

JQueryDataTablesForMVC

- Overview
 - Plugin Description
- Basic Usage
 - Almost Zero Config
 - Inline Editable Table
 - Deletable Table
 - Dialog Editable Table
 - Creatable Table (Adding records)
- Customizing The Table
 - Title
 - Table Width
 - Disable Scroller Plugin
 - Conditional Table Data Values
 - Specifying Row Id for each row to store Id to pass to actions
 - Specifying Table Id for each table to store Id to pass to actions
 - DropDownList Editor
 - Styling the table
 - Custom Dialog Buttons
 - Custom Table Sorting

Overview:

Plugin Description:

JQueryDatatables ForMVC is a c# plugin that I created because I was using jQuery datatables in my projects quite often. I liked how using jQuery datatables sped up my project making it robust and fast. In certain circumstances speeding up page load to 4 seconds from nearly 2 minutes. Anyways I was getting tired of writing all of the javascript over and over again. So I wrote this plugin. The plugin allows you to define a C# object and pass it to your view. In the view all you need do is reference the object and voila you have a datatable. Editable with multiple UI options. and no need to write a bunch of clientside javascript.

Basic Usage:

Almost Zero Config:

To get started using the plugin you must create an action method to display your view.

```
public ActionResult Test()
{
    return View();
}
```

You will also need an action that will return Json data to the datatable. For simplicity I just created a small array of anonymous objects and returned them in a JsonResult... But in actuality you will probably retrieve a list of objects from your database.

```
public ActionResult GetTestData()
{
    object[] data = new[] {
        new {
            EmployeeName = "Airi Satou",
            EmployeePosition = "Accountant",
            OfficeLocation = "Tokyo",
            EmployeeAge = 33,
            Salary = "$162,700"
        },
        new {
            EmployeeName = "Angelica Ramos",
            EmployeePosition = "Chief Executive Officer (CEO)",
            OfficeLocation = "London",
            EmployeeAge = 47,
            Salary = "$1,200,000"
        },
        new {
            EmployeeName = "Ashton Cox",
            EmployeePosition = "Junior Technical Author",
            OfficeLocation = "San Francisco",
            EmployeeAge = 66,
            Salary = "$86,000"
        }
    };
    return Json(data, JsonRequestBehavior.AllowGet);
}
```

Next we will reference the JQueryDataTablesForMVC namespace at the top of the controller.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using JQueryDataTablesForMVC;
```

Once that is done you can now start to create a new JQueryDataTable object to be passed to the view.

Below is the simplest implementation of the object. You must specify a table name, AjaxDataSource, and Column Definition.

```
public ActionResult Test()
{
    var jqueryDataTable = new JQueryDataTable(
        "TestTable",
        Url.Action("GetTestData"),
        new List<Column>
        {
            new Column("Name", "EmployeeName"),
            new Column("Position", "EmployeePosition"),
            new Column("Location", "OfficeLocation"),
            new Column("Age", "EmployeeAge"),
            new Column("Salary", "Salary")
        });
    return View(jqueryDataTable);
}
```

Notice the second parameter for each Column Object Constructor. The data parameter must match the object property name in your data source. The first parameter defines what the column header will display.

The next thing you must do is create a view that shows the datatable.

```
@model JQueryDataTablesForMVC.JQueryDataTable
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Test</title>
    <link rel="stylesheet" type="text/css" href="@Url.Content("~/Content/css/jquery-ui-1.10.4.custom.min.css")" />
    <link rel="stylesheet" type="text/css" href="@Url.Content("~/Content/DataTables-1.9.4/media/css/jquery.dataTables_themeroller.css")" />
    <script src="@Url.Content("~/Scripts/jquery-1.10.2.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/jquery-ui-1.10.4.custom.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/jquery.dataTables.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/dataTables.scroller.js")" type="text/javascript"></script>
</head>
<body>
    <div>
        @((Model))
    </div>
</body>
</html>
```

Above shows the minimal script requirements for this plugin to work.

Also notice how the body of the view only has a reference to the model. Since the JQueryDataTable object extends IHtmlString it can utilize the ToHtmlString() method in the view. If you have a complex view model you can set the datatable as a property of that view model and just reference that as well.

```
@(ViewModel.DataTable)
```

The Rendered datatable will now look like below:

Search: <input type="text"/>				
Name	Position	Location	Age	Salary
Airi Satou	Accountant	Tokyo	33	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	\$86,000

Showing 1 to 3 of 3 entries

Inline Editable Table:

To Create an Inline Editable Table you just create the table in the same way as before but with a few properties set differently from the default.

```
public ActionResult Test()
{
    var jqueryDataTable = new JQueryDataTable(
        "TestTable",
        Url.Action("GetTestData"),
        new List<Column>
        {
            new Column("Name", "EmployeeName"),
            new Column("Position", "EmployeePosition"),
            new Column("Location", "OfficeLocation"),
            new Column("Age", "EmployeeAge"),
            new Column("Salary", "Salary")
        }
    );
    {
        Editable = true,
        UpdateUrl = Url.Action("UpdateTestData"),
        Params = new List<TableParameter>{
            new TableParameter{ ColumnNumber = 1, Editable = true, Name = "name", IsAddParam = true },
            new TableParameter{ ColumnNumber = 2, Editable = true, Name = "position" },
            new TableParameter{ ColumnNumber = 3, Editable = true, Name = "location" },
            new TableParameter{ ColumnNumber = 4, Editable = true, Name = "age" },
            new TableParameter{ ColumnNumber = 5, Editable = true, Name = "salary" },
        }
    };
    return View(jqueryDataTable);
}
```

You must set the Editable property to true, specify an UpdateUrl (UpdateAction), and specify all parameters that need to be kept track of and sent to the UpdateAction.

When configuring the table parameters you must set the ColumnNumber to tell the template which Column that the parameter will get and set data from and to respectively. The

ColumnNumber property is NOT Zero Based. If you want to be able to edit and change the stored data in the database for that column then you also need to set its Editable property to true. This will tell the template to create an editor for this field. and last you must give the parameter a name. this is so that the data will be able to be passed to the controller update action. Parameter names must match the names of the parameters in the specified Action. Below is a sample update action method signature. ***Notice that the parameter names match the names in the list of parameters set in the JQueryDataTable Object**

```
public ActionResult UpdateTestData(string name, string position, string location, int age, string salary)
{
    /* Update Code Goes Here */

    return null;
}
```

The datatable should now look like this when the View is rendered with an Edit link specified for each row. when clicked the row will highlight and all “Editable” Columns will show in their specified editors. When the user clicks cancel the row will unhighlight. When the user clicks update the values from the specified parameter columns will be sent to the Update Action. upon successful update a success message will show.

Search: <input type="text"/>					
Name	Position	Location	Age	Salary	
Airi Satou	Accountant	Tokyo	33	\$162,700	Edit
Angelica Ramos	Chief Executive Officer (CEO)	London	47	\$1,200,000	Edit
<input type="text" value="Ashton Cox"/>	<input type="text" value="Junior Technical Author"/>	<input type="text" value="San Francisco"/>	<input type="text" value="66"/>	<input type="text" value="\$86,000"/>	Update Cancel

Showing 1 to 3 of 3 entries

✓ Alert: Record Updated

Deletable Table:

You can also easily make the rows deletable by specifying a “DeleteUrl” or (Delete Action) and setting the tables “Deletable” property to true.

```
public ActionResult Test()
{
    var jQueryDataTable = new JQueryDataTable(
        "TestTable",
        Url.Action("GetTestData"),
        new List<Column>
        {
            new Column("Name", "EmployeeName"),
            new Column("Position", "EmployeePosition"),
            new Column("Location", "OfficeLocation"),
            new Column("Age", "EmployeeAge"),
            new Column("Salary", "Salary")
        })
    {
        Editable = true,
        UpdateUrl = Url.Action("UpdateTestData"),
        Deletable = true,
        DeleteUrl = Url.Action("DeleteTestData"),
        Params = new List<TableParameter>{
            new TableParameter{ ColumnNumber = 1, Editable = true, Name = "name", IsAddParam = true },
            new TableParameter{ ColumnNumber = 2, Editable = true, Name = "position" },
            new TableParameter{ ColumnNumber = 3, Editable = true, Name = "location" },
            new TableParameter{ ColumnNumber = 4, Editable = true, Name = "age" },
            new TableParameter{ ColumnNumber = 5, Editable = true, Name = "salary" },
        }
    };
    return View(jQueryDataTable);
}

public ActionResult DeleteTestData(string name, string position, string location, int age, string salary)
{
    /* Remove row from database Code Goes Here */

    return null;
}
```

The datatable now has a Delete Link showing and when clicked displays a modal confirm dialog asking if you are sure that you would like to delete the specified row.

Name	Position	Location	Age	Salary	Edit	Delete
Airi Satou	Accountant	Tokyo	33	\$162,700	Edit	Delete
Angelica Ramos	Chief Executive Officer (CEO)	London	47	\$1,200,000	Edit	Delete
Ashton Cox	Junior Technical Author	San Francisco	66	\$86,000	Edit	Delete



Upon clicking delete or cancel appropriate success or error messages will show to the user.

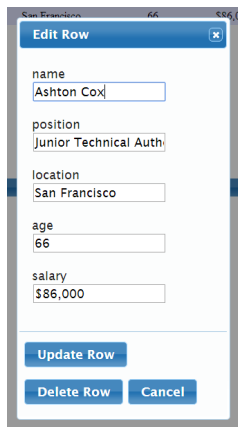
Dialog Editable Table:

To implement the Dialog Interface you must do two more things. You must make the table “Selectable” by setting the “SelectableRows” property to true. this will make the row change color on mouse over and trigger the popup on click. you also must change the tables default editable interface from Interface.Inline to Interface.Dialog. this is done to the previous Test Table by adding two more properties in the JQueryDataTable object construction:

```
new TableParameter{ ColumnNumber = 3, Editable = true, Name = "age" },  
new TableParameter{ ColumnNumber = 4, Editable = true, Name = "salary" }  
},  
SelectableRows = true,  
EditableTableInterface = Interface.Dialog  
};  
return View(jQueryDataTable);
```

Name	Position	Location	Age	Salary
Airi Satou	Accountant	Tokyo	33	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	\$86,000

When the user clicks on a row the Edit Row Dialog pops up.



San Francisco 66 \$86,000

Edit Row

name
Ashton Cox

position
Junior Technical Auth

location
San Francisco

age
66

salary
\$86,000

Update Row

Delete Row Cancel

Creatable Table:

Along with Updating and deleting records for your table you can also create records. All you need do is set the “Creatable Property” to true and specify a URL or Action to handle the data and add a record to your database. You will also need to tell the plugin which parameters are to be used for adding a record. This is easily done when declaring your parameters. for simplicity I will update the name parameter and set its “IsAddParam” property to true.

```
Editable = true,
UpdateUrl = Url.Action("UpdateTestData"),
Deletable = true,
DeleteUrl = Url.Action("DeleteTestData"),
Creatable = true,
CreateUrl = Url.Action("CreateTestData"),
Params = new List<TableParameter>{
    new TableParameter{ ColumnNumber = 1, Editable = true, Name = "name", IsAddParam = true },
    new TableParameter{ ColumnNumber = 2, Editable = true, Name = "position" },
    new TableParameter{ ColumnNumber = 3, Editable = true, Name = "location" },
    new TableParameter{ ColumnNumber = 4, Editable = true, Name = "age" },
    new TableParameter{ ColumnNumber = 5, Editable = true, Name = "salary" },
}
```

Name	Position	Location	Age	Salary
Airi Satou	Accountant	Tokyo	33	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	\$86,000

Showing 1 to 3 of 3 entries

Add Record

Notice that you now have an Add Record button at the bottom of your table. When clicked it will show the user a modal form that allows them to enter the required data (specified in Params with “IsAddParam” property) to add a record to your database.

Add Record to TestTable?

All Fields Are Required!

name:

Create Record Cancel

Customizing the Table:

Title:

Throughout the template you will see many references to the Name of the table. One example is the header on the modal delete confirmation popup.



The Template will grab the name of the table and stick it in the appropriate spots throughout unless the Title attribute is set, then the template will use the Title attribute for reference. This makes it super easy to customize the way that your table name is displayed.

```
//  
{  
  Title = "Custom Table Name",  
  Editable = true,  
  ...  
}
```

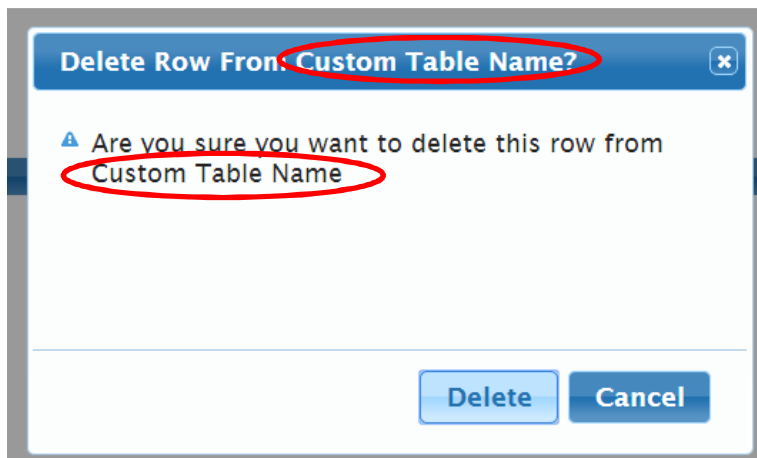
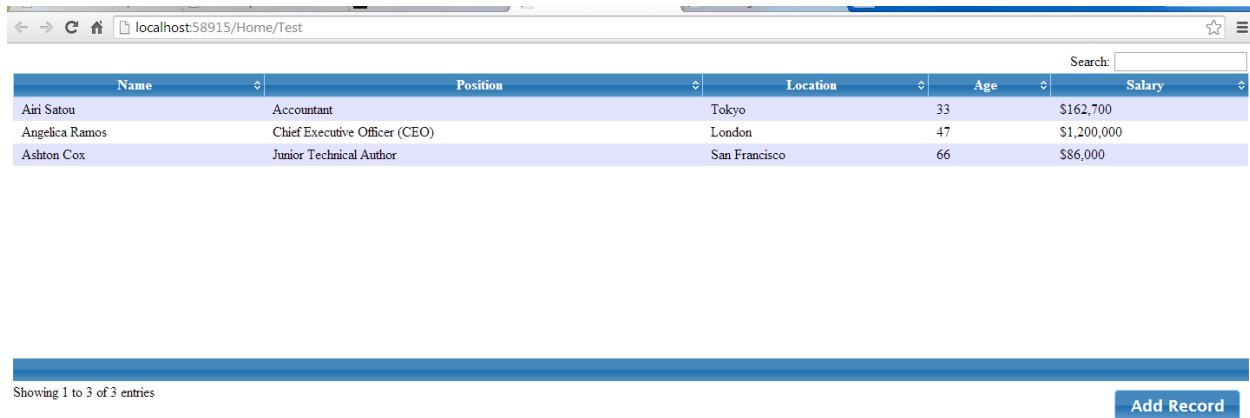


Table Width:

The table width defaults to 70% of the containing element. you can easily change this by specifying a value for this property in the object construction.

TableWidth = "100%"



Name	Position	Location	Age	Salary
Airi Satou	Accountant	Tokyo	33	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	\$86,000


Showing 1 to 3 of 3 entries

Add Record

Disable Scroller plugin:

Disable the Scroller plugin by setting the table's "Scrollable" property to false.

```
    "function(source){ var sal = Number(sou
    })
    {
        Scrollable = false,
        TableDataAttribute = "EmployeeCategory",
        RowDataAttribute = "EmployeeID",
```



Name	Position	Location	Age	Salary	Monthly Salary
Airi Satou	Accountant	Tokyo	33	\$162,700	\$13558
Angelica Ramos	Chief Executive Officer (CEO)	London	47	\$1,200,000	\$100000
Ashton Cox	Junior Technical Author	San Francisco	66	\$86,000	\$7166

Showing 1 to 3 of 3 entries

Conditional Table Data Values:

Sometimes you may need to do a calculation on other fields in the row and display them in its own column this is easily done by creating a column with a javascript function as its data source.

I will just create a simple function to determine the monthly salary of the three employees.

```
new Column("Salary", "Salary"),
new Column("Monthly Salary",
    "function(source){ var sal = Number(source.Salary.replace(/^[^0-9\\.]+/g, '')); return '$'+parseInt(sal / 12);}"
})
{
    Fditable = true.
```

Name	Position	Location	Age	Salary	Monthly Salary
Airi Satou	Accountant	Tokyo	33	\$162,700	\$13558
Angelica Ramos	Chief Executive Officer (CEO)	London	47	\$1,200,000	\$100000
Ashton Cox	Junior Technical Author	San Francisco	66	\$86,000	\$7166

Showing 1 to 3 of 3 entries

Add Record

Specifying RowDataAttribute and RowID for each row:

Sometimes you will need an id of some sort to update or delete rows in your database. Take our existing Delete Action. This takes name, position, location, age, and salary parameters. But perhaps you just want to find the row with a unique identifier and delete that row. this is no problem. You just need to specify the parameter in the Params List and set its "IsRowId" property to true. I'll demonstrate by creating an Id for each of the employees, Specifying an Id parameter and updating the DeleteAction.

First give the Employee's an ID:

```
public ActionResult GetTestData()
{
    object[] data = new[] {
        new {
            EmployeeName = "Airi Satou",
            EmployeePosition = "Accountant",
            OfficeLocation = "Tokyo",
            EmployeeAge = 33,
            Salary = "$162,700",
            EmployeeID = 1,
        },
        new {
            EmployeeName = "Angelica Ramos",
            EmployeePosition = "Chief Executive Officer (CEO)",
            OfficeLocation = "London",
            EmployeeAge = 47,
            Salary = "$1,200,000",
            EmployeeID = 2,
        },
        new {
            EmployeeName = "Ashton Cox",
            EmployeePosition = "Junior Technical Author",
            OfficeLocation = "San Francisco",
            EmployeeAge = 66,
            Salary = "$86,000",
            EmployeeID = 3,
        }
    };
    return Json(data, JsonRequestBehavior.AllowGet);
}
```

Next, Set the RowDataAttribute to point to that field of each employee and create our new TableParameter:

```
public ActionResult Test()
{
    var jQueryDataTable = new jQueryDataTable(
        "TestTable",
        Url.Action("GetTestData"),
        new List<Column>
        {
            new Column("Name", "EmployeeName"),
            new Column("Position", "EmployeePosition"),
            new Column("Location", "OfficeLocation"),
            new Column("Age", "EmployeeAge"),
            new Column("Salary", "Salary"),
            new Column("Monthly Salary",
                "function(source){ var sal = Number(source.Salary.replace(/^[0-9\\.]*/g, '')); return '$'+parseInt(sal / 12);}")
        })
    {
        RowDataAttribute = "EmployeeID",
        Editable = true,
        UpdateUrl = Url.Action("UpdateTestData"),
        Deletable = true,
        DeleteUrl = Url.Action("DeleteTestData"),
        Creatable = true,
        CreateUrl = Url.Action("CreateTestData"),
        Params = new List<TableParameter>{
            new TableParameter{ ColumnNumber = 1, Editable = true, Name = "name", IsAddParam = true },
            new TableParameter{ ColumnNumber = 2, Editable = true, Name = "position" },
            new TableParameter{ ColumnNumber = 3, Editable = true, Name = "location" },
            new TableParameter{ ColumnNumber = 4, Editable = true, Name = "age" },
            new TableParameter{ ColumnNumber = 5, Editable = true, Name = "salary" },
            new TableParameter{ Name = "id", IsRowID = true }
        },
        SelectableRows = true,
        EditableTableInterface = Interface.Dialog,
        TableWidth = "100%"
    };
    return View(jQueryDataTable);
}
```

Now I will just delete all of the other parameters in the delete action and just reference my unique id:

```
public ActionResult DeleteTestData(int id)
{
    /* Remove row from database Code Goes Here */

    return null;
}
```

Specifying TableDataAttribute and a TableId on each row:

Sometimes you might have different tables that will use the same methods or actions for your crud operations. This can be achieved by setting a TableId parameter to be passed to your Actions. the implementation is the same as above. for demonstration I will create a field on each object called employee category and set the TableDataAttribute to point to that field. I will also create a TableParameter called "category".

```
public ActionResult Test()
{
    var jqueryDataTable = new JQueryDataTable(
        "TestTable",
        Url.Action("GetTestData"),
        new List<Column>
        {
            new Column("Name", "EmployeeName"),
            new Column("Position", "EmployeePosition"),
            new Column("Location", "OfficeLocation"),
            new Column("Age", "EmployeeAge"),
            new Column("Salary", "Salary"),
            new Column("Monthly Salary",
                "function(source){ var sal = Number(source.Salary.replace(/^[^0-9\\.]+/g, ''); return '$'+parseInt(sal / 12);}"))
        })
    {
        TableDataAttribute = "EmployeeCategory",
        RowDataAttribute = "EmployeeID",
        Editable = true,
        UpdateUrl = Url.Action("UpdateTestData"),
        Deletable = true,
        DeleteUrl = Url.Action("DeleteTestData"),
        Creatable = true,
        CreateUrl = Url.Action("CreateTestData"),
        Params = new List<TableParameter>{
            new TableParameter{ ColumnNumber = 1, Editable = true, Name = "name", IsAddParam = true },
            new TableParameter{ ColumnNumber = 2, Editable = true, Name = "position" },
            new TableParameter{ ColumnNumber = 3, Editable = true, Name = "location" },
            new TableParameter{ ColumnNumber = 4, Editable = true, Name = "age" },
            new TableParameter{ ColumnNumber = 5, Editable = true, Name = "salary" },
            new TableParameter{ Name = "id", IsRowID = true},
            new TableParameter{ Name = "category", IsTableID = true}
        },
        SelectableRows = true,
        EditableTableInterface = Interface.Dialog,
        TableWidth = "100%"
    };
    return View(jqueryDataTable);
}
```

```

public ActionResult GetTestData()
{
    object[] data = new[] {
        new {
            EmployeeName = "Airi Satou",
            EmployeePosition = "Accountant",
            OfficeLocation = "Tokyo",
            EmployeeAge = 33,
            Salary = "$162,700",
            EmployeeID = 1,
            EmployeeCategory = "test"
        },
        new {
            EmployeeName = "Angelica Ramos",
            EmployeePosition = "Chief Executive Officer (CEO)",
            OfficeLocation = "London",
            EmployeeAge = 47,
            Salary = "$1,200,000",
            EmployeeID = 2,
            EmployeeCategory = "test"
        },
        new {
            EmployeeName = "Ashton Cox",
            EmployeePosition = "Junior Technical Author",
            OfficeLocation = "San Francisco",
            EmployeeAge = 66,
            Salary = "$86,000",
            EmployeeID = 3,
            EmployeeCategory = "test"
        }
    };
    return Json(data, JsonRequestBehavior.AllowGet);
}

public ActionResult DeleteTestData(int id, string category)
{
    /* Remove row from database Code Goes Here */

    return null;
}

```

Custom DropDownList Editor:

To create the dropdown list editor for a parameter you need to set the parameter's EditorType property and set the DropDownList property to an array of SelectListItem's.

```

createUrl = Url.Action("CreateTestData"),
Params = new List<TableParameter>{
    new TableParameter{ ColumnNumber = 1, Editable = true, Name = "name", IsAddParam = true },
    new TableParameter{ ColumnNumber = 2, Editable = true, Name = "position" },
    new TableParameter{ ColumnNumber = 3, Editable = true, Name = "location", Editor = EditorTypes.DropDownList,
        DropDownList = new []{
            new SelectListItem{Text = "Tokyo", Value = "Tokyo"},
            new SelectListItem{Text = "London", Value = "London"},
            new SelectListItem{Text = "San Francisco", Value = "San Francisco"}
        }
    },
    new TableParameter{ ColumnNumber = 4, Editable = true, Name = "age" },
    new TableParameter{ ColumnNumber = 5, Editable = true, Name = "salary" },
    new TableParameter{ Name = "id", IsRowID = true},
    new TableParameter{ Name = "category", IsTableID = true}
},
SelectableRows = true,

```

This will render the following dropdownlist for each Location field in the table:

```
<select name="location" id="location">
  <option value>Select</option>
  <option value="Tokyo">Tokyo</option>
  <option value="London">London</option>
  <option value="San Francisco">San Francisco</option>
</select>
```

Styling the Table:

I understand that you might want to style your table in your own way to match a previous style that is on your current site. All styling for the table can be done through the “TableDrawCallback” and “RowDrawCallback” properties. these properties store an object of type “JavascriptFunction”. If you plan on styling more than one datatable with a custom style via these properties then I suggest creating an extension method that you can call on the datatable itself after initialization. Below is a sample:

```
public static JQueryDataTable CustomStyle(this JQueryDataTable jqdt)
{
    jqdt.TableDrawCallback = new JavascriptFunction
    {
        ImmediateInvoke = true,
        Body = new[] {
            "$('table').css({'color':'#333333','border-collapse':'collapse'});",
            "$('#"+jqdt.Name+"DataTableWrapper th, .tfooter').css({'background':'#20548e','color':'white', 'cursor':'pointer'});"
        }
    };

    jqdt.RowDrawCallback = new JavascriptFunction
    {
        IsBodyOnly = true,
        Body = new[] {
            "if($(nRow).hasClass('odd')){",
            "$$(nRow).css({'background-color':'#eff3fb','color':'#333333'});",
            "} else {",
            "$$(nRow).css({'background-color':'white','color':'#333333'});"}
    };

    return jqdt;
}
```

Above I create an extension method that sets the drawcallback properties of the passed datatable object to specific styling javascript functions.

Next in the controller you would just call this method after the datatable has been initialized and before you pass it to the view.

```
MyDataTable.CustomStyle();
```

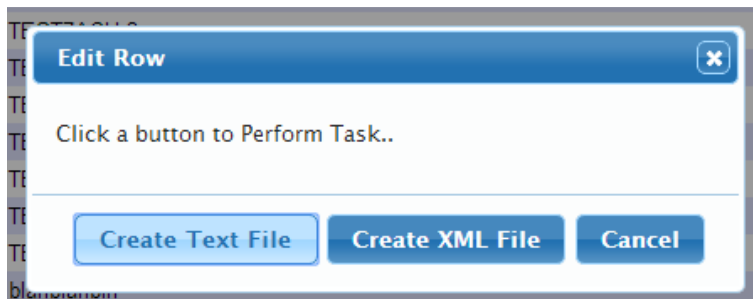
Creating Custom Dialog Buttons:

Creating custom dialog buttons for your edit row dialog is easy. for example I had a project that I needed the user to be able to click on a row and then generate a text report or xml report for that specific record.

The CustomButtons property holds a Dictionary<string,string> object. the key of the dictionary object is the button text. and the value of the dictionary object is its javascript function. below is a sample implementation of custom buttons:

```
CustomButtons = new Dictionary<string, string> {
    {
        "Create Text File",
        new JavascriptFunction() {
            Anonymous = true,
            Body = new[] {
                "window.location='/Home/GetTextFile/?id=' + $('#NBIIHistoryRowEditorModal').data('id');",
                "$(this).dialog('close');",
                "NBIIHistoryShowMessage('Text File Download Started!',true);"
            }
        }.ToString()
    },
    {
        "Create XML File",
        new JavascriptFunction() {
            Anonymous = true,
            Body = new[] {
                "window.location='/Home/GetXMLFile/?id=' + $('#NBIIHistoryRowEditorModal').data('id');",
                "$(this).dialog('close');",
                "NBIIHistoryShowMessage('Text File Download Started!',true);"
            }
        }.ToString()
    }
},
Creatable = true,
createUrl = Url.Action("AddNBIIHistory", "Home"),
EditableTableInterface = Interface Dialog
```

The above code renders the following buttons in the view:



Creating Custom Row Sorting:

To define a custom sorting of your datatable I recommend reading up on how jQuery Datatables javascript api does this at their website. That being said, here is a sample of the use that I had for it.

Defining custom row sorting is done in the Column definition.

```
new List<Column>{  
    new Column(  
        "Report Date",  
        new JavascriptFunction()  
        {  
            Anonymous = true,  
            Body = new []{  
                "return new Date(parseInt(source.ReportDate.substr(6))).toLocaleDateString();" |  
            },  
            Parameters = new[] { "source" }  
        }.ToString(),  
        null,  
        "desc"  
    ),  
}
```

This Column has the heading "Report Date", and its data is a new javascript function that takes the Json date in the source and converts it to a javascript date and calls a toString method on it. The next parameter is the width which I did not need for this specific column so I just set to null and the last is the sorting. (which is set to descending)

Now all rows will be sorted by the report date in descending order.