                                    </form>
```

Read the XML file and parse it out into an Array:

```
                        <!--- parse the xml file --->
                <cfset variables.xml = XMLParse("test.xml") />
    <!--- <cfdump var="#variables.xml#" label="Raw xml document"> --->
```

```
<cfset variables.fields = XMLSearch(variables.xml,"form/field")>
  <cfset variables.totalRecords = ArrayLen(variables.fields)>
```

Use something like this to dynamically generate the JQuery and table values:

```
                                    <cfoutput>
  <cfloop index="variables.index" from="1" to="#ArrayLen(variables.fields)#">
les.index].XmlAttributes.source EQ "data">"sName": "#variables.fields[variables.in
lds[variables.index].XmlAttributes, 'sTitle')>"sTitle": "#variables.fields[variabl
.ds[variables.index].XmlAttributes, 'sWidth')>,"sWidth":"#variables.fields[variable
            } <cfif variables.index NEQ variables.totalRecords>,</cfif>
                                    </cfloop>
                                   </cfoutput>
                            <!--- table values --->

                                   <cfoutput>
  <cfloop index="variables.index" from="1" to="#ArrayLen(variables.fields)#">
        <th align="left">#variables.fields[variables.index].XmlText#</th>
                                     </cfloop>
                                   </cfoutput>
```

The JSON response is hard coded in this example, so the result will not filter or search. I'll handle that in article two.

There is a full example of this here .