

# Flexicious HTMLTreeGrid

## Table of contents

---

Summary .....	4
System Requirements .....	4
Introduction .....	4
Grid Architecture .....	6
Getting Started - Simple Example .....	10
jQuery Sample .....	10
Ext-JS Sample .....	11
DOJO Sample .....	13
Details of the Markup .....	15
Common Configuration Options .....	17
Basic Configuration Options .....	17
Configuration Options - Filters .....	22
Configuration Options - Footers .....	25
Configuration Options - Paging .....	30
Configuration Options - Print & Export .....	31
Configuration Options - Preferences/Settings .....	33
Configuration Options – Column Formatting .....	35
Configuration Options – Interactive HTML in cells .....	46
Styling and Theming .....	48
Styling Using CSS .....	48
Styling Using Markup and API .....	48
Creating your own theme .....	51
Connecting To Data .....	62
Basic Concepts .....	62
Flat Data .....	62
Flat Data - Lazy Load - Dot Net Sample .....	63
Hierarchical Data .....	69
Hierarchical Data - Lazy Load - Java and PHP Sample .....	73
Beyond the Basics .....	79
OO Concepts for JavaScript Developers .....	79
flexiciousNmsp.TypedObject Class .....	79
flexiciousNmsp.UIComponent Class .....	82
Configuration - XML vs API .....	84
XML Configuration .....	84
API Configuration .....	86
Event Handlers and Function Callbacks .....	89
Events .....	89
Function Callbacks .....	98
Cells and Renderers .....	100
Recycling, Virtualization and Buffering .....	102
Custom Renderers .....	104
Class Factories .....	109
Filter Options - Deep Dive .....	111
Filter Page Sort Mode .....	111
Built in filter controls .....	112
Built in filter control options .....	113
Text Input Filter .....	113

Multi Select Combo Box Filter .....	116
Date Combo Box Filter .....	117
Other Controls .....	119
Anatomy of a filter control .....	119
Sample Custom Filter Control .....	122
External Filters and Filter Functions .....	123
Export Options - Deep Dive .....	124
Built in Exporters .....	124
Echo URL .....	125
Export with Server Paging .....	127
Custom Export Popup .....	128
Custom Exporter .....	129
Preference Persistence .....	134
Introduction .....	134
Built in features .....	134
Persisting Preferences on the Server .....	136
Customizing the Preference Popups .....	143
Hierarchical Grids .....	144
Paging Options & Toolbar .....	148
Angular JS Integration .....	148
Miscellaneous .....	154
Advanced Configuration Options – Single Level Grids .....	154
Advanced Configuration Options – Hierarchical Grids .....	159
Setting Column Widths .....	161
Selected Key Field .....	163
Row and Column Span .....	163
Localization .....	167
Next Steps & Further Reading .....	174

## Summary

---

Flexicious HTMLTreeGrid (henceforth referred to as the grid) is a powerful DataGrid designed for the HTML5 platform. It is built to render highly complex, Hierarchical/Nested as well as Flat Data in JQuery, DOJO, and Sencha EXT-JS based HTML5/JavaScript Line of Business RIA applications, with a massive set of features Right out Of the Box. Inline Filters, Summary Footers, Fully Customizable Server/Client Paging, DataGrid Print, Excel Export, Word Export, and User Settings Persistence, Smooth Scroll, Nested hierarchical Tree/Child grids, Right Locked columns, built in Lazy Load support, and a whole lot more. A full list of features can be found here: <http://www.htmltreegrid.com/Home/Features>.

## System Requirements

---

Internet Explorer 8+	Chrome 11+	Firefox 6+	Safari 5.1+	Opera 11+
-------------------------	---------------	---------------	----------------	--------------

## Introduction

---

Note: You can skip this section and get directly to the "Getting Started - Simple Example" topic if you are familiar with our Flex Products.

The HTMLTreeGrid is a JavaScript port of the Flexicious Ultimate DataGrid, which is the most powerful DataGrid component available for Flex application development. For those of you who are not familiar with flex, Flex is a highly productive, open source application framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops, and devices. It provides a modern, standards-based language and programming model that supports common design patterns suitable for developers from many backgrounds.

### Why HTMLTreeGrid?

HTMLTreeGrid is built on the shoulders of giants: There is a flavor of the HTMLTreeGrid built on top of the excellent JQuery, DOJO, and EXT-JS libraries/frameworks. HTMLTreeGrid leverages the underlying framework for a lot of things like dialog boxes, date pickers, item editors, as well as core concepts like component life-cycle management, display list management, event handling framework, and more. By leveraging the framework features for a lot of these dependencies, not only do we reduce the footprint of our Code base, but also make it easier for our customers to integrate our product into their systems, while at the same time benefiting from the many man months of effort that goes into developing these framework pieces.

**A Feature set unmatched by any**: Although these frameworks offer a DataGrid component of their own, none of the available DataGrid components offer all the features that are available in the HTMLTreeGrid in a unified API. The HTMLTreeGrid was specifically designed to handle very complex use cases that involve hierarchical data as well as complex interactions you can perform with such data sets. Some of the examples of such interactions include lazy loading child records that each hierarchical level, recursive selection, selection cascading and bubbling, virtual scroll, built-in support for drill up and drill down, built in support for in-line detail panels, in addition to integrating with regular DataGrid features like in-line filtering, paging, multiple columns sort, grouped as well as left and right locked columns, row span and column span, a number of selection modes like cell, row, multiple cell, multiple row, as well as customizable programmatic cell backgrounds contents borders and colors. The complete list of features is way too long to cover in this single post so please look at the features and the demos to get an idea of what's possible.

**Highly Optimized**: We have gone to great lengths to optimize every single piece of the rendering mechanism. We recycle renderers as you scroll in both directions horizontal as well as vertical. We draw just the visible area for all the sections of the DataGrid including headers footers filters the toolbar as well as the data area. This makes it possible for us to render very large record sets in a very short period of time.



**Enterprise Ready, Proven, and Robust:** Since the HTMLTreeGrid has its roots in the Flexicious ultimate flex DataGrid; it automatically inherits years and years of development, testing, refining as well as the level of robustness that comes from being used in the most demanding enterprise applications ever built. For years flex has been the application development framework of choice for large organizations especially with the Java J2EE stack. There is good reason for it, because a number of the concepts that have been developed in the flex ecosystem make it a very attractive option as the UI technology of choice for large-scale applications. HTMLTreeGrid benefits from inheriting all of these underlying concepts that are baked into its architecture.

**Laser Sharp Focus:** As a company, we have been making a living for years by providing our customers with the most powerful Flex DataGrid component money can buy. Today we are bringing that same level of polish to flex developers moving to JavaScript as well as JavaScript developers in general. We have a team of very talented developers and designers who have thought about the DataGrid component more than what should be considered healthy. We have pondered over use cases, argued over user interaction schematics, listened to our customers, refined and improved our product on basis of what their requirements are, and relentlessly re-factored, redesigned and redeveloped features until they achieve perfection. We're committed to constantly improving our product in pursuit of the highest level of quality.

Below are some of the features of the grid, with new ones being added with each release.

## FEATURES

- Basic DataGrid Features : Ability to organize information as rows and columns, with locked headers, Ability to Customize the appearance Rows and Columns, User Interactive, Draggable, Resizable, Sortable and Editable Columns, Keyboard Navigation and Accessibility Support.
- In-line Filtering, with numerous built in Filter Controls, and extensible architecture to define your own.
- Server and Client Paging, with a fully Customizable Pager UI.
- Summary Footers, with fine-tuned control over Formula, Precision, Formatting, Placement and Rendering of Footers.
- Ability to Export to Excel, Word, Text, XML and other formats. Ability to plug in your own Exporters.
- Preference Persistence (Ability for your end users to save viewing preferences, like column order, visibility, widths, filter criteria, print settings etc.)
- Support for Hierarchical Data with Smooth Scrolling and Nested Tree Grids
- Ability to define Fully Lazy Loaded, Partially Lazy Loaded and Initial Loaded Flat as well as Hierarchical Data Grids.
- Support for Row Span and Column Span.
- Support for Virtual Scroll of Hierarchical Data.
- A vast number of Business Scenarios supported out of the box:
- Fully Configurable Drag and Drop
- Multi Column Sort
- Multi-Level, Grouped Column Headers
- Single Cell, Single Row, Multiple Cell, Multiple Row Selection Modes.
- Display of Homogeneous Hierarchical Data (Single Set Of Columns)
- Checkbox Selection of data, with Tri State Checkbox Header
- Customizable Loading Animation
- Smooth Scrolling
- Display of Heterogeneous Hierarchical Data (Multiple Sets Of Columns)
- Display of Heterogeneous Hierarchical Data (Multiple Sets Of Columns)
- Ability to define paging, filtering and summary footers at each hierarchical level
- Left and Right Locked columns
- Built in Cascading and Bubbling of checkbox/row selection for hierarchical data
- Built in Drill Down/Drill Up/Drill to of Hierarchical data

- Toolbar action icons, with ability to define custom actions
- Built in Ability to define initial sort values at any level
- Built in Ability to show custom ToolTip with interactive content on hover over
- Built in Ability to define custom logic that controls row selection, enabled, background, and border.
- Built in Ability to Auto Adjust height based on rows, as well as prevent multiple scrollbars at hierarchical levels.
- Programmatic control over which cells are editable, which rows are selectable, background, border and text colors of each cell.
- Read write nested properties of complex objects
- Context menu based copying of data rows
- Ability to declaratively define Hand Cursors, Underline, truncateToFit, wordWrap on the columns
- Ability to define various column width modes, like fitToContent, Percentage and Fixed, Automatic column width adjustment on basis of data

## Grid Architecture

The Grid is an extremely versatile and powerful control, able to display both hierarchical, as well as flat data. For Flat data, the grid resembles a table, while for hierarchical data, the grid resembles a tree with columns.

This is what a flat grid looks like:

Items 1 to 50 of 411. Page 1 of 9							
Go to Page: 1							
ID	LegalName of the Organization	Address					Financials
		Lines		Region			Annual Revenue
		Line 1	Line 2	City	State	Country	
				All	All	All	to
20800	3M Co	863 Newark St	Suite #913	Springfield	North Carolina	United States	12,000
20801	Abbott Laboratori...	502 Park Rd	Suite #31	Grand Rapids	North Carolina	United States	51,000
20802	Adobe Systems	448 King Ave	Suite #790	Springfield	Ohio	United States	28,000
20803	Advanced Micro ...	884 West Lane	Suite #776	Springfield	Penn	United States	31,000
20804	Aetna Inc	295 Newark Blvd	Suite #64	Stroudsburch	Ohio	United States	18,000
20805	Affiliated Comput...	957 Gardner Ave	Suite #897	Grand Rapids	Penn	United States	1,000
20806	AFLAC Inc	736 West St	Suite #19	Stroudsburch	New York	United States	51,000
20807	Air Products & Ch...	519 Gardner Blvd	Suite #266	Springfield	North Carolina	United States	25,000
20808	Airgas Inc	573 Newark Lane	Suite #213	Albany	Michigan	United States	42,000
		Count:411.00					
		Avg:\$30,055.79					

With hierarchy, there are two different modes, nested and grouped. Nested has a series of subgrids (grids within grids) while grouped is a traditional tree with a single set of top level columns:

Nested:

Items 1 to 20 of 411. Page 1 of 21										Go to Page: 1										Edit Delete																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
	<input type="checkbox"/>	ID	Legal Name				Annual Revenue				Num Employees				EPS				Stock Price																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
	<input type="checkbox"/>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
▼	<input type="checkbox"/>	20800	3M Co				53,985				42,810				1.44				13.33																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
	<input type="checkbox"/>	Deal Description					Deal Amount					Deal Date																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
▼	<input type="checkbox"/>	Project # 1 - 3M Co - 6/2013					138,573					Jun-26-2013																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
	<input type="checkbox"/>	Invoice Number			Invoice Amount			Invoice Status			Invoice Date			Due Date																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
▼	<input type="checkbox"/>	2080000			45,255			Paid			Feb-04-2014			Mar-06-2014																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
	<input type="checkbox"/>	Line Item Description							Line Item Amount																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
	<input type="checkbox"/>	Professional Services - Jason Bourne														20,949																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
	<input type="checkbox"/>	Professional Services - Tarah Silverman														24,306																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
		Count:2.00														Total:\$45,255																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
▼	<input type="checkbox"/>	2080001			93,318			Transmitted			Oct-01-2012			Oct-31-2012																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
	<input type="checkbox"/>	Line Item Description							Line Item Amount																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
	<input type="checkbox"/>	Professional Services - Jason Bourne														47,413																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
	<input type="checkbox"/>	Professional Services - Kristian Donovan														45,905																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
		Count:2.00														Total:\$93,318																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
		Count:2.00			Total:\$138,573																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
Items 1 to 2 of 2. Page 1 of 1																				Go to Page: 1										Edit Delete																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						

Grouped:

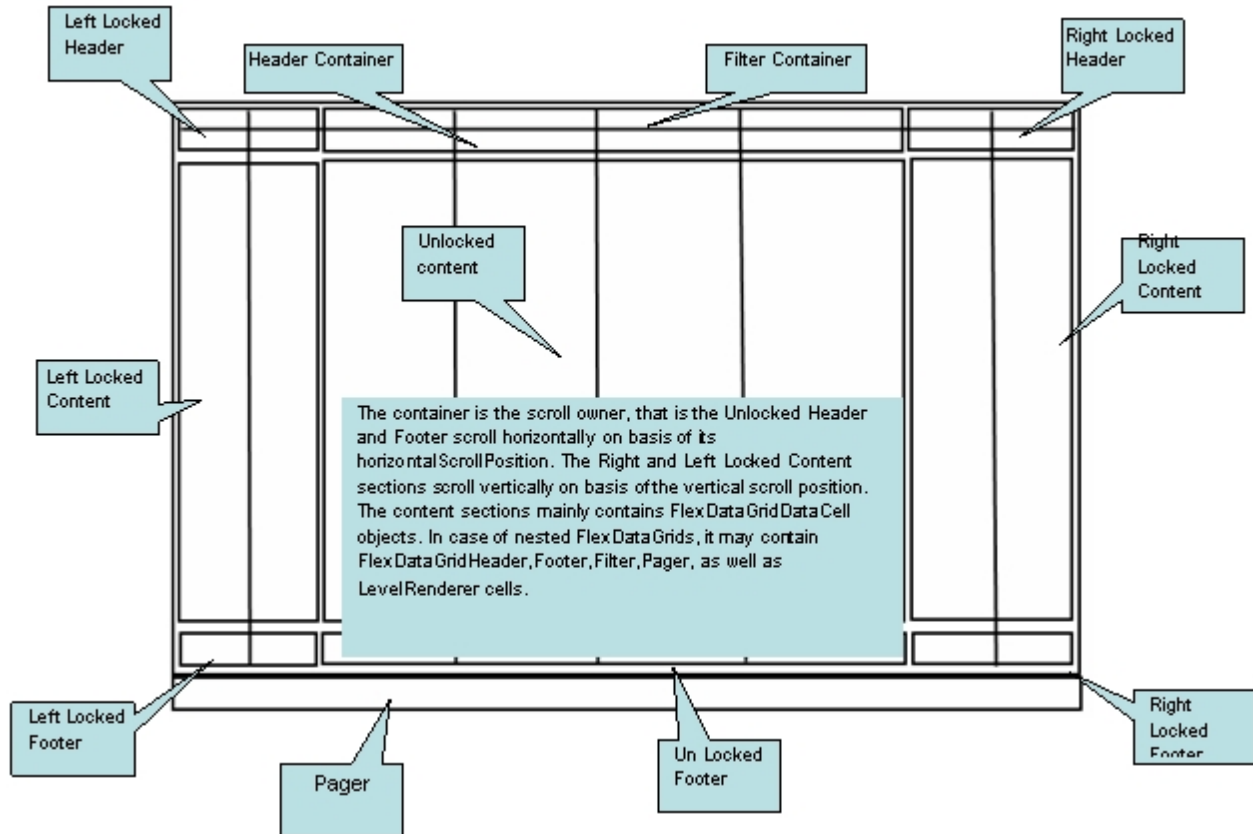
Items 1 to 20 of 411. Page 1 of 21					Go to Page: 1 ▼			
<input type="checkbox"/>	Name	Amount	Invoice Number	Invoice Status	Invoice Date	Due Date		
				All ▼	All ▼	All ▼		
▼ <input type="checkbox"/>	3M Co	242,314						
▼ <input type="checkbox"/>	Project # 1 - 3M Co - 6/2013	138,573						
<input type="checkbox"/>	2080000	45,255		Paid	Feb-04-2014	Mar-06-2014		
<input type="checkbox"/>	2080001	93,318		Transmitted	Oct-01-2012	Oct-31-2012		
		Total:\$138,573	Count:2.00					
Items 1 to 2 of 2. Page 1 of 1					Go to Page: 1 ▼			
▼ <input type="checkbox"/>	Project # 2 - 3M Co - 6/2013	103,741						
<input type="checkbox"/>	2080010	74,248		Approved	Oct-06-2013	Nov-05-2013		
<input type="checkbox"/>	2080011	29,493		Paid	Jul-04-2012	Aug-03-2012		
		Total:\$103,741	Count:2.00					
Items 1 to 2 of 2. Page 1 of 1					Go to Page: 1 ▼			
▼ <input type="checkbox"/>	AFLAC Inc	225,609						
▼ <input type="checkbox"/>	Project # 1 - AFLAC Inc - 7/2014	115,732						
<input type="checkbox"/>	2080600	48,234		Draft	Jun-20-2014	Jul-20-2014		
<input type="checkbox"/>	2080601	67,498		Paid	Jun-02-2014	Jul-02-2014		
		Total:\$115,732	Count:2.00					
Items 1 to 2 of 2. Page 1 of 1					Go to Page: 1 ▼			

One of the most important concepts behind the Architecture of the grid arose from the fundamental requirement that we created the product for - that is display of Heterogeneous Hierarchical Data. The notion of nested levels is baked in to the grid via the "columnLevel" property. This is a property of type "FlexDataGridColumnLevel". This grid always has at least one column level. This is also referred to as the top level, or the root level. In flat grids (non hierarchical), this is the only level. But in nested grids, you could have any number of nested levels. The columns collection actually belongs to the columnLevel, and since there is one root level, the columns collection of the grid basically points to the columns collection of this root level.

The FlexDataGridColumnLevel class has a "nextLevel" property, which is a pointer to another instance of the same class, or a "nextLevelRenderer" property, which is a reference to a ClassFactory the next level. Please note, currently, if you specify nextLevelRenderer, the nextLevel is ignored. This means, at the same level, you cannot have both a nested subgrid as well as a level renderer. Bottom line - use nextLevelRenderer only at the innermost level. Our

examples demonstrate this.

Another one important feature of the product is left and right locked columns, as well as a locked header and footer. This complicates the grid structure a little bit like below:



You can access each of the above container sections via the following properties:

- `leftLockedHeader (com.flexicious.nestedtreedatagrid.cells.LockedContent)` - The container for the left locked filter and header cells. This is an instance of the LockedContent class, which basically is an extended UIComponent that manages the filter and footer cell visibility, heights, and the y positions.
- `leftLockedContent (com.flexicious.nestedtreedatagrid.cells.ElasticContainer)` - The container for the left locked data cells. This is an instance of the ElasticContainer class, which basically attaches to the owner component (which is the bodyContainer) and scrolls vertically along with it. The horizontal scroll of this component is set to off)
- `leftLockedFooter (com.flexicious.nestedtreedatagrid.cells.LockedContent)`
- `filterContainer (com.flexicious.nestedtreedatagrid.cells. FlexDataGridHeaderContainer)`
- `headerContainer (com.flexicious.nestedtreedatagrid.cells.FlexDataGridHeaderContainer)`
- `bodyContainer(com.flexicious.nestedtreedatagrid.cells. FlexDataGridBodyContainer)`
- `rightLockedHeader (com.flexicious.nestedtreedatagrid.cells.LockedContent)`

- rightLockedContent (com.flexicious.nestedtreedatagrid.cells.ElasticContainer)
- rightLockedFooter (com.flexicious.nestedtreedatagrid.cells.LockedContent)
- pagerContainer (com.flexicious.nestedtreedatagrid.cells.FlexDataGridHeaderContainer)

You should normally not need to touch any of these properties. But they're there if you need to. Also, if you need a handle on the actual cells, you should go to the rows collection of filterContainer, headerContainer, bodyContainer, footerContainer, or pagerContainer. Each of these has a rows property, which in case of all but the bodyContainer is collection with 1 Item, of type RowInfo. The rowInfo object, that has the cells collection. The Cells are basically a collection of ComponentInfo object, that each contain a pointer to the actual component which is an instance of one of the subclasses of the FlexDataGridCell object. The FlexDataGridCell has a renderer property, which is the actual component being displayed. These concrete classes that inherit from FlexDataGridCell are:

- FlexDataGridHeaderCell
- FlexDataGridFilterCell
- FlexDataGridFooterCell
- FlexDataGridPagerCell
- FlexDataGridLevelRendererCell
- FlexDataGridExpandCollapseHeaderCell
- FlexDataGridExpandCollapseCell
- FlexDataGridPaddingCell.

Each of the above cells has a renderer property. The renderer is the actual component that is displayed on the UI. The FlexDataGridCell is responsible for sizing, positioning (based on padding), drawing the background, and drawing the borders. In case of the Header, Data or Footer cells the default renderer is a simple Label Control. For Filter, it is an instance of the IFilterControl. For the Pager, it is an IPager control. For the LevelRenderer it is an instance of the Class Factory that you specify in the nextLevelRenderer of the associated column Level. For the ExpandCollapse cells, it will draw an instance of the expand collapse icon on basis of the disclosureIcon style property. All the drawing happens in the drawCell method. It internally calls the drawBackground and drawBorder methods. Usually specifying the style attributes or the cellBackground/rowBackground/cellBorder/rowBorder functions is sufficient, but in case its needed, these methods can be overridden in a custom implementation, and this custom implementation can then be hooked in via the dataCellRenderer, headerCellRenderer, footerCellRenderer, pagerCellRenderer, filterCellRender, expandCollapseHeaderCellRenderer, nestIndentPaddingCellRenderer, and expandCollapseCellRenderer of the column or the level. These are discussed in the "[Cells And Renderers](#)" section

## Getting Started - Simple Example

---

In this section, we will begin with a small example. First, one thing to note is that the grid is built to work with EITHER jQuery, DOJO or EXT-JS. You should already have made the decision as to which framework you want to use prior to starting with this section. jQuery is the most easy to setup, and most of the bits are hosted off our site at [htmltreegrid.com](http://htmltreegrid.com). Angular apps can use the jQuery bits as well.

As with any HTML Grid, the HTMLTreeGrid requires a set of scripts, css imports, and some html markup. Let's take a quick look at a fully self-contained HTMLTreeGrid. Below you will find an html page for the jQuery flavor, EXT-JS flavor, or the DOJO flavor:

### jQuery Sample

```
<!doctype html>
<html lang="en" ng-app="myApp">
<head>
  <meta charset="utf-8">
  <title>Simple</title>

  <!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery-1.8.2.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery-ui-1.9.1.custom.min.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.ui.menu.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.toaster.js"></script>
  <!--End-->

  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
minified-compiled-jquery.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
examples/js/Configuration.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
themes.js"></script>
  <!--End-->
  <!--css imports-->
  <link rel="stylesheet" href="http://htmltreegrid.com/demo/
flexicious/css/flexicious.css" type="text/css"/>
  <link rel="stylesheet" href="http://htmltreegrid.com/demo/
external/css/adapters/jquery/jquery-ui-1.9.1.custom.min.css"
type="text/css"/>
  <!--End-->

  <script type="text/javascript">
```

```

$(document).ready(function(){
    var grid = new
flexiciousNmisp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        dataProvider: [
            { "id": "5001", "type":
"None" },
            { "id": "5002", "type":
"Glazed" },
            { "id": "5005", "type":
"Sugar" },
            { "id": "5007", "type":
"Powdered Sugar" },
            { "id": "5006", "type":
"Chocolate with Sprinkles" },
            { "id": "5003", "type":
"Chocolate" },
            { "id": "5004", "type":
"Maple" }
        ]
    });
});
</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px;width: 100%;">

    </div>
</body>
</html>

```

## Ext-JS Sample

```

<!doctype html>
<html lang="en" ng-app="myApp">
<head>
    <meta charset="utf-8">
    <title>Simple</title>

    <!--These are ext and plugins that we use from ext-->
    <script type="text/javascript" src="http://extjs-
public.googlecode.com/svn/tags/extjs-4.0.7/release/bootstrap.js"></
script>
    <script type="text/javascript" src="http://htmltreegrid.com/
demo/external/js/adapters/ext/Ext.uX.Notification.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/
demo/external/js/adapters/ext/app/app.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/

```

```

external/js/adapters/jquery/jquery-1.8.2.js"></script>
    <!--End-->

    <!--These are specific to htmltreegrid-->
    <script type="text/javascript" src="http://htmltreegrid.com/demo/
minified-compiled-ext.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/
examples/js/Configuration.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/
themes.js"></script>
    <!--End-->
    <!--css imports-->
    <link rel="stylesheet" href="http://htmltreegrid.com/demo/
flexicious/css/flexicious.css" type="text/css"/>
    <link rel="stylesheet" type="text/css" href="http://extjs-
public.googlecode.com/svn/tags/extjs-4.0.7/release/resources/css/ext-
all.css" />
    <!--End-->

    <script type="text/javascript">
        $(document).ready(function(){
            var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
            {
                dataProvider: [
                    { "id": "5001", "type":
"None" },
                    { "id": "5002", "type":
"Glazed" },
                    { "id": "5005", "type":
"Sugar" },
                    { "id": "5007", "type":
"Powdered Sugar" },
                    { "id": "5006", "type":
"Chocolate with Sprinkles" },
                    { "id": "5003", "type":
"Chocolate" },
                    { "id": "5004", "type":
"Maple" }
                ]
            });
        });
    </script>
</head>
<body>
    <div id="gridContainer" style="height: 400px;width: 100%;">

    </div>

```



```
</body>
</html>
```

## DOJO Sample

```
<!doctype html>
<html lang="en" ng-app="myApp">
<head>
  <meta charset="utf-8">
  <title>Simple</title>

  <!--These are ext and plugins that we use from ext-->
  <script type="text/javascript" src="http://ajax.googleapis.com/
ajax/libs/dojo/1.8.1/dojo/dojo.js" data-dojo-config="parseOnLoad:
true"></script>
  <!--End-->

  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
minified-compiled-dojo.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
examples/js/Configuration.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
themes.js"></script>
  <!--End-->
  <!--css imports-->
  <link rel="stylesheet" href="http://htmltreegrid.com/demo/
flexicious/css/flexicious.css" type="text/css"/>
  <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/
libs/dojo/1.8.1/dijit/themes/claro/document.css"/>
  <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/
libs/dojo/1.8.1/dijit/themes/claro/claro.css"/>
  <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/
libs/dojo/1.8.1/dojox/widget/Toaster/Toaster.css"/>
  <!--End-->

  <script type="text/javascript">
    require(["dojo/dom", "dojo/_base/fx", "dojo/domReady!"],
function (dom, fx) {

    //This is how you define the grid - you pass in a div
    element as a constructor, and a data provider.
    //This is one of the ways in which this can be done, you
    can specify configuration and themes as we will see in later examples
    var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
    {
      dataProvider: [
        { "id": "5001", "type":
"None" },
```

```

        { "id": "5002", "type":
"Glazed" },
        { "id": "5005", "type":
"Sugar" },
        { "id": "5007", "type":
"Powdered Sugar" },
        { "id": "5006", "type":
"Chocolate with Sprinkles" },
        { "id": "5003", "type":
"Chocolate" },
        { "id": "5004", "type":
"Maple" }
    ]
    });
});
</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px;width: 100%;"></div>

</body>
</html>

```

## Details of the Markup

If you copy paste any of the above examples, you should see a basic grid render in the browser like the screenshot below.

id	type	HTMLTreeGrid Trial Version!
5001	None	
5002	Glazed	
5005	Sugar	
5007	Powdered Sugar	
5006	Chocolate with Sprinkles	
5003	Chocolate	
5004	Maple	

Let us examine the html markup used to generate the grid. Since the comments in the

1) The HTML wrapper:

```
<!doctype html>
<html lang="en" ng-app="myApp">
<head>
  <meta charset="utf-8">
  <title>Simple</title>
```

2) The framework specific imports (These are from the jquery sample)

```
<!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript" src="http://
htmltreegrid.com/demo/external/js/adapters/jquery/jquery-
1.8.2.js"></script>
  <script type="text/javascript" src="http://
htmltreegrid.com/demo/external/js/adapters/jquery/jquery-
ui-1.9.1.custom.min.js"></script>
  <script type="text/javascript" src="http://
htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://
htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://
htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.ui.menu.js"></script>
  <script type="text/javascript" src="http://
htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.toaster.js"></script>
```

3) The HTMLTreeGrid imports - The first one is the actual framework specific minified version, the second one lets you specify where your

images are stored, and the third one is just a reference to the dozens of styling properties that the grid exposes.

```
<!--These are specific to htmltreegrid-->
<script type="text/javascript" src="http://
htmltreegrid.com/demo/minified-compiled-jquery.js"></
script>
<script type="text/javascript" src="http://
htmltreegrid.com/demo/examples/js/Configuration.js"></
script>
<script type="text/javascript" src="http://
htmltreegrid.com/demo/themes.js"></script>
```

3) The css imports, one for the framework (jQuery UI) and the other for the HTMLTreeGrid (flexicious.css).

```
<!--css imports-->
<link rel="stylesheet" href="http://htmltreegrid.com/
demo/flexicious/css/flexicious.css" type="text/css"/>
<link rel="stylesheet" href="http://htmltreegrid.com/
demo/external/css/adapters/jquery/jquery-ui-
1.9.1.custom.min.css" type="text/css"/>
```

4) The grid initialization code, including the data that the grid is supposed to show. This data is usually JSON, although we will go over some examples that cover other kinds of backend data:

```
<script type="text/javascript">
$(document).ready(function(){
    var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        dataProvider: [
            { "id": "5001", "type": "None" },
            { "id": "5002", "type": "Glazed" },
            { "id": "5005", "type": "Sugar" },
            { "id": "5007", "type": "Powdered Sugar" },
            { "id": "5006", "type": "Chocolate with
Sprinkles" },
            { "id": "5003", "type":
"Chocolate" },
            { "id": "5004", "type": "Maple" }
        ]
    });
});
```

5) The div that the grid will render inside:

```
<div id="gridContainer" style="height: 400px; width: 100%;">

</div>
```

## Common Configuration Options

---

The aim of this section is to introduce you to the most basic configuration settings. It is not meant to be an exhaustive guide on every single possible configuration option, just the most common ones.

### Basic Configuration Options

In the previous section, we saw how to create a simple grid using the framework of your choice. The one thing you may have asked, where did the columns come from? How do you specify additional columns? How to customize the cells, enable various options, etc.? In this section, we will cover some of the basis of the massive customization options. For a full list of available options, please refer: <http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html> (Please note, this link is still a work in progress, and will be refined over time).

In most line of business applications, you usually define columns, header text, data field, formatting, and other options for the grid. With HTMLTreeGrid, you use XML to do this. The reason being that since most of our customers develop applications across various platforms that use our products (Flex, iOS, Android and HTML), the same configuration XML can be reused across different product lines. Let us take the above example and add some configuration information to it. In these sections, we will start with the jQuery example, but the DOJO and EXT would follow quite similar approach.

The first thing we will do is to enable the toolbar and the plethora of features that the grid ships with. We will also add custom defined columns, and set some options on them. In the previous example, since we had not defined any configuration information, the grid performed introspection on the data provider that we gave it, and then auto generated the columns. To specify configuration, we will simply add the configuration parameter when we initialize the grid. Below is the configuration we will provide.

```
'<grid id="grid" enablePrint="true"
enablePreferencePersistence="true" enableExport="true"
forcePagerRow="true" pageSize="50" enableFilters="true"
enableFooters="true" >' +
'           <level>' +
'           <columns>' +
'           <column dataField="id" headerText="ID" />'
+
'           <column dataField="type"
headerText="Type"/>' +
'           </columns>' +
'           </level>' +
'           ' +
'       </grid>'
```

**Important:** The properties you see in the markup above, are defined on the `FlexDataGrid`, `FlexDataGridColumnLevel` and `FlexDataGridColumn` classes. These are documented here : <http://htmltreegrid.com/docs/>

[classes/flexiciousNmsp.FlexDataGrid.html](http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html)

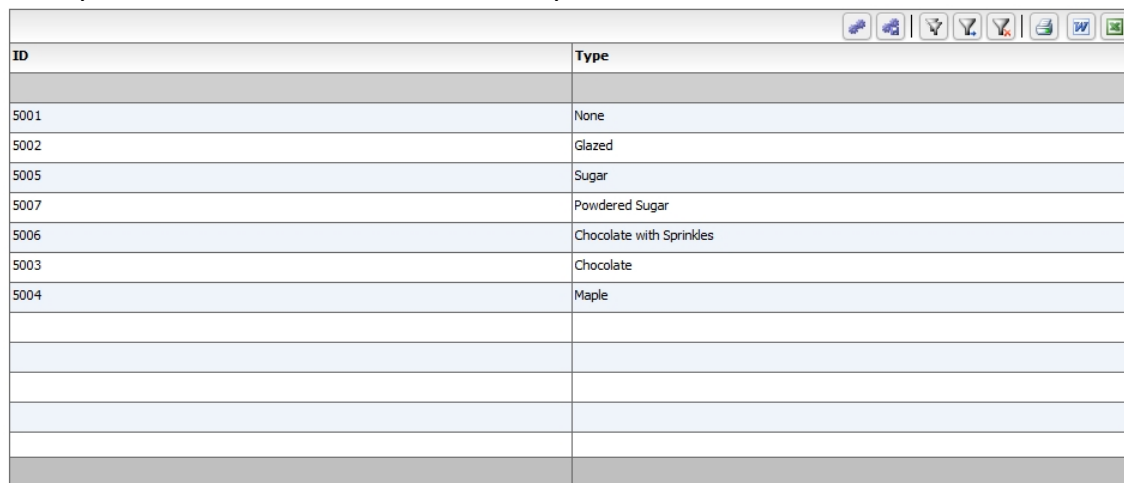
[http://htmltreegrid.com/docs/classes/](http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumn.html)

[flexiciousNmsp.FlexDataGridColumn.html](http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumnLevel.html)

[flexiciousNmsp.FlexDataGridColumnLevel.html](http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumnLevel.html)

They should map to either a field exposed via a property, or via getter/setter methods (e.g. `getDataField()` or `setDataField()`). The XML parsing mechanism of the grid automatically figures out if there is a setter by the name of the property you are trying to specify, and uses that first. If it does not find one, then it uses the property name. The initialization mechanism also tries to intelligently map strings to complex objects, like Functions, Class Factories, and Renderers. For details, refer to the Advanced Example - deep dive later in the guide.

Once you do this, below is the screenshot you should see:



ID	Type
5001	None
5002	Glazed
5005	Sugar
5007	Powdered Sugar
5006	Chocolate with Sprinkles
5003	Chocolate
5004	Maple

That's much better! The power of the product is coming into view! Below is the markup that you should have so far:

```
<!doctype html>
<html lang="en" ng-app="myApp">
<head>
  <meta charset="utf-8">
  <title>Simple</title>

  <!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery-1.8.2.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery-ui-1.9.1.custom.min.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
```

```

external/js/adapters/jquery/jquery.ui.menu.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.toaster.js"></script>
    <!--End-->

    <!--These are specific to htmltreegrid-->
    <script type="text/javascript" src="http://htmltreegrid.com/demo/
minified-compiled-jquery.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/
examples/js/Configuration.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/
themes.js"></script>
    <!--End-->
    <!--css imports-->
    <link rel="stylesheet" href="http://htmltreegrid.com/demo/
flexicious/css/flexicious.css" type="text/css"/>
    <link rel="stylesheet" href="http://htmltreegrid.com/demo/
external/css/adapters/jquery/jquery-ui-1.9.1.custom.min.css"
type="text/css"/>
    <!--End-->

    <script type="text/javascript">
        $(document).ready(function(){
            var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
            {
                configuration: '<grid id="grid"
enablePrint="true" enablePreferencePersistence="true"
enableExport="true" forcePagerRow="true" pageSize="50"
enableFilters="true" enableFooters="true" >' +
                    '
                        <level>' +
                        '
                            <columns>' +
                            '
                                <column
dataField="id" headerText="ID" />' +
                                '
                                    <column
dataField="type" headerText="Type"/>' +
                                    '
                                        </columns>' +
                                        '
                                            </level>' +
                                            '
                                                ' +
                                                ' </grid>',
                dataProvider: [
                    { "id": "5001", "type":
"None" },
                    { "id": "5002", "type":
"Glazed" },
                    { "id": "5005", "type":
"Sugar" },
                    { "id": "5007", "type":
"Powdered Sugar" },

```

```

        { "id": "5006", "type":
"Chocolate with Sprinkles" },
        { "id": "5003", "type":
"Chocolate" },
        { "id": "5004", "type":
"Maple" }
    ]
    });

});
</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px;width: 100%;">

        </div>
</body>
</html>

```

So what did we do here? We just gave the grid an initial configuration to render itself. Let's quickly examine the properties:

```

enablePrint="true" // This Enables the print operation in the toolbar
enablePreferencePersistence="true" // This Enables the preferences in
the toolbar
enableExport="true" // This Enables the export operation in the
toolbar

```

```

forcePagerRow="true"
Flag to force appearance of the pager row even with
enablePaging=false. Use this flag to show the pager control even if
the enablePaging is set to false. This is used in a scenario where
you wish to show buttons other than the paging buttons in the pager
bar.

```

```

enableFilters="true" // This Enables the inline filters
enableFooters="true" // This Enables the footers.

```

And on the column:

```

dataField="id" //The property on the data provider object that you
want to show in this column. This can be a nested property - like
address.line1 or address.country.name.
headerText="ID" //The header text to show on the column.

```

In the above example, an important property is the dataField. Each of these properties has a description in the API documentation. Please NOTE – since some properties have getters and



setters, the documentation will most likely be associated with the corresponding getter OR setter, depending on which is most likely to be used.. For example:

```
dataField="id"
```

The documentation in this case is associated with dataField appears under getDataField: [http://www.htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumn.html#method\\_getDataField](http://www.htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumn.html#method_getDataField)  
getDataField ()

The name of the field or property in the data provider item associated with the column. Each DataGridColumn control requires this property and/or the labelFunction property to be set in order to calculate the displayable text for the item renderer. If the dataField and labelFunction properties are set, the data is displayed using the labelFunction and sorted using the dataField. If the property named in the dataField does not exist, the sortCompareFunction must be set for the sort to work correctly.

This value of this property is not necessarily the String that is displayed in the column header. This property is used only to access the data in the data provider. For more information, see the headerText property.

If you specify a complex property, the grid takes over the sortCompareFunction, and the sortField property is ignored.

For the most part, each of the properties should be found on the FlexDataGrid, FlexDataGridColumn, FlexDataGridColumnLevel, or FlexDataGridColumnGroup classes in the documentation:

<http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html>

**One thing to keep in mind - in the API docs, these might appear in getter/setter format. For example, enablePaging appears as getEnablePaging here: [http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html#method\\_getEnablePaging](http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html#method_getEnablePaging)**



- MultiSelectComboBox
- DateComboBox
- DateRangeBox
- NumericRangeBox
- NumericTextInput

You can write your own custom text input controls by extending any of these, or by implementing the IFilterControl, ICustomMatchFilterControl, or IDynamicFilterControl. For an example of this, please look at <http://blog.flexicious.com/post/ICustomMatchFilterControl-example.aspx>

The filterOperation also, can be one of a number of different options. These options are defined in the FilterExpression class, and are mentioned below for quick reference:

```

        public static var FILTER_OPERATION_TYPE_NONE:String =
"None";
        public static var FILTER_OPERATION_TYPE_EQUALS:String =
"Equals";
        public static var FILTER_OPERATION_TYPE_NOT_EQUALS:String
= "NotEquals";
        public static var FILTER_OPERATION_TYPE_BEGINS_WITH:String
= "BeginsWith";
        public static var FILTER_OPERATION_TYPE_ENDS_WITH:String
= "EndsWith";
        public static var
FILTER_OPERATION_TYPE_CONTAINS:String="Contains";
        public static var
FILTER_OPERATION_TYPE_DOES_NOT_CONTAIN:String = "DoesNotContain";
        public static var
FILTER_OPERATION_TYPE_GREATER_THAN:String = "GreaterThan";
        public static var FILTER_OPERATION_TYPE_LESS_THAN:String
= "LessThan";
        public static var
FILTER_OPERATION_TYPE_GREATERTHANEQUALS:String = "GreaterThanEquals";
        public static var
FILTER_OPERATION_TYPE_LESS_THAN_EQUALS:String = "LessThanEquals";
        public static var FILTER_OPERATION_TYPE_IN_LIST:String =
"InList";
        public static var FILTER_OPERATION_TYPE_NOT_IN_LIST:String
= "NotInList";
        public static var FILTER_OPERATION_TYPE_BETWEEN:String =
"Between";
        public static var FILTER_OPERATION_TYPE_IS_NOT_NULL:String
= "IsNotNull";
        public static var FILTER_OPERATION_TYPE_IS_NULL:String =
"IsNull";

```

This is documented at:

[http://htmltreegrid.com/docs/classes/flexiciousNmosp.FlexDataGridColumn.html#property\\_filterOperation](http://htmltreegrid.com/docs/classes/flexiciousNmosp.FlexDataGridColumn.html#property_filterOperation)

One of the things to keep in mind here, are that the filter mechanism itself is quite a powerful beast. Not only can you specify a number of parameters associated with the built in filters, but you can also create your own filters, with your own logic.

There are many more options with filters, as you can see in our demo console. This is a getting started guide, so we are only going to cover the basics here. You can refer to configuration files from the demo console as well as our API docs for additional options with filters.

## Configuration Options - Footers

In the previous section, we saw how to enable a basic set of filters. In this section, we will look into another nice feature, the footers.

Footers by default are drawn below the grid. You can customize this, by using the `displayOrder` property. In this section, we will add footer to the grid we built. All you have to do, is to specify a `footerOperation`. First, let's look at the markup required to enable simple `footerOperation` for the grid:

All you have to do is to add the text in red to the markup from above:

```
configuration
```

```
'
        <columns>' +
        <column dataField="id" headerText="ID"
filterControl="TextInput" filterOperation="Contains"
footerOperation="sum"/>footerOperation="count"/>
```

Once you do this, you should see the following UI:

ID	Type
5001	None
5002	Glazed
5005	Sugar
5007	Powdered Sugar
5006	Chocolate with Sprinkles
5003	Chocolate
5004	Maple
35028.00	7.00

Notice the footers are automatically created and shown for you. The footer operation by default can be one of a number of built in operations. From the documentation at:

<http://htmltreegrid.com/docs/classes/>

[flexiciousNmsp.FlexDataGridColumn.html#property\\_footerOperation](#)

The operation to apply to the footer. One of the following values:

- average
- sum
- min
- max
- count

You can specify custom footer labels as well. For example, modifying the markup a little bit gives us this:

ID	Type
5001	None
5002	Glazed
5005	Sugar
5007	Powdered Sugar
5006	Chocolate with Sprinkles
5003	Chocolate
5004	Maple
Sum: 35028.00	Sum: 7.000

And here is the markup we used:

```
'
    <columns>' +
    '
        <column dataField="id" headerText="ID"
filterControl="TextInput" filterOperation="Contains"
footerLabel="Sum: " footerOperation="sum"
footerOperationPrecision="2"/>' +
    '
        <column dataField="type" headerText="Type"
filterControl="TextInput" filterOperation="Contains"
footerLabel="Sum: " footerOperation="count"
footerOperationPrecision="3"/>' +
    '
    </columns>' +
```

Finally, it is possible to provide your own footerLabelFunction as well.

First, just add the custom label function as below:

```
var myCompanyNameSpace = {};

myCompanyNameSpace.customFooterFunction = function () {
    var html = "<div> Look Mommy, custom footer!</div>";
    return html;
};
```

Then, reference it in the markup as below:

```
'
    <column dataField="type" headerText="Type"
filterControl="TextInput" filterOperation="Contains"
footerLabelFunction="myCompanyNameSpace.customFooterFunction"/>' +
```

You should see:

ID	Type
5001	None
5002	Glazed
5005	Sugar
5007	Powdered Sugar
5006	Chocolate with Sprinkles
5003	Chocolate
5004	Maple
Sum: 35028.00	Look Mommy, custom footer!

Here, we covered an important concept - passing functions into configurations. The same concept is used to add event listeners, provide custom renderers, etc, as we will see in future examples. We basically expose a namespaced function, and pass it into the configuration so it can be navigated to. The grid then keeps a pointer to it and uses it for generating custom HTML for the footer. Same concept is used in labelFunction with the regular cells, as we will see in future examples.

For those of you who want the source for the complete running example above, here it is, in all its glory!

```
<!doctype html>
<html lang="en" ng-app="myApp">
<head>
  <meta charset="utf-8">
  <title>Simple</title>

  <!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery-1.8.2.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery-ui-1.9.1.custom.min.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.ui.menu.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
external/js/adapters/jquery/jquery.toaster.js"></script>
  <!--End-->

  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
minified-compiled-jquery.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
```

```

examples/js/Configuration.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/
themes.js"></script>
    <!--End-->
    <!--css imports-->
    <link rel="stylesheet" href="http://htmltreegrid.com/demo/
flexicious/css/flexicious.css" type="text/css"/>
    <link rel="stylesheet" href="http://htmltreegrid.com/demo/
external/css/adaptor/jquery/jquery-ui-1.9.1.custom.min.css"
type="text/css"/>
    <!--End-->

    <script type="text/javascript">
        var myCompanyNameSpace = {};

        myCompanyNameSpace.customFooterFunction = function () {
            var html = "<div> Look Mommy, custom footer!</div>";
            return html;
        };

        $(document).ready(function(){
            var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
            {
                configuration: '<grid id="grid"
enablePrint="true" enablePreferencePersistence="true"
enableExport="true" forcePagerRow="true" pageSize="50"
enableFilters="true" enableFooters="true" >' +
                    '
                        <level>' +
                    '
                        <columns>' +
                    '
                        <column
dataField="id" headerText="ID" filterControl="TextInput"
filterOperation="Contains" footerLabel="Sum: " footerOperation="sum"
footerOperationPrecision="2"/>' +
                    '
                        <column
dataField="type" headerText="Type" filterControl="TextInput"
filterOperation="Contains"
footerLabelFunction="myCompanyNameSpace.customFooterFunction"/>' +
                    '
                        </columns>' +
                    '
                        </level>' +
                    '
                    ' +
                    '</grid>',
                dataProvider: [
                    { "id": "5001", "type":
"None" },
                    { "id": "5002", "type":
"Glazed" },
                    { "id": "5005", "type":
"Sugar" },

```



```

        { "id": "5007", "type":
"Powdered Sugar" },
        { "id": "5006", "type":
"Chocolate with Sprinkles" },
        { "id": "5003", "type":
"Chocolate" },
        { "id": "5004", "type":
"Maple" }
    ]
    });

    });
</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px;width: 100%;">

        </div>
</body>
</html>

```









## Configuration Options - Paging

In the previous sections, we saw how to enable a basic set of filters and footers. In this section, we will look into another nifty little gem, paging.

In most LOB applications, you almost always have a ton of data to show. One of the very frequent mechanisms to show a manageable chunk of data is the paging mechanism. At the very basic level, enabling paging is dead simple: just set `enablePaging=true`.

```
'<grid id="grid" enablePrint="true"
enablePreferencePersistence="true" enableExport="true"
forcePagerRow="true" enablePaging="true" pageSize="2"
enableFilters="true" enableFooters="true" >' +
```

And voila:

Items 1 to 2 of 7. Page 1 of 4		Go to Page: 1		       			
ID	Type						
5001	None						
5002	Glazed						
</							

Now, what you see here is called client paging. In that, all records are loaded in memory in the grid, and the grid is simply showing you the first page. You don't have to do anything! Just let the grid handle paging for you. This works well, if the number of records is manageable, but can get difficult if you have hundreds of thousands of records, since you never want to have that many records in memory. So, we provide the concept of `filterPageSortMode`. Although too complex a topic to cover in this getting started guide, it is something you should know - it's a potent tool in your toolbox, something you may want to whip out when facing a situation with a large number of records.

From the docs at [http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html#method\\_getFilterPageSortMode](http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html#method_getFilterPageSortMode)  
`getFilterPageSortMode()`

The Filter/Page/Sort Mode. Can be either "server" or "client". In client mode, the grid will take care of paging, sorting and filtering once the dataprovider is set. In server mode, the grid will fire a `com.flexicious.grids.events.FilterPageSortChangeEvent` named `filterPageSortChange` that should be used to construct an appropriate query to be sent to the backend.

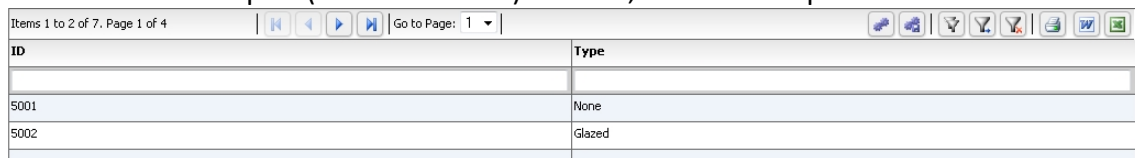
## Configuration Options - Print & Export

In the previous sections, we saw how to enable a basic set of filters and footers, as well as add paging. In this section, we will look into another powerful feature of the product, print and export.

As with the earlier features, enabling Print and Export is really simple. Just set enableExport and enablePrint to true.

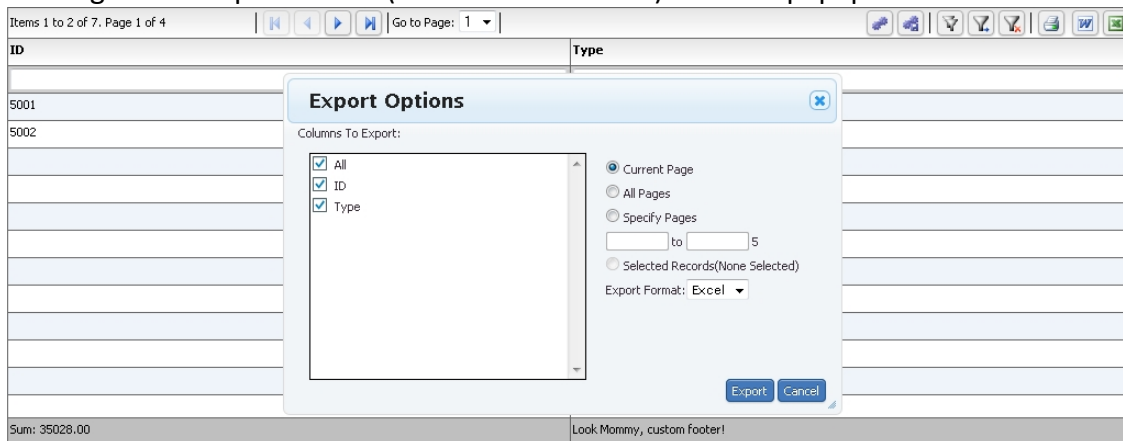
```
configuration: '<grid id="grid" enablePrint="true"
enablePreferencePersistence="true" enableExport="true"
forcePagerRow="true" enablePaging="true" pageSize="2"
enableFilters="true" enableFooters="true" >'
```

This will enable export (word and excel) buttons, as well as a print button.



ID	Type
5001	None
5002	Glazed

Clicking on the Export button (either word or excel) shows a popup like this:



Items 1 to 2 of 7. Page 1 of 4

Go to Page: 1

**Export Options**

Columns To Export:

- ☒ All
- ☒ ID
- ☒ Type

☒ Current Page  
☐ All Pages  
☐ Specify Pages  
 to 5  
☐ Selected Records(None Selected)

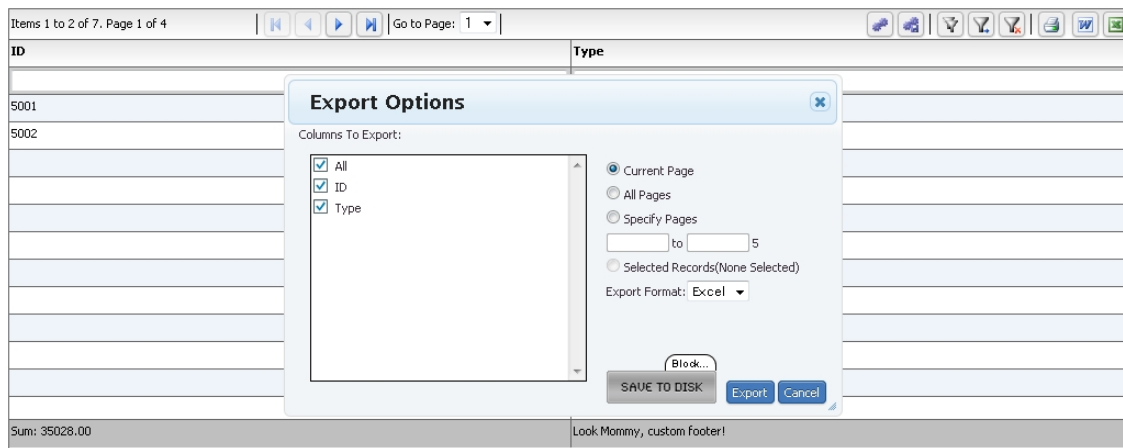
Export Format: Excel

Export Cancel

Sum: 35028.00 Look Mommy, custom footer!

The user can then choose to export either the current page, all pages or specify pages they want to export. If they had something selected, they can choose to specifically export those selected records as well.

One the export button is pressed; we use the excellent Downloadify library to download the file without any server echo.



When you click on the save to disk button, a file is saved on to your system that contains the exported data. (Please note, this only works on a webserver, it wont work if you run locally).

Please note: The downloadify library internally uses the Flash Player to save the file to disk. In a browser, there is no fully supported cross browser compliant mechanism that will allow client side generation of the file. Also, if you want to customize the download image, you may need to provide your own download.png image in \flexicious\css\images. If you want to use a custom UI for the download, and not an image, you can refer to this issue on the downloadify forums: <https://github.com/dcneiner/Downloadify/issues/11>. If you are deploying to a plugin free environment like mobile devices, then downloadify will not work. You may need to use the server echo mechanism.

The way you do this is to set enableLocalFilePersistence flag on exportOptions to false. This makes the grid use the server echo mechanism.

```
$(document).ready(function(){
    var grid = new
flexiciousNmisp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        configuration: . . . ,
        dataProvider:
    });
    grid.excelOptions.enableLocalFilePersistence = false;
});
```

There is a full discussion about server echo here:

<http://blog.flexicious.com/post/Excel-Word-and-Html-Export-and-the-Echo-URL.aspx>

Similar to the Export, Print also works out of the box with the same set of options, i.e. user can then choose to print either the current page, all pages or specify pages they want to print. If they had something selected, they can choose to specifically print those selected records as well.

As with all our features, both Print and Export are quite customizable. The PrintExportOptions

and ExportOptions class provide the means to do so. The Print is nothing but an HTML based Export.

Here are lists of properties from the PrintExportOptions class that give you the means to customize this behavior:

`excludeHiddenColumns`

A flag that will To disable the user from printing or exporting invisible columns altogether . User cannot go into column picker and include these columns. To hide any columns that are not visible from being printed or exported by default, please use the `hideHiddenColumns` instead

**Default:** false

`hideHiddenColumns`

A flag that will hide any columns that are not visible from being printed or exported by default. User can still go into column picker and include these columns. To disable the user from printing or exporting invisible columns altogether, please use the `excludeHiddenColumns` instead. Please note, when you set this flag, the `columnsToPrint` will get overwritten, so any changes you make to that array will be ignored.

**Default:** false

`pageFrom`

In conjunction with `printOption/exportOption = PRINT_EXORT_SPECIFIED_PAGES` determines which pages of a grid with paging enabled to print.

`pageTo`

In conjunction with `printOption/exportOption = PRINT_EXORT_SPECIFIED_PAGES` determines which pages of a grid with paging enabled to print.

`printExportOption`

Specifies what to export, valid options are: `EXPORT_CURRENT_PAGE`  
`EXPORT_ALL_PAGES` `EXPORT_SPECIFIED_PAGES`  
`EXPORT_SELECTED_RECORDS`

The All pages and specified pages are only available if paging is enabled and the grid is in client mode.

`showColumnPicker`

Flag to control whether or not to show the column picker. Defaults to true.

Additional parameters can be found here:

<http://htmltreegrid.com/docs/classes/flexiciousNmsp.ExportOptions.html>

You can also specify custom exporters, and we will cover this in some of our advanced example deep dives.

## Configuration Options - Preferences/Settings

In the previous sections, we saw how to enable a basic set of filters, footers, paging, and addition of Print/Export functionality. In this section, we will look into another highly visible feature of the product, user settings and preferences.

Enabling this behavior is quite straightforward.

```
'<grid id="grid" enablePrint="true"
enablePreferencePersistence="true" enableExport="true"
forcePagerRow="true" enablePaging="true" pageSize="2"
```

```
enableFilters="true" enableFooters="true" >' +
```

This will enable the settings and the save settings button buttons like below:

Items 1 to 2 of 7. Page 1 of 4

Go to Page: 1

ID	Type
5001	None
5002	Glazed

Clicking on the Settings button shows a popup like this:

Items 1 to 2 of 7, Page 1 of 4

Go to Page: 1

ID	Type
5001	
5002	

Settings

Columns To Show

☒ All  
☒ ID  
☒ Type

☒ Show Footers  
☒ Show Filter  
Records Per Page 2

Apply

Cancel

Clicking on the Save Settings Button, shows a popup like this:

[illegible]

As should be obvious from the above screen, the settings popup allows you to change the visibility of your columns, visibility of footers and filters, and the page size. The grid headers are sortable, resizable, and re-arrangeable via drag and drop. All of these settings, once changed are usually lost when the user closes the application that uses the product. With the preference persistence mechanism, all these settings can be persisted and loaded in the future when the grid is loaded.

There are a few properties that you should keep in mind for preference persistence: `setPreferencePersistenceKey` (or the `preferencePersistenceKey` property in XML)

- String value that uniquely identifies this grid across the application. If you have multiple grids' with `enablePreferencePersistence`, and they happen to share the same value for the `id` field, e.g. `id="grid1"` they might overwrite each others' preferences. To combat this situation, we provide a property, which defaults to the `id` of the grid, but you can override to provide a globally unique key.

The other nice addition in regards to this feature is the ability to save preferences on the server.

The grid has a robust persistence preference mechanism that "just works" out of the box, but the preferences are stored on the client machine in the form of LocalStorage by default. This enables the preference persistence mechanism to work without any additional coding on the part of the developers utilizing the library. While this may be sufficient for most people, this will not work in scenarios when the same user uses multiple machines to access your application, or if multiple users access the application using the same machine. This is where the `preferencePersistenceMode` comes into play:

`preferencePersistenceMode`  
String

String value "server" or "client". When this property is set to client(default), the grid uses local storage on the client to store preference settings. When it is set to server, the grid fires an event, `preferencesChanged`, which contains a string representation of the preference values. This can then be persisted on the backend, tied to a specific user id.

**Default:** client

This post discusses the backend implementation:

<http://blog.flexicious.com/post/Persisting-Preferences-on-the-Server.aspx>

## Configuration Options – Column Formatting

So far, we have seen how to enable filters, footers, paging, print, preferences and more. In this section let's look at a very common requirement, one that of formatting column values. Depending on your situation, there are numerous ways to customize the content of the cells. We provide you with a rich API that allows you to pretty much customize every single grid cell.

There are a number of options that the grid exposes to let you control the content of each cell:

### 1) The Header cells:

- a. The default text of the header cell maps to the `headerText` of the column configuration. If nothing is specified, it uses the `dataField` of the column.
- b. If you want to customize the content of the header cell, you can use a custom `headerRenderer`.

### 2) The Footer Cells:

- a. The default text of the footer is calculated on basis of the following logic:
  - i. First we check to see if a `footerLabelFunction2` is specified on the column. If so, we use it.
  - ii. Second, we check to see if there is a `footerLabelFunction` specified on the column. If so, we use it.
  - iii. Finally, if neither of these is specified, we use a default footer label function, which performs the following logic:
    1. Checks to see the value of the columns `footerOperation`. The valid values for this property are sum, min, max, count and average. On basis of this, it computes the value.
    2. It calls `toFixed` method passing in the `footerOperationPrecision` value to give it the appropriate number of decimal places
    3. If there is a `footerFormatter` specified, it calls the `format` method of the `footerFormatter` passing in the value and displays the result.
    4. If there is a `footerLabel` specified, it will concatenate the value of

- the footer label to the result of the calculated value in step 3.
- b. If the above customization logic is not enough, there is always a mechanism to specify a custom footer renderer.
- 3) The Data Cells: For the most part, this is where most of the requirements come in for customization. Based upon your needs, there are several different methods that the grid exposes for you to achieve customization of your data cells
- a. If there is a linktext property specified on the column, this function returns that.
  - b. If the DataGridColumn or its DataGrid control has a non-null `<code>labelFunction</code>` property, it applies the function to the data object.
  - c. If the DataGridColumn or its DataGrid control has a non-null `<code>labelFunction2</code>` property, it applies the function to the data object.
  - d. Otherwise, the method extracts the contents of the field specified by the `<code>dataField</code>` property, or gets the string value of the data object.
  - e. If the method cannot convert the parameter to a String, it returns a single space
  - f. Finally, if none of the above works for you, you can use what we call a custom item renderer. This is especially useful when there is custom functionality that you want to associate with each cell in the grid.

Now that all of this is said, let's look at what the typical markup for each of the above options looks like:

```
configuration: '<grid id="grid" enablePrint="true"
enablePreferencePersistence="true" enableExport="true" forcePagerRow="true"
pageSize="50" enableFilters="true" enableFooters="true" >' +
    '
    <level>' +
    '
    <columns>' +
    '
        <column dataField="id"
headerText="ID" filterControl="TextInput" filterOperation="Contains"
footerLabel="Sum: " footerOperation="sum" footerOperationPrecision="2"/>' +
    '
        <column dataField="type"
headerText="Type" filterControl="TextInput" filterOperation="Contains"
footerLabelFunction="myCompanyNameSpace.customFooterFunction"/>' +
    '
        <column dataField=""
headerText="Label Function Example" filterControl="TextInput"
filterOperation="Contains"
labelFunction="myCompanyNameSpace.labelFunctionExample"/>' +
    '
        <column dataField=""
headerText="Label Function 2 Example" filterControl="TextInput"
filterOperation="Contains"
labelFunction2="myCompanyNameSpace.labelFunction2Example"/>' +
    '
        <column dataField="active"
headerText="Item Renderer Example"
headerRenderer="myCompanyNameSpace.CheckBoxHeaderRenderer"
filterControl="TextInput" filterOperation="Contains"
itemRenderer="myCompanyNameSpace.CheckBoxRenderer"/>' +
    '
    </columns>' +
    '
    </level>' +
    '
    ' +
    '</grid>',
```

You will notice that we have highlighted a few important items here:

- 1) For `labelFunction="myCompanyNameSpace.labelFunctionExample"`, let's look at the



actual label function. In this method, we get the item being rendered, and the column being rendered.

```
myCompanyNameSpace.labelFunctionExample = function (item,
column) {
    var html = "<b> Column:" + column.getHeaderText() + " : </b>
Item: " + item.type ;
    return html;
};
```

- 2) For labelFunction2="myCompanyNameSpace.labelFunction2Example", let's look at the actual label function – the key difference here is that we also get the cell being rendered. Each cell has a rowInfo object. The rowInfo object associated with this cell. This cell should be within the cells collection of this rowInfo object. Each cell has a corresponding rowinfo object. The rowInfo, in turn has a rowPositionInfo object. The key difference between these two objects is that there are only as many rowInfo objects as there visible rows on the page. RowPositionInfo objects on the other hand, there are as many of these as there are total items in the data provider.

```
myCompanyNameSpace.labelFunction2Example = function (item,
column, cell) {
    var html = "<b> Column:" + column.getHeaderText() + " : </b>
Item: " + item.type + ", cell: " +
cell.rowInfo.rowPositionInfo.getRowIndex();
    return html;
};
```

- 3) Finally, if specifying custom html is not enough for your needs, you can use itemRenderers, which is a very powerful concept. Item Renderers are basically custom JavaScript classes, that wrap regular dom elements and sit inside the various cells of the grid. Every UI Element in the HTMLTreeGrid extends from flexiciousNmsp.UIComponent class. This is nothing but a thin wrapper that encapsulates a domElement, and provides various utility methods that allow for things like sizing, positioning, validation and invalidation, event management and display list hierarchy. In fact, even the HTMLTreeGrid class (FlexDataGrid) class eventually inherits from UIComponent. When you define your own item renderer, you have to inherit it from the flexiciousNmsp.UIComponent class. We use prototype based inheritance. The key about item renderers is that they should expose a method called setData. This method is called when the grid instantiates an itemRenderer and prepares it for display. Finally, another important thing to keep in mind is that each item renderer gets a parent property. This points to the parent FlexDataGridCell object. The actual type is FlexDataGridDataCell for itemRenderers, FlexDataGridHeaderCell for headerRenderers, FlexDataGridFilterCell for filterRenderers, and FlexDataGridFooterCell for footerRenderers. Let's take a look at what a simple item renderer looks like:

```
/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function(window)
{
    "use strict";
    var CheckBoxRenderer, uiUtil = flexiciousNmsp.UIUtils,
```

```

flxConstants = flexiciousNmsp.Constants;
/**
 * A CheckBoxRenderer is a custom item renderer, that defines
how to use custom cells with logic that you can control
 * @constructor
 * @namespace flexiciousNmsp
 * @extends UIComponent
 */
CheckBoxRenderer=function(){
    //make sure to call constructor
    flexiciousNmsp.UIComponent.apply(this,["input"]); //second
parameter is the tag name for the dom element.
    this.domElement.type = "checkbox"; //so our input element
becomes a checkbox;

    /**
     * This is a getter/setter for the data property. When the
cell is created, it belongs to a row
     * The data property points to the item in the grids
dataprotider that is being rendered by this cell.
     * @type {*}
     */
    this.data=null;

    //the add evt listener will basically proxy all DomEvents
to your code to handle.
    this.addEventListener(this,flxConstants.EVENT_CHANGE,this.
onChange);
};
myCompanyNameSpace.CheckBoxRenderer = CheckBoxRenderer; //add
to name space
CheckBoxRenderer.prototype = new flexiciousNmsp.UIComponent();
//setup hierarchy
CheckBoxRenderer.prototype.typeName =
CheckBoxRenderer.typeName = 'CheckBoxRenderer'; //for quick
inspection
CheckBoxRenderer.prototype.getClassName=function(){
    return ["CheckBoxRenderer","UIComponent"]; //this is a
mechanism to replicate the "is" and "as" keywords of most other OO
programming languages
};

/**
 * This is important, because the grid looks for a "setData"
method on the renderer.
 * In here, we intercept the call to setData, and inject our
logic to populate the text input.
 * @param val
 */
CheckBoxRenderer.prototype.setData=function(val){
    flexiciousNmsp.UIComponent.prototype.setData.apply(this,

```

```

[val]);
    var cell = this.parent; //this is an instance of
FlexDataGridDataCell (For data rows)
    var column = cell.getColumn();//this is an instance of
FlexDataGridColumn.
    this.domElement.checked=this.data[column.getDataField()];
};
/**
 * This event is dispatched when the user clicks on the icon.
The event is actually a flexicious event, and has a trigger event
 * property that points back to the original domEvent.
 * @param event
 */
CheckBoxRenderer.prototype.onChange=function(evt){

    //in the renderer, you have the handle to the cell that
the renderer belongs to, via the this.parent property that you
inherit from flexiciousNmsp.UIComponent.

    var cell = this.parent; //this is an instance of
FlexDataGridDataCell (For data rows)
    var column = cell.getColumn();//this is an instance of
FlexDataGridColumn.

    this.data[column.getDataField()]
=this.domElement.checked;//we use the dom element to wire back the
value to the data object.
};
//This sets the inner html, and grid will try to set it.
Since we are an input field, IE 8 will complain. So we ignore it
since we dont need it anyway.
CheckBoxRenderer.prototype.setText=function(val){

};
}(window));

```

Similar to the above, the same concept is extended to header renderers (and footer, filter, pager as well as nextLevelRenderers). So, now that all of that is done, let's take a quick look at the final markup for this page, and generated output:

ID	Type	Label Function Example	Label Function 2 Example	
5001	None	Column:Label Function Example : Item: None	Column:Label Function 2 Example : Item: None, cell: 0	<input checked="" type="checkbox"/>
5002	Glazed	Column:Label Function Example : Item: Glazed	Column:Label Function 2 Example : Item: Glazed, cell: 1	<input checked="" type="checkbox"/>
5005	Sugar	Column:Label Function Example : Item: Sugar	Column:Label Function 2 Example : Item: Sugar, cell: 2	<input checked="" type="checkbox"/>
5007	Powdered Sugar	Column:Label Function Example : Item: Powdered Sugar	Column:Label Function 2 Example : Item: Powdered Sugar, cell: 3	<input type="checkbox"/>
5006	Chocolate with Sprinkles	Column:Label Function Example : Item: Chocolate with Sprinkles	Column:Label Function 2 Example : Item: Chocolate with Sprinkles	<input checked="" type="checkbox"/>
5003	Chocolate	Column:Label Function Example : Item: Chocolate	Column:Label Function 2 Example : Item: Chocolate, cell: 5	<input type="checkbox"/>
5004	Maple	Column:Label Function Example : Item: Maple	Column:Label Function 2 Example : Item: Maple, cell: 6	<input checked="" type="checkbox"/>
Sum: 35	Look Mommy, custom fc			

```

<!doctype html>
<html lang="en" ng-app="myApp">
<head>
  <meta charset="utf-8">
  <title>Simple</title>

  <!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-1.8.2.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-ui-1.9.1.custom.min.js"></
script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.ui.menu.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.toaster.js"></script>
  <!--End-->

  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/minified-compiled-jquery.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/examples/js/Configuration.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/themes.js"></script>
  <!--End-->
  <!--css imports-->
  <link rel="stylesheet" href="http://www.htmltreegrid.com/demo/

```

```

flexicious/css/flexicious.css" type="text/css"/>
<link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
external/css/adapters/jquery/jquery-ui-1.9.1.custom.min.css"
type="text/css"/>
<!--End-->

<script type="text/javascript">
    var myCompanyNameSpace = {};

    myCompanyNameSpace.labelFunctionExample = function (item,
column) {
        var html = "<b> Column:" + column.getHeaderText() + " : </b>
Item: " + item.type ;
        return html;
    };
    myCompanyNameSpace.labelFunction2Example = function (item,
column, cell) {
        var html = "<b> Column:" + column.getHeaderText() + " : </b>
Item: " + item.type + ", cell: " +
cell.rowInfo.rowPositionInfo.getRowIndex();
        return html;
    };
    myCompanyNameSpace.customFooterFunction = function () {
        var html = "<div> Look Mommy, custom footer!</div>";
        return html;
    };

    /**
     * Flexicious
     * Copyright 2011, Flexicious LLC
     */
    (function(window)
    {
        "use strict";
        var CheckBoxRenderer, uiUtil = flexiciousNmsp.UIUtils,
        flxConstants = flexiciousNmsp.Constants;
        /**
         * A CheckBoxRenderer is a custom item renderer, that defines how
         to use custom cells with logic that you can control
         * @constructor
         * @namespace flexiciousNmsp
         * @extends UIComponent
         */
        CheckBoxRenderer=function(){
            //make sure to call constructor
            flexiciousNmsp.UIComponent.apply(this,["input"]); //second
            parameter is the tag name for the dom element.
            this.domElement.type = "checkbox"; //so our input element
            becomes a checkbox;

```

```

    /**
     * This is a getter/setter for the data property. When the
    cell is created, it belongs to a row
     * The data property points to the item in the grids
    dataprovider that is being rendered by this cell.
     * @type {*}
     */
    this.data=null;

    //the add evt listener will basically proxy all DomEvents to
    your code to handle.
    this.addEventListener(this,flxConstants.EVENT_CHANGE,this.onC
    hange);
    };
    myCompanyNameSpace.CheckBoxRenderer = CheckBoxRenderer; //add to
    name space
    CheckBoxRenderer.prototype = new flexiciousNmsp.UIComponent(); //
    setup hierarchy
    CheckBoxRenderer.prototype.typeName = CheckBoxRenderer.typeName
    = 'CheckBoxRenderer';//for quick inspection
    CheckBoxRenderer.prototype.getClassName=function(){
        return ["CheckBoxRenderer","UIComponent"]; //this is a
    mechanism to replicate the "is" and "as" keywords of most other OO
    programming languages
    };

    /**
     * This is important, because the grid looks for a "setData"
    method on the renderer.
     * In here, we intercept the call to setData, and inject our
    logic to populate the text input.
     * @param val
     */
    CheckBoxRenderer.prototype.setData=function(val){
        flexiciousNmsp.UIComponent.prototype.setData.apply(this,
    [val]);
        var cell = this.parent; //this is an instance of
    FlexDataGridDataCell (For data rows)
        var column = cell.getColumn();//this is an instance of
    FlexDataGridColumn.
        this.domElement.checked=this.data[column.getDataField()];
    };
    /**
     * This event is dispatched when the user clicks on the icon. The
    event is actually a flexicious event, and has a trigger event
     * property that points back to the original domEvent.
     * @param event
     */
    CheckBoxRenderer.prototype.onChange=function(evt){

        //in the renderer, you have the handle to the cell that the

```

renderer belongs to, via the `this.parent` property that you inherit from `flexiciousNmsp.UIComponent`.

```

        var cell = this.parent; //this is an instance of
FlexDataGridDataCell (For data rows)
        var column = cell.getColumn();//this is an instance of
FlexDataGridColumn.

        this.data[column.getDataField()]=this.domElement.checked;//we
use the dom element to wire back the value to the data object.
    };
    //This sets the inner html, and grid will try to set it. Since
we are an input field, IE 8 will complain. So we ignore it since we
dont need it anyway.
    CheckBoxRenderer.prototype.setText=function(val){

    };
}(window));

/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function(window)
{
    "use strict";
    var CheckBoxHeaderRenderer, uiUtil = flexiciousNmsp.UIUtils,
    flxConstants = flexiciousNmsp.Constants;
    /**
     * A CheckBoxHeaderRenderer is a custom item renderer, that you
can use in a header cell. In this case, we customize the header
     * so that instead of showing a header label, we show a checkbox
that switches the dataField flag on all items.
     * @constructor
     * @namespace flexiciousNmsp
     * @extends UIComponent
     */
    CheckBoxHeaderRenderer=function(){
        //make sure to call constructor
        flexiciousNmsp.UIComponent.apply(this,["input"]);//second
parameter is the tag name for the dom element.
        this.domElement.type = "checkbox"; //so our input element
becomes a checkbox;
        this.domElement.checked=true;
        //the add event listener will basically proxy all DomEvents
to your code to handle.
        this.addEventListener(this,flxConstants.EVENT_CHANGE,this.onC
hange);
    };
    myCompanyNameSpace.CheckBoxHeaderRenderer =

```

```

CheckBoxHeaderRenderer; //add to name space
    CheckBoxHeaderRenderer.prototype = new
flexiciousNmsp.UIComponent(); //setup hierarchy
    CheckBoxHeaderRenderer.prototype.typeName =
CheckBoxHeaderRenderer.typeName = 'CheckBoxHeaderRenderer';//for
quick inspection
    CheckBoxHeaderRenderer.prototype.getClassName=function(){
        return ["CheckBoxHeaderRenderer","UIComponent"]; //this is a
mechanism to replicate the "is" and "as" keywords of most other OO
programming languages
    };

    /**
    * This event is dispatched when the user clicks on the icon. The
event is actually a flexicious event, and has a trigger event
    * property that points back to the original domEvent.
    * @param event
    */
    CheckBoxHeaderRenderer.prototype.onChange=function(event){

        //in the renderer, you have the handle to the cell that the
renderer belongs to, via the this.parent property that you inherit
from flexiciousNmsp.UIComponent.

        var cell = this.parent; //this is an instance of
FlexDataGridDataCell (For data rows)
        var column = cell.getColumn();//this is an instance of
FlexDataGridColumn.
        //var dp = cell.level.getGrid().getDataProvider();//this is a
pointer back to the grid and its dataprovider.
        var dp=this.data;//for header cells, specifically in case of
nested grids, the data property is a pointer back to the top level
array, or the children array

        if(this.data.hasOwnProperty("deals")){
            //this means we are at a inner level checkbox header
            dp=this.data.deals;
        }
        //based upon which level this renderer appears.
        for (var i=0;i<dp.length;i++){
            dp[i][column.getDataField()] = this.domElement.checked;
        }

        column.level.grid.refreshCells();//this will re-render the
cells.
    };
    //This sets the inner html, and grid will try to set it. Since
we are an input field, IE 8 will complain. So we ignore it since we
dont need it anyway.
    CheckBoxHeaderRenderer.prototype.setText=function(val){

```



```

    };
}(window));

$(document).ready(function(){
    var grid = new
flexiciousNmosp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        configuration: '<grid id="grid"
enablePrint="true" enablePreferencePersistence="true"
enableExport="true" forcePagerRow="true" pageSize="50"
enableFilters="true" enableFooters="true" >' +
                        '                <level>' +
                        '                <columns>' +
                        '                <column
dataField="id" headerText="ID" filterControl="TextInput"
filterOperation="Contains" footerLabel="Sum: " footerOperation="sum"
footerOperationPrecision="2"/>' +
                        '                <column
dataField="type" headerText="Type" filterControl="TextInput"
filterOperation="Contains"
footerLabelFunction="myCompanyNameSpace.customFooterFunction"/>' +
                        '                <column
dataField="" headerText="Label Function Example"
filterControl="TextInput" filterOperation="Contains"
labelFunction="myCompanyNameSpace.labelFunctionExample"/>' +
                        '                <column
dataField="" headerText="Label Function 2 Example"
filterControl="TextInput" filterOperation="Contains"
labelFunction2="myCompanyNameSpace.labelFunction2Example"/>' +
                        '                <column
dataField="active" headerText="Item Renderer Example"
headerRenderer="myCompanyNameSpace.CheckBoxHeaderRenderer"
filterControl="TextInput" filterOperation="Contains"
itemRenderer="myCompanyNameSpace.CheckBoxRenderer"/>' +
                        '                </columns>' +
                        '                </level>' +
                        '            ' +
                        '        </grid>',
        dataProvider: [
            { "id": "5001", "type": "None",
"active" : true },
            { "id": "5002", "type": "Glazed",
, "active" : true},
            { "id": "5005", "type": "Sugar"
, "active" : true},
            { "id": "5007", "type":
"Powdered Sugar" , "active" : false},
            { "id": "5006", "type":
"Chocolate with Sprinkles" , "active" : true},

```

```

        "Chocolate" , "active" : false},
        , "active" : true}
    ]
    });

});
</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px;width: 100%;">

        </div>
</body>
</html>

```

## Configuration Options – Interactive HTML in cells

In the previous section we look at a very common requirement, one that of formatting column values. We saw that there are numerous ways to customize the content of the cells. We explored a rich API that allows you to pretty much customize every single grid cell.

In this section, let's take that concept further, and embed interactive content within cells. By interactive content, we mean things like tooltips, programmatic interactive html, etc. We will continue to use the example from the previous section, just enhance the label function like so:

```
myCompanyNameSpace.labelFunctionExample = function (item,
column) {
    var html = "<a href='#' onclick='showTooltip(this)'" + " Click
:" + column.getHeaderText() + " for tooltip : </a> Item: " + item.type ;
    return html;
};

function showTooltip(trigger) {
    alert(" Show Tooltip for " +
trigger.parentNode.parentNode.component.rowInfo.getData().type)
}
```

We will also make one small change to the configuration XML:

```
dataField="" headerText="Label Function Example"
enableCellClickRowSelect="false" filterControl="TextInput"
filterOperation="Contains"
labelFunction="myCompanyNameSpace.labelFunctionExample"/>' +
```

And here is what we see:

	Label Function Example	Label Function 2 Exa
	<a href="#">Click :Label Function Example for tooltip : Item: None</a>	Column:Label Functi
	<a href="#">Click :Label Function Example for tooltip : Item: Glazed</a>	Column:Label Functi
	<a href="#">Click :Label Function Example for tooltip : Item: Sugar</a>	Column:Label Functi
	<a href="#">Click :Label Function Example for tooltip : Item: Powdered Sugar</a>	Column:Label Functi
	<a href="#">Click :Label Function Example for tooltip : Item: Chocolate with S</a>	Column:Label Functi
	<a href="#">Click :Label Function Example for tooltip : Item: Chocolate</a>	Column:Label Functi
	<a href="#">Click :Label Function Example for tooltip : Item: Maple</a>	Column:Label Functi

JavaScript Alert

Show Tooltip for Chocolate with Sprinkles

OK

So let's look at what we did here.

First, we set `enableCellClickRowSelect="false"`

This prevents the click on the anchor from selecting the row.

Second, in the HTML we generate, we added an onClick handler:

```
var html = "<a href='#' onclick='showTooltip(this)'"> Click
: "+column.getHeaderText()+" for tooltip : </a> Item: " + item.type ;
return html;
```

Finally, we provided the callback for the handler:

```
function showTooltip(trigger) {
    alert(" Show Tooltip for " +
trigger.parentNode.parentNode.component.rowInfo.getData().type)
}
```

There are a few things to keep in mind in this method.

1. First, we pass in the anchor that the user clicked on. This lets us get a handle on what the user clicked on.
2. Second, we navigate up the HTML hierarchy until we reach the DIV component that map to the FlexDataGridDataCell object. This div has a component property that is the call back that JavaScript has to reach into the FlexDataGrid API. This gives you the hook into the FlexDataGridCell, which has a reference to its rowInfo object. This is an object of class flexiciousNmosp.RowInfo, that has among other things, a `getData()` method, that will get the row data that you are displaying in the row.
3. Once you have a handle to the row data, you can do whatever you wish with it, in this

case, we simply show it as an alert.

## Styling and Theming

In the previous section we embedded interactive content within cells. By interactive content, we mean things like tooltips, programmatic interactive html, etc. In this section, we will explore the two different mechanisms for styling and theming the grid.

The entire look and feel of the grid, every single cell, is fully customizable. The markup, the css, everything is customizable. There are basically two ways to accomplish this.

### Styling Using CSS

The HTMLTreegrid ships with its own css file, Flexicious.css. This is located at <http://www.htmltreegrid.com/demo/flexicious/css/flexicious.css>. You can pretty much customize all the elements in there and affect all instances of your grid.

For example, if you simply add the following stylesheet to your html page:

```
.flexiciousGrid .flexDataGridCell .cellRenderer {
    font-size:12;
    font-weight:bold;
    font-style : italic;
    font-family:Tahoma;
}
```

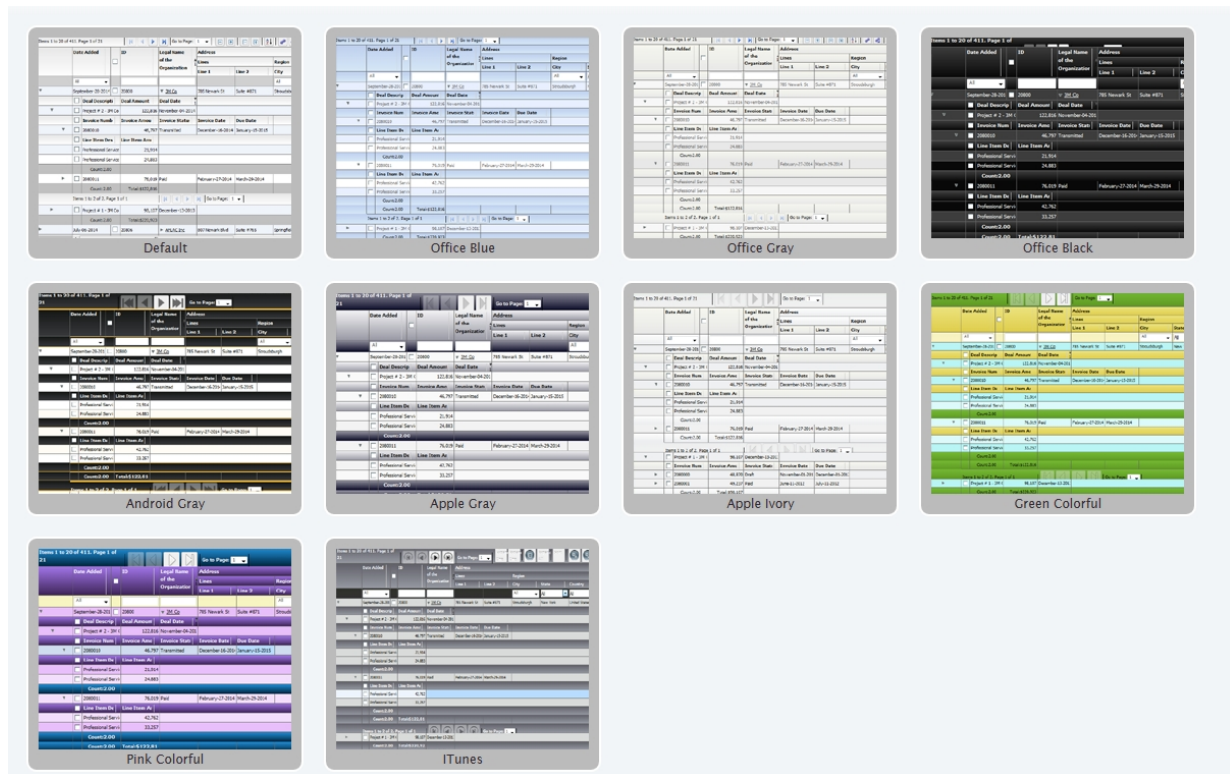
ID	Type	Label Function Example	Label Function 2 Example	
5001	None	Click Label Function Example for tooltip: Item: None	Column:Label Function 2 Example: Item: None, cell: 0	✓
5002	Glazed	Click Label Function Example for tooltip: Item: Glazed	Column:Label Function 2 Example: Item: Glazed, cell: 1	✓
5005	Sugar	Click Label Function Example for tooltip: Item: Sugar	Column:Label Function 2 Example: Item: Sugar, cell: 2	✓
5007	Powdered Sugar	Click Label Function Example for tooltip: Item: Powder	Column:Label Function 2 Example: Item: Powdered Sug	✓
5006	Chocolate with Sprinkles	Click Label Function Example for tooltip: Item: Chocola	Column:Label Function 2 Example: Item: Chocolate with	✓
5003	Chocolate	Click Label Function Example for tooltip: Item: Chocola	Column:Label Function 2 Example: Item: Chocolate, cel	✓
5004	Maple	Click Label Function Example for tooltip: Item: Maple	Column:Label Function 2 Example: Item: Maple, cell: 6	✓
Sum: 35028.00	Look Mommy, custom footer!			

There are a few limitations of the css method. The grid itself exposes a rich API for styling purposes. It has properties exposed for padding, borders, background, etc. This allows you to programmatically control these properties in a very intuitive way. This API will always win over css. This is because the result of this is applied directly on the element. We will cover this in the next section.

### Styling Using Markup and API

The grids API offers a much more powerful programmatic model for a number of requirements that are customary in line of business applications. For an example of this programmatic control, please refer to the “Programmatic cell formatting” example in the main demo.

The best application of this mechanism is the themes that are built in to the app. Out of the box; we ship the product with almost a dozen professional looking themes as you can see in the screenshot below:



How do you use these themes? First of all, ensure that you have the themes.js file imported in your html page.

```
<script type="text/javascript" src="http://www.htmltreegrid.com/demo/themes.js"></script>
```

Finally, add the styles declaration to your grid. (In this example we are using officeblue, but if you look at themes.js there is the full list):

```
$(document).ready(function(){
    var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        configuration: . . .,
        dataProvider: [ . . . ],
        styles :
flexiciousNmsp.UIUtils.getThemeById('officeblue').styles
    });
});
```

This gives you a grid that looks like:

Flexicious HTMLTreeGrid

ID	Type	Label Function Example	Label Function 2 Example						
5001	None	<a href="#">Click Label Function Example for tooltip : Item: None</a>	ColumnLabel Function 2 Example : Item: None, cell: 0						
5002	Glazed	<a href="#">Click Label Function Example for tooltip : Item: Glazed</a>	ColumnLabel Function 2 Example : Item: Glazed, cell: 1						
5005	Sugar	<a href="#">Click Label Function Example for tooltip : Item: Sugar</a>	ColumnLabel Function 2 Example : Item: Sugar, cell: 2						
5007	Powdered Sugar	<a href="#">Click Label Function Example for tooltip : Item: Powdered Sugar</a>	ColumnLabel Function 2 Example : Item: Powdered Sugar,						
5006	Chocolate with Sprinkles	<a href="#">Click Label Function Example for tooltip : Item: Chocolate with S</a>	ColumnLabel Function 2 Example : Item: Chocolate with Sp						
5003	Chocolate	<a href="#">Click Label Function Example for tooltip : Item: Chocolate</a>	ColumnLabel Function 2 Example : Item: Chocolate, cell: 5						
5004	Maple	<a href="#">Click Label Function Example for tooltip : Item: Maple</a>	ColumnLabel Function 2 Example : Item: Maple, cell: 6						
Sum: 35028.00	Look Mommy, custom footer!								

Isn't that pretty? A full list of themes is located in themes.js.

## Creating your own theme

Creating your own theme is as simple as adding a JavaScript object to the flexiciousNmsp.themes array.

```
flexiciousNmsp.themes.push(
  {id:'redAndBlack', name:'redAndBlack',
   styles:{
     /**
      * Usually the toolbar root is the same as the images
      root, but for some custom themes, we have their own icons.
      */
     toolbarImagesRoot:flexiciousNmsp.Constants.IMAGE_PATH +
"/themeIcons/itunes/32",
     pagerRowHeight : 50,
     pagerStyleName:"whiteText largeIcons",
     headerStyleName:"whiteText",
     columnGroupStyleName:"whiteText",
     footerStyleName:"whiteText",
     alternatingItemColors: [0xFFFFFFFF, 0xFFFFFFFF],
     alternatingTextColors: [0x000000, 0x000000],
     selectionColor: [0xFABB39, 0xFABB39],
     rolloverColor: 0xCEDBEF,
     headerRolloverColors: [0x1C1E1D, 0x3A3B3D],
     headerColors: [0x1C1E1D, 0x3A3B3D],
     columnGroupRolloverColors: [0x1C1E1D, 0x3A3B3D],
     columnGroupColors: [0x1C1E1D, 0x3A3B3D],
     pagerRolloverColors: [0x1C1E1D, 0x3A3B3D],
     pagerColors: [0x1C1E1D, 0x3A3B3D],
     footerRolloverColors: [0x1C1E1D, 0x3A3B3D],
     footerColors: [0x1C1E1D, 0x3A3B3D],
     filterRolloverColors: [0x1C1E1D, 0x3A3B3D],
     filterColors: [0x1C1E1D, 0x3A3B3D],
     fixedColumnFillColor: [0xEFEFEF, 0xEFEFEF],
     activeCellColor: 0xB7DBFF,
     rendererRolloverColors: [0xFFFFFFFF, 0xFFFFFFFF],
     rendererColors: [0xFFFFFFFF, 0xFFFFFFFF],
     textSelectedColor: 0x000000,
     textRolloverColor: 0x000000,
     borderColor: 0xFF0000,
     columnGroupVerticalGridLineColor: 0xFF0000,
     columnGroupVerticalGridLines:true,
     columnGroupVerticalGridLineThickness:1,

     columnGroupHorizontalGridLineColor: 0xFF0000,
     columnGroupHorizontalGridLines:true,
     columnGroupHorizontalGridLineThickness:1,
     columnGroupDrawTopBorder:false,
```

```

headerVerticalGridLineColor: 0xFF0000,
headerVerticalGridLines:true,
headerVerticalGridLineThickness:1,

headerHorizontalGridLineColor: 0xFF0000,
headerHorizontalGridLines:true,
headerHorizontalGridLineThickness:1,
headerDrawTopBorder:false,
headerSortSeparatorRight:16,

filterVerticalGridLineColor: 0xFF0000,
filterVerticalGridLines:true,
filterVerticalGridLineThickness:1,

filterHorizontalGridLineColor: 0xFF0000,
filterHorizontalGridLines:true,
filterHorizontalGridLineThickness:1,
filterDrawTopBorder:false,

footerVerticalGridLineColor: 0xFF0000,
footerVerticalGridLines:true,
footerVerticalGridLineThickness:1,

footerHorizontalGridLineColor: 0xFF0000,
footerHorizontalGridLines:false,
footerHorizontalGridLineThickness:1,
footerDrawTopBorder:false,

pagerVerticalGridLineColor: 0xFF0000,
pagerVerticalGridLines:true,
pagerVerticalGridLineThickness:1,

pagerHorizontalGridLineColor: 0xFF0000,
pagerHorizontalGridLines:true,
pagerHorizontalGridLineThickness:1,

renderVerticalGridLineColor: 0xFF0000,
renderVerticalGridLines:true,
renderVerticalGridLineThickness:1,

renderHorizontalGridLineColor: 0xFF0000,
renderHorizontalGridLines:true,
renderHorizontalGridLineThickness:1,
renderDrawTopBorder:false,
    }
})

```

Since in our custom theme we are referring to a custom css class, we define it as well:



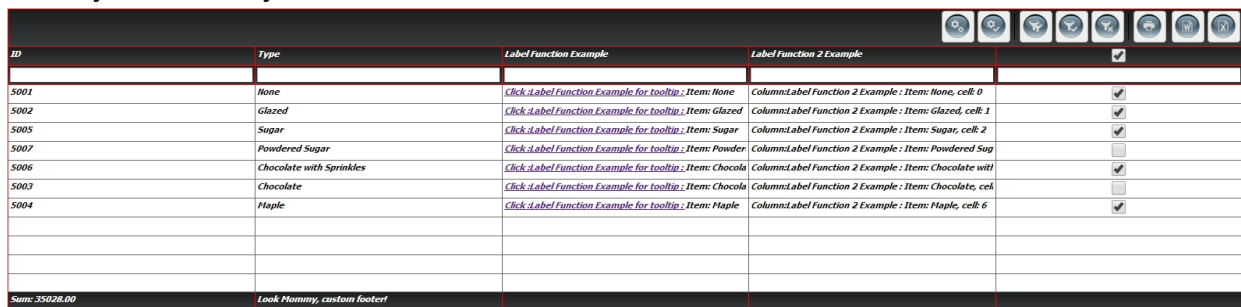
```
<style>
    .whiteText
    {
        color:#ffffff;
    }
</style>
```

And then finally:

```
$(document).ready(function(){
    var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        configuration: . . .,
        dataProvider: [. . . ],
        styles :
flexiciousNmsp.UIUtils.getThemeById('redAndBlack').styles
    });
});
```

Note, you don't need to add the object to the themes array if you don't want to. You can simply pass in any object that has the properties that match to the styles parameter.

When you run this, you should see:



ID	Type	Label Function Example	Label Function 2 Example	
5001	None	Click a Label Function Example for tooltip: Item: None	Column:Label Function 2 Example : Item: None, cell: 0	<input checked="" type="checkbox"/>
5002	Glazed	Click a Label Function Example for tooltip: Item: Glazed	Column:Label Function 2 Example : Item: Glazed, cell: 1	<input checked="" type="checkbox"/>
5005	Sugar	Click a Label Function Example for tooltip: Item: Sugar	Column:Label Function 2 Example : Item: Sugar, cell: 2	<input checked="" type="checkbox"/>
5007	Powdered Sugar	Click a Label Function Example for tooltip: Item: Powder	Column:Label Function 2 Example : Item: Powdered Sug	<input type="checkbox"/>
5006	Chocolate with Sprinkles	Click a Label Function Example for tooltip: Item: Chocola	Column:Label Function 2 Example : Item: Chocolate with	<input checked="" type="checkbox"/>
5003	Chocolate	Click a Label Function Example for tooltip: Item: Chocola	Column:Label Function 2 Example : Item: Chocolate, cel	<input type="checkbox"/>
5004	Haple	Click a Label Function Example for tooltip: Item: Haple	Column:Label Function 2 Example : Item: Haple, cell: 6	<input checked="" type="checkbox"/>
Sum: 35028.00		Look Mommy, custom footer!		

The entire list of options is quite extensive, and is fully documented here: <http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html>

Below is the entire code for what we have so far:

```
<!doctype html>
<html lang="en" ng-app="myApp">
<head>
    <meta charset="utf-8">
    <title>Simple</title>
    <!--These are jquery and plugins that we use from jquery-->
    <script type="text/javascript" src="http://www.htmltreegrid.com/
```

```

demo/external/js/adapters/jquery/jquery-1.8.2.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-ui-1.9.1.custom.min.js"></
script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.ui.menu.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.toaster.js"></script>
  <!--End-->
  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/minified-compiled-jquery.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/examples/js/Configuration.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/themes.js"></script>
  <!--End-->
  <!--css imports-->
  <link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
flexicious/css/flexicious.css"
    type="text/css" />
  <link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
external/css/adapters/jquery/jquery-ui-1.9.1.custom.min.css"
    type="text/css" />
  <!--End-->
  <style>
    .flexiciousGrid .flexDataGridCell .cellRenderer
    {
      font-size: 12;
      font-weight: bold;
      font-style: italic;
      font-family: Tahoma;
    }
    .whiteText
    {
      color: #ffffff;
    }
  </style>
  <script type="text/javascript">

    flexiciousNmosp.themes.push(
      {id: 'redAndBlack', name: 'redAndBlack',
      styles: {
        /**
         * Usually the toolbar root is the same as the images
         root, but for some custom themes, we have their own icons.

```

```

*/
toolbarImagesRoot:flexiciousNmsp.Constants.IMAGE_PATH +
"/themeIcons/itunes/32",
pagerRowHeight : 50,
pagerStyleName:"whiteText largeIcons",
headerStyleName:"whiteText",
columnGroupStyleName:"whiteText",
footerStyleName:"whiteText",
alternatingItemColors: [0xFFFFFFFF, 0xFFFFFFFF],
alternatingTextColors: [0x000000, 0x000000],
selectionColor: [0xFABB39, 0xFABB39],
rollOverColor: 0xCEDBEF,
headerRollOverColors: [0x1C1E1D, 0x3A3B3D],
headerColors: [0x1C1E1D, 0x3A3B3D],
columnGroupRollOverColors: [0x1C1E1D, 0x3A3B3D],
columnGroupColors: [0x1C1E1D, 0x3A3B3D],
pagerRollOverColors: [0x1C1E1D, 0x3A3B3D],
pagerColors: [0x1C1E1D, 0x3A3B3D],
footerRollOverColors: [0x1C1E1D, 0x3A3B3D],
footerColors: [0x1C1E1D, 0x3A3B3D],
filterRollOverColors: [0x1C1E1D, 0x3A3B3D],
filterColors: [0x1C1E1D, 0x3A3B3D],
fixedColumnFillColor: [0xEFEFEF, 0xEFEFEF],
activeCellColor: 0xB7DBFF,
rendererRollOverColors: [0xFFFFFFFF, 0xFFFFFFFF],
rendererColors: [0xFFFFFFFF, 0xFFFFFFFF],
textSelectedColor: 0x000000,
textRollOverColor: 0x000000,
borderColor: 0xFF0000,
columnGroupVerticalGridLineColor: 0xFF0000,
columnGroupVerticalGridLines:true,
columnGroupVerticalGridLineThickness:1,

columnGroupHorizontalGridLineColor: 0xFF0000,
columnGroupHorizontalGridLines:true,
columnGroupHorizontalGridLineThickness:1,
columnGroupDrawTopBorder:false,

headerVerticalGridLineColor: 0xFF0000,
headerVerticalGridLines:true,
headerVerticalGridLineThickness:1,

headerHorizontalGridLineColor: 0xFF0000,
headerHorizontalGridLines:true,
headerHorizontalGridLineThickness:1,
headerDrawTopBorder:false,
headerSortSeparatorRight:16,

filterVerticalGridLineColor: 0xFF0000,

```

```

        filterVerticalGridLines:true,
        filterVerticalGridLineThickness:1,

        filterHorizontalGridLineColor: 0xFF0000,
        filterHorizontalGridLines:true,
        filterHorizontalGridLineThickness:1,
        filterDrawTopBorder:false,

        footerVerticalGridLineColor: 0xFF0000,
        footerVerticalGridLines:true,
        footerVerticalGridLineThickness:1,

        footerHorizontalGridLineColor: 0xFF0000,
        footerHorizontalGridLines:false,
        footerHorizontalGridLineThickness:1,
        footerDrawTopBorder:false,

        pagerVerticalGridLineColor: 0xFF0000,
        pagerVerticalGridLines:true,
        pagerVerticalGridLineThickness:1,

        pagerHorizontalGridLineColor: 0xFF0000,
        pagerHorizontalGridLines:true,
        pagerHorizontalGridLineThickness:1,

        rendererVerticalGridLineColor: 0xFF0000,
        rendererVerticalGridLines:true,
        rendererVerticalGridLineThickness:1,

        rendererHorizontalGridLineColor: 0xFF0000,
        rendererHorizontalGridLines:true,
        rendererHorizontalGridLineThickness:1,
        rendererDrawTopBorder:false,
    }
})

var myCompanyNameSpace = {};

myCompanyNameSpace.labelFunctionExample = function (item,
column) {
    var html = "<a href='#' onclick='showTooltip(this)'"> Click
:"+column.getHeaderText()+" for tooltip : </a> Item: " + item.type ;
    return html;
};

myCompanyNameSpace.labelFunction2Example = function (item,
column, cell) {
    var html = "<b> Column:"+column.getHeaderText()+" : </b>
Item: " + item.type + ", cell: " +
cell.rowInfo.rowPositionInfo.getRowIndex();
    return html;
};

```

```

        myCompanyNameSpace.customFooterFunction = function () {
            var html = "<div> Look Mommy, custom footer!</div>";
            return html;
        };
        function showTooltip(trigger) {
            alert(" Show Tooltip for " +
trigger.parentNode.parentNode.component.rowInfo.getData().type)
        }

        /**
         * Flexicious
         * Copyright 2011, Flexicious LLC
         */
        (function(window)
        {
            "use strict";
            var CheckBoxRenderer, uiUtil = flexiciousNmsp.UIUtils,
            flxConstants = flexiciousNmsp.Constants;
            /**
             * A CheckBoxRenderer is a custom item renderer, that defines how
             to use custom cells with logic that you can control
             * @constructor
             * @namespace flexiciousNmsp
             * @extends UIComponent
             */
            CheckBoxRenderer=function(){
                //make sure to call constructor
                flexiciousNmsp.UIComponent.apply(this,["input"]); //second
parameter is the tag name for the dom element.
                this.domElement.type = "checkbox"; //so our input element
becomes a checkbox;

                /**
                 * This is a getter/setter for the data property. When the
                 cell is created, it belongs to a row
                 * The data property points to the item in the grids
                 dataprovider that is being rendered by this cell.
                 * @type {*}
                 */
                this.data=null;

                //the add evt listener will basically proxy all DomEvents to
your code to handle.
                this.addEventListener(this,flxConstants.EVENT_CHANGE,this.onC
hange);
            };
            myCompanyNameSpace.CheckBoxRenderer = CheckBoxRenderer; //add to
name space
            CheckBoxRenderer.prototype = new flexiciousNmsp.UIComponent(); //
setup hierarchy
            CheckBoxRenderer.prototype.typeName = CheckBoxRenderer.typeName

```

```

= 'CheckBoxRenderer';//for quick inspection
    CheckBoxRenderer.prototype.getClassName=function(){
        return ["CheckBoxRenderer","UIComponent"]; //this is a
mechanism to replicate the "is" and "as" keywords of most other OO
programming languages
    };

    /**
     * This is important, because the grid looks for a "setData"
method on the renderer.
     * In here, we intercept the call to setData, and inject our
logic to populate the text input.
     * @param val
     */
    CheckBoxRenderer.prototype.setData=function(val){
        flexiciousNmsp.UIComponent.prototype.setData.apply(this,
[val]);
        var cell = this.parent; //this is an instance of
FlexDataGridDataCell (For data rows)
        var column = cell.getColumn();//this is an instance of
FlexDataGridColumn.
        this.domElement.checked=this.data[column.getDataField()];
    };
    /**
     * This event is dispatched when the user clicks on the icon. The
event is actually a flexicious event, and has a trigger event
     * property that points back to the original domEvent.
     * @param event
     */
    CheckBoxRenderer.prototype.onChange=function(evt){

        //in the renderer, you have the handle to the cell that the
renderer belongs to, via the this.parent property that you inherit
from flexiciousNmsp.UIComponent.

        var cell = this.parent; //this is an instance of
FlexDataGridDataCell (For data rows)
        var column = cell.getColumn();//this is an instance of
FlexDataGridColumn.

        this.data[column.getDataField()]=this.domElement.checked;//we
use the dom element to wire back the value to the data object.
    };
    //This sets the inner html, and grid will try to set it. Since
we are an input field, IE 8 will complain. So we ignore it since we
dont need it anyway.
    CheckBoxRenderer.prototype.setText=function(val){

    };
}(window));

```

```

/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function(window)
{
    "use strict";
    var CheckBoxHeaderRenderer, uiUtil = flexiciousNmsp.UIUtils,
    flxConstants = flexiciousNmsp.Constants;
    /**
     * A CheckBoxHeaderRenderer is a custom item renderer, that you
     can use in a header cell. In this case, we customize the header
     * so that instead of showing a header label, we show a checkbox
     that switches the dataField flag on all items.
     * @constructor
     * @namespace flexiciousNmsp
     * @extends UIComponent
     */
    CheckBoxHeaderRenderer=function(){
        //make sure to call constructor
        flexiciousNmsp.UIComponent.apply(this,["input"]); //second
parameter is the tag name for the dom element.
        this.domElement.type = "checkbox"; //so our input element
becomes a checkbox;
        this.domElement.checked=true;
        //the add event listener will basically proxy all DomEvents
to your code to handle.
        this.addEventListener(this, flxConstants.EVENT_CHANGE, this.onC
hange);
    };
    myCompanyNameSpace.CheckBoxHeaderRenderer =
CheckBoxHeaderRenderer; //add to name space
    CheckBoxHeaderRenderer.prototype = new
flexiciousNmsp.UIComponent(); //setup hierarchy
    CheckBoxHeaderRenderer.prototype.typeName =
CheckBoxHeaderRenderer.typeName = 'CheckBoxHeaderRenderer'; //for
quick inspection
    CheckBoxHeaderRenderer.prototype.getClassName=function(){
        return ["CheckBoxHeaderRenderer", "UIComponent"]; //this is a
mechanism to replicate the "is" and "as" keywords of most other OO
programming languages
    };

    /**
     * This event is dispatched when the user clicks on the icon. The
     event is actually a flexicious event, and has a trigger event
     * property that points back to the original domEvent.
     * @param event
     */
    CheckBoxHeaderRenderer.prototype.onChange=function(event){

```

//in the renderer, you have the handle to the cell that the renderer belongs to, via the this.parent property that you inherit from flexiciousNmsp.UIComponent.

```

    var cell = this.parent; //this is an instance of
    FlexDataGridDataCell (For data rows)
    var column = cell.getColumn();//this is an instance of
    FlexDataGridColumn.
    //var dp = cell.level.getGrid().getDataProvider();//this is a
    pointer back to the grid and its dataprovider.
    var dp=this.data;//for header cells, specifically in case of
    nested grids, the data property is a pointer back to the top level
    array, or the children array

    if(this.data.hasOwnProperty("deals")){
        //this means we are at a inner level checkbox header
        dp=this.data.deals;
    }
    //based upon which level this renderer appears.
    for (var i=0;i<dp.length;i++){
        dp[i][column.getDataField()] = this.domElement.checked;
    }

    column.level.grid.refreshCells();//this will re-render the
    cells.
    };
    //This sets the inner html, and grid will try to set it. Since
    we are an input field, IE 8 will complain. So we ignore it since we
    dont need it anyway.
    CheckBoxHeaderRenderer.prototype.setText=function(val){
};
}(window));

```

```

$(document).ready(function(){
    var grid = new
    flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        configuration: '<grid id="grid"
        enablePrint="true" enablePreferencePersistence="true"
        enableExport="true" forcePagerRow="true" pageSize="50"
        enableFilters="true" enableFooters="true" >' +
            '                <level>' +
            '                    <columns>' +
            '                        <column
        dataField="id" headerText="ID" filterControl="TextInput"
        filterOperation="Contains" footerLabel="Sum: " footerOperation="sum"
        footerOperationPrecision="2"/>' +

```



```

        '
        <column
dataField="type" headerText="Type" filterControl="TextInput"
filterOperation="Contains"
footerLabelFunction="myCompanyNameSpace.customFooterFunction"/>' +
        '
        <column
dataField="" headerText="Label Function Example"
enableCellClickRowSelect="false" filterControl="TextInput"
filterOperation="Contains"
labelFunction="myCompanyNameSpace.labelFunctionExample"/>' +
        '
        <column
dataField="" headerText="Label Function 2 Example"
filterControl="TextInput" filterOperation="Contains"
labelFunction2="myCompanyNameSpace.labelFunction2Example"/>' +
        '
        <column
dataField="active" headerText="Item Renderer Example"
headerRenderer="myCompanyNameSpace.CheckBoxHeaderRenderer"
filterControl="TextInput" filterOperation="Contains"
itemRenderer="myCompanyNameSpace.CheckBoxRenderer"/>' +
        '
        </columns>' +
        '
        </level>' +
        '
        ' +
        '</grid>',
    dataProvider: [
        { "id": "5001", "type": "None",
"active" : true },
        { "id": "5002", "type": "Glazed",
, "active" : true},
        { "id": "5005", "type": "Sugar",
, "active" : true},
        { "id": "5007", "type":
"Powdered Sugar" , "active" : false},
        { "id": "5006", "type":
"Chocolate with Sprinkles" , "active" : true},
        { "id": "5003", "type":
"Chocolate" , "active" : false},
        { "id": "5004", "type": "Maple"
, "active" : true}
    ],
    styles:
flexiciousNmsp.UIUtils.getThemeById('redAndBlack').styles
    });
});
</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px; width: 100%;">
    </div>
</body>
</html>

```

## Connecting To Data

The HTMLTreeGrid is designed to connect to any kind of data source as long as the data can get to the browser in form of an array of simple JavaScript objects. How it gets from the server to a javascript object is completely upto you. When you download a trial, you get our entire demo console – dozens of examples that help you get up and running quickly. However, the demo console does not connect to a live data source. We did this because we did not want you to have to go through the setup process for a backend server to be able to evaluate the product. This way you can also evaluate the product without having any network connectivity.

### Basic Concepts

Once you get your feet wet with the demo console, its time to look further into actual live examples. Majority of the times, the back end sends data in JSON format. Although this is not necessary, and the grid is able to consume XML data as well. Just look at the XML Data and XML Grouped data examples. One thing to note, is that even if you send the grid XML, internally it converts it to JSON prior to rendering using the XML2Json library. That said, there are a number of use cases that the grid supports that involve complex interactions with the back-end. Lets take a look at some of these use cases, and along the way learn how to implement them with various different back-end technologies.

As most of you are aware the HTMLTreeGrid supports two types of data flat and hierarchical. In the next couple of sections, we will see what each of these means.

### Flat Data

Flat data is basically a javascript array of JavaScript objects. imagine an array of objects like below

```
<script type="text/javascript">
    $(document).ready(function(){
        var grid = new
flexiciousNmosp.FlexDataGrid(document.getElementById("gridContainer"),
        {
            configuration: '<grid id="grid" enablePrint="true"
enablePreferencePersistence="true" enableExport="true" forcePagerRow="true" pageSize="50"
enableFilters="true" enableFooters="true" >' +
                '
                <level>' +
                <columns>' +
                <column
dataField="id" headerText="ID" />' +
                <column
dataField="type" headerText="Type"/>' +
                </columns>' +
                </level>' +
                ' +
                </grid>',
            dataProvider: [
                { "id": "5001", "type": "None" },
                { "id": "5002", "type": "Glazed" },
                { "id": "5005", "type": "Sugar" },
                { "id": "5007", "type": "Powdered
Sugar" },
                { "id": "5006", "type": "Chocolate with
Sprinkles" },
                { "id": "5003", "type": "Chocolate" },
                { "id": "5004", "type": "Maple" }
            ]
        });
    });
```

</script>

We have seen this in numerous examples so far where we supply a list of JavaScript objects via the provider parameter. You can also supply the same thing using the setDataProvider method.

## Flat Data - Lazy Load - Dot Net Sample

Among the most powerful features of the grid is the building support for lazy loading of large datasets. For flat data this is done via the use of the filterPageSortMode property. This is a key property to understand. There are two values for this property, "server" and "client". Let us assume you are working on a Human Resource Management Application. You have a DataGrid that shows a list of Employees. You have a few hundred Employees to show. This can be easily accomplished by getting the list of employees from the server, and setting the data provider property of the grid to an Array that contains this list of Employees. As you need, you enable filters, footers, paging, export, print, etc, and all is well and good. You will be up and running in no time with filterPageSortMode=client (The default setting). The grid will take care of paging, filtering and cross page sorting for you. However, now consider a scenario where you have to display a time sheet search page. Each employee creates hundreds of time-sheets each year, and multiply that by the number of employees, you are looking at thousands, even hundreds of thousands of records of data. Although it may be possible to load thousands of records in any DataGrid (including ours) with no noticeable drag, this is not recommended, and often unnecessary. What we suggest in this scenario is you use the filterPageSortMode ="server". What the product does, in this setup, is it assumes that the current Array is a part of a much larger record-set, and you are showing just the current page of data. So if there are 100,000 records in the database, and the pageSize is 50, the grid will show the 50 records, and provide paging UI for the user to navigate to any record in the result-set. At any point in time, no more than 50 records will be loaded on client memory. This setup, will require a little more effort in terms of configuration, but it will be considerably easier to do this with our grid as opposed to building it yourself, because the product is architected to cater to a scenario like this.

### **filterPageSortMode=client**

When this property is set to client mode, there's very little work that you have to do. Just give us an array of objects and the grid takes care of paging sorting, filtering, export and everything.. We are able to do this because we have all the data on the client in memory. so when your filter we can run through the entire data set and get you the records that you're filtering on. The same applies to the sort. Since all the pages of data are re loaded in memory all we have to do is to navigate the grid to the page that your request when you click on the paging buttons. similarly when you click on the print or the export buttons we have the entire dataset loaded in memory so were able to export or print all the data without any interaction from the server.

### **filterPageSortMode=server**

However in server mode, things are a lot different. At any point in time we only have the current page of data in memory. This means when you click on the paging buttons we have to go back to the server to get the requested page of data. When you click on the sort headers we have to run the sort on the server and get only the current page of data according to the updated sort criteria. Along the same lines when you run a filter, we have to go back to the server and run the filter on the server potentially converting it into a SQL statement and returning the results of that SQL statement on basis of the current page filter and sort criteria. Finally when you click on the train or the export buttons and choose to export the entire data set as opposed to the current page we have to go back to the server and get the entire data set for the print or the export. There are a few things you have to do to make this happen. In this document, we are only going to focus on the client side of things (the server side is technology dependent, and we provide samples of this independent of this document.)

1. **filterPageSortMode** : Needless to say, this needs to be set to "server".
2. **The filterPageSortChange Event:** You have to wire up the "filterPageSortChange" event. This is the event that get dispatched when the user user clicks on the sort header on any of the columns, or request a change using either the page navigation buttons or the page navigation drop down in the toolbar, or runs a filter with in any of the columns. This event has 2 properties that are of interest:
  - **event.filter**: This object contains all the information that you would potentially need to construct a SQL statement on the backend. Full documentation on this object can be found

here:

<http://www.flexicious.com/resources/docs29/com/flexicious/grids/filters/Filter.html>

- **event.cause** - This can be one of the three values:  
 public static const FILTER\_CHANGE:String = filterChange  
 public static const PAGE\_CHANGE:String = pageChange  
 public static const SORT\_CHANGE:String = sortChange
- 3. **The printExportDataRequest Event:** If you want to support exporting of the entire dataset (all pages), then you also have to wire up the printExportDataRequest event. This is the same as filterPageSortChange event, in that it has a event.filter object, that in addition to the filter, also contains **printExportOptions object**.
- 4. **The selectedKeyField:** Also, another key aspect - is the selectedKeyField. You have to wire this up if you want to maintain selection across pages. The selectedKeyField is a property on the object that identifies the object uniquely. Similar to a surrogate key, or a business key, as long as it uniquely identifies the object. When this property is set, the selected keys returns the ID values of the objects that were selected. When a row is selected in the grid, we store the selectedKeyField property of the selected object in this array collection. This allows for multiple pages of data that comes down from the server and not maintained in memory to still keep track of the ids that were selected on other pages. If you use Flexicious in filterPageSortMode=client, this really does not apply to you, but in server mode, each page of data potentially represents a brand new dataprovider. Let's assume you have a Database table of 100,000 records, with the pageSize property set on Flexicious to 50. You load page 1, select a few items, and move on to page 2. The grid exposes a property called selectedItems, which will be lost when the new page of data is loaded. This is why we have the selectedObjects and selectedKeys property, that is, to keep the selection that was loaded in memory on prior pages of data. Now, in most LOB applications, each record can be identified by a surrogate key (or some unique identifier). This surrogate key is then used to uniquely identify different instances of at the same Object. For example, when the page 1 is loaded for the first time, there is a Employee OBJECT with EmployeeId=1. When the user selects this record, navigates to a different page, and switches back to page 1, the Employee with ID 1 is still there, and need to stay selected, but it could be an altogether different INSTANCE of the same record in the database. This is why we have the selectedKeyField property, which would in this case, be "EmployeeID" so we can uniquely identify the selection made by the user across multiple pages.
- 5. **filterComboBoxDataProvider where filterControl="MultiSelectComboBox" or "ComboBox":**  
 These have lookup based filters. Since at any time, we only load the top level filter, we need to query the database for all possible values for this pickers. This is not a problem with filterPageSortMode=client, because we load up the entire dataset and run a distinct on it to figure out the values for the picker. However with server based filterPageSortMode, we need to query the server to get the entire list of possible values to pick from.

One additional caveat: With filterPageSortMode=server grid, the automatic calculations built into the Grid cannot evaluate the footer values. So for example, we have no way of calculating what the "total" of all records is - because, we do not have all the records loaded. So, we have to provide this information separately. This example does not cover this scenario, but we demonstrate this in the Fully Lazy Loaded example in the demo console.

In the following example we are going to look at the intricacies involved in this use case.

You can find the example running below here : <http://www.sqledt.com/DotNetSample>

The source code for this example can be downloaded from : <http://www.htmltreegrid.com/demo/dotnetsample.zip>

Lets review the sample code that demonstrates this:

```
<div class="row">
  <div ng-app="app">
    <div ng-controller="myCtrl">
      <div id="gridContainer" fd-grid="" ng-model="gridOptions" style="height:
```

```

300px;width: 100%;"
    xicreation-complete="onGridCreationComplete"
    xienable-export="true"
    xienable-copy="true"
    xiforce-pagerrow="true"
    xipage-size="20"
    xienable-multi-column-sort="true"
    xienable-filters="true"
    xino-data-message=""
    xihorizontal-scroll-policy="auto"
    xienable-footers="false">

    <level xienable-paging="true" xipage-size="25" xienable-
filters="true" xiselectd-key-field="employeeId"
    xienable-footers="false"
    xireuse-previous-level-columns="true" xifilter-page-sort-
change="filterPageSortChangeHandler" xifilter-page-sort-mode="server">
    <columns>
    <column xitype="checkbox"></column>
    @*<column xidata-field="employeeId" xiheader-text="id"
xiheader-align="middle" xitext-align="left" xifilter-control="TextInput" xifilter-
operation="Contains" xisort-case-insensitive="true"></column>*@

    <column xidata-field="firstName" xiheader-text="First Name"
xiheader-align="middle" xifilter-control="TextInput" xifilter-operation="Contains"></
column>

    <column xidata-field="lastName" xiheader-text="Last Name"
xiheader-align="middle" xifilter-control="TextInput" xifilter-operation="Contains"></
column>

    <column xidata-field="department.departmentId" xiheader-
text="Department" xiheader-align="middle" xifilter-control="MultiSelectComboBox"
xifilter-combobox-build-from-grid="false"
    xifilter-combobox-width="150" xilabel-
function="getDepartment"></column>
    @*<column xidata-field="departmentId" xiheader-text="department
id" xiheader-align="middle" xifilter-control="TextInput" xifilter-operation="Contains"></
column>*@

    <column xidata-field="phoneNumber" xiheader-text="Phone"
xiheader-align="middle" xifilter-control="TextInput" xifilter-operation="Contains"></
column>

    <column xidata-field="stateCode" xiheader-text="State" xiheader-
align="middle" xifilter-control="ComboBox" xifilter-combobox-build-from-grid="true"
xifilter-combobox-width="150"></column>
    <column xidata-field="hireDate" xiheader-text="Hired" xifilter-
control="DateComboBox"
    itemeditor="flexiciousNmsp.DatePicker" xicolumn-width-
mode="fitToContent"></column>

    <column xiheader-align="middle" xisort-numeric="true" xidata-
field="annualSalary" xiheader-text="Annual Salary"
    xifilter-control="NumericRangeBox" xilabel-
function="flexiciousNmsp.UIUtils.dataGridFormatCurrencyLabelFunction"></column>
    <column xidata-field="isActive.toString()" xiheader-text="Active"
    xifilter-control="TriStateCheckBox" xifilter-operation="Equals" xilabel-
function="getActive"></column>
    </columns>
    </level>
    </div>
    </div>
    </div>
</div>
@section css{
    <!--css imports-->
    <link rel="stylesheet" href="http://htmltreegrid.com/demo/flexicious/css/

```

```

flexicious.css" type="text/css" />
    <link rel="stylesheet"
        href="http://htmltreegrid.com/demo/external/css/adaptor/jquery/jquery-ui-
1.9.1.custom.min.css"
        type="text/css" />
    <!--End-->
}
@section scripts{
    <script src="//www.parsecdn.com/js/parse-1.3.0.min.js"></script>

    <script type="text/javascript"
        src="http://htmltreegrid.com/demo/external/js/adapters/jquery/jquery-
1.8.2.js"></script>
    <script type="text/javascript"
        src="http://htmltreegrid.com/demo/external/js/adapters/jquery/jquery-ui-
1.9.1.custom.min.js"></script>
    <script type="text/javascript"
        src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.maskedinput-1.3.js"></script>
    <script type="text/javascript"
        src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.watermarkinput.js"></script>
    <script type="text/javascript"
        src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.ui.menu.js"></script>
    <script type="text/javascript"
        src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.toaster.js"></script>
    <!--End-->
    <!--These are specific to htmltreegrid-->
    <script type="text/javascript" src="http://htmltreegrid.com/demo/minified-compiled-
jquery.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/examples/js/
Configuration.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/themes.js"></script>

    <!--AngularJs -->
    <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/
angularjs/1.2.16/angular.min.js"></script>
    <script type="text/javascript" src="http://htmltreegrid.com/demo/flexicious/js/
htmltreegrid-angular-directive.js"></script>

    <!--End-->

    <script>
        /*
        We are using the angular integration in this example, so setup a module.
        */
        angular.module('app', ['fdGrid'])
            .factory('localStorage', function () {
                return {};
            })
            .factory('baseRequest', function ($q, $http) {
                return function (api, param, verb) {
                    verb = verb || 'post';
                    var defer = $q.defer();
                    $http[verb]('/api/' + api, param)
                        .then(function (response, status) { defer.resolve(response,
status); },

```

```

        function (response, status) { defer.reject(response, status); }
    );
    return defer.promise;
}
})
/*
    In this grid, we have a couple of columns - Departments and States. These
    have lookup based filters. Since at any time, we only load the top level filter, we need
    to query the database
    for all possible values for this pickers.
    This is not a problem with filterPageSortMode=client, because we load up the
    entire dataset and run a distinct on it to figure out the values for the picker.
    However with server based filterPageSortMode, we need to query the server to
    get the entire list of possible values to pick from.
*/
.service('lookUps', function (baseRequest) {
    return {
        departments: function () { //get all departments from server
            return baseRequest('Departments', null, 'get')
                .then(function (r) { return r.data; });
        },
        states: function () { //get all states from server
            return baseRequest('States', null, 'get')
                .then(function (r) { return r.data; });
        }
    }
})

/*
    This is the call for the actual data to render in the grid. This will be called
    in the following situations:
    1) On initial load - we will send an empty filter to get the first page of data
    (no filters, default sort)
    2) On page change : When the user changes the page via the paging buttons or the
    page dropdown
    3) On filter change : When the user changes the value of any of the filter
    controls within the columns.
    4) On sort change : When the user clicks on the column header for any of the
    columns.
    5) Programatically : If we call grid.processFilter()
    All cases above internally will dispatch a filterPageSortChange event, below you
    will see that in scope.filterPageSortChangeHandler we basically call this method.
*/
.factory('query', function (baseRequest) {
    return function (f) {
        var args = f.filterExpressions;
        if (args) {
            for (var i in args) {
                delete args[i].filterControl; //we dont want to send a
                //UIComponent to the server - not needed (and cannot serialize!)
            }
        }
        var filter = { //just send the entire filter object for the server to
            //convert to SQL Statement.
            filterDescription: f.filterDescription,
            arguments: f.filterExpressions || [],
            pageIndex: f.pageIndex || 0,
            pageSize: f.pageSize || 25,
            sorts: f.sorts
        };

        return baseRequest('Employee', filter)
            .then(function (r) { return r.data; });
    }
}

```

```

    })
    .controller('myCtrl1', function ($scope, query, lookUps) {

        $scope.gridOptions = {
            dataProvider: [],
            delegate: $scope
        };

        $scope.getDepartment = function (item) {
            return item && item.department ? item.department : ""; //
labelFunction
        }
        $scope.getActive = function (item) {
            return item.isActive ? 'Yes' : 'No'; //labelFunction
        }
        /*
        The filterPageSortChange Event: You have to wire up the
        "filterPageSortChange" event. This is the event that get dispatched when the user user
        clicks on the sort header on any of the columns, or request a change using either the
        page navigation buttons or the page navigation drop down in the toolbar, or runs a filter
        with in any of the columns. This event has 2 properties that are of interest:
            event.filter: This object contains all the information that you would
            potentially need to construct a SQL statement on the backend. Full documentation on this
            object can be found here:
            http://www.flexicious.com/resources/docs29/com/flexicious/grids/filters/
            Filter.html
            event.cause - This can be one of the three values:
            public static const FILTER_CHANGE:String = filterChange
            public static const PAGE_CHANGE:String = pageChange
            public static const SORT_CHANGE:String = sortChange
            */
        $scope.filterPageSortChangeHandler = function (evt1) {
            var grid = evt1.target;

            query(evt1.filter).then(function (data) {
                grid.setPreservePager(true);
                grid.setDataProvider(data.records);
                grid.setTotalRecords(data.totalRecords);
                grid.validateNow();
            });
        }
        $scope.onGridCreationComplete = function (event) {

            var grid = event.target;
            // These have lookup based filters. Since at any time, we only load
            the top level filter, we need to query the database for all possible values for this
            pickers.

            // This is not a problem with filterPageSortMode=client, because we
            load up the entire dataset and run a distinct on it to figure out the values for the
            picker.

            // However with server based filterPageSortMode, we need to query the
            server to get the entire list of possible values to pick from.
            //filterComboBoxDataProvider where
            filterControl="MultiSelectComboBox" or "ComboBox":
            lookUps.departments().then(function (data) {
                var filteredArray = flexiciousNmisp.UIUtils.filterArray(data,
                grid.createFilter(), grid, grid.getColumnLevel(), false);
                var stateCol =
                grid.getColumnByDataField("department.departmentId");
                stateCol.filterComboBoxLabelField = 'departmentName';
                stateCol.filterComboBoxDataField = 'departmentId';
                stateCol.filterComboBoxDataProvider = filteredArray;
                grid.rebuildFilter();
            });
        }
    });

```



```

    });
    ///filterComboBoxDataProvider where
filterControl="MultiSelectComboBox" or "ComboBox":
    lookUps.states().then(function (data) {
        var data2 = [];
        data.forEach(function (a) {
            var t = { label: a, data: a }
            data2.push(t);
        });

        var filteredArray = flexiciousNmsp.UIUtils.filterArray(data2,
grid.createFilter(), grid, grid.getColumnLevel(), false);
        var stateCol = grid.getColumnByDataField("stateCode");
        stateCol.filterComboBoxDataProvider = filteredArray;
        grid.rebuildFilter();
    });
    //Initial data load with a blank filter.
    query({ pageSize: grid.getPageSize() }).then(function (data) {
        grid.setPreservePager(true);
        grid.setDataProvider(data.records);
        grid.setTotalRecords(data.totalRecords);
        grid.validateNow();
    });

});

    })
</script>
}

```

## Hierarchical Data

In the previous section we saw the power of the datagrid with lazy loaded flat data. That example demonstrated lazily loading flat data. In terms of hierarchical data, this means it demonstrated lazy loading data at the "top level". As you may have guessed, we support this same functionality at the inner (children) levels. Before we get too far ahead, you may be better served by understanding the concept of "level"

Below is the documentation of the class FlexDataGridColumnLevel:

A class that contains information about a nest level of grid. This includes the columns at this level, information about whether or not to enable paging, footers, filters, the row sizes of each, the property of the dataprovider to be used as the key for selection, the property of the data provider to be used as the children field, the renderers for each of the cells, etc. The Grid always contains at least one level. This is the top level, and is accessible via the columnLevel property of the grid.

One of the most important concepts behind the Architecture of Flexicious Ultimate arose from the fundamental requirement that the product was created for - that is display of Heterogeneous Hierarchical Data.

The notion of nested levels is baked in to the grid via the "columnLevel" property. This is a property of type "FlexDataGridColumnLevel". The grid always has at least one column level. This is also referred to as the top level, or the root level. In flat grids (non hierarchical), this is the only level. But in nested grids, you could have any number of nested levels. This can be demonstrated in the following image.

The screenshot displays a hierarchical tree grid. On the left, callouts identify 'Level 1' through 'Level 4' and a 'Pager bar at any level'. The grid shows a tree of data with columns: ID, Legal Name, Address Line 1, Address Line, Deal Description, Invoice Number, Invoice Amount, and Deal Amount. The tree structure is as follows:

- Level 1: 20813 Allegheny Technologies (508 Park Lane, Suite #293)
- Level 2: Project # 1 - Allegheny Technologies - 10/2010
- Level 3: Invoice Number 2081300 (Invoice Amount: 165,754.00)
- Level 4: Line Item Description
  - Professional Services - Jason Bourne
  - Professional Services - Betty White
  - Professional Services - Jason Bourne
  - Professional Services - Jason Bourne
  - Professional Services - Lars Wilson

Summary rows show 'Count:5' for the current level and 'Total:\$679,376.00' for the entire tree. The pager bar at the bottom indicates 'Items 1 to 3 of 5. Page 1 of 2' and 'Go to Page: 1'.

The columns collection actually belongs to the columnLevel, and since there is one root level, the columns collection of the grid basically points to the columns collection of this root level. The FlexDataGridColumnLevel class has a "nextLevel" property, which is a pointer to another instance of the same class, or a "nextLevelRenderer" property, which is a reference to a ClassFactory the next level. Please note, currently, if you specify nextLevelRenderer, the nextLevel is ignored. This means, at the same level, you cannot have both a nested subgrid as well as a level renderer. Bottom line - use nextLevelRenderer only at the innermost level. Our examples demonstrate this.

Another thing to note is that there are two modes in which inner levels work with columns. Either same set of columns shared across each level, or each level has its own set of columns (We call them nested grids vs grouped grids). More information here : <http://htmltreegrid.com/newdocs/html/Flexicious%20HTMLTreeGrid.html?AdvancedConfigurationOptionsHier.html>

In markup, these would be defined as such:

```
=<grid id="grid" ...>'+
  <level ...>'+
  <columns>'+
  <column type="checkbox" />'+
  ...
  <column enableCellClickRowSelect="false" width="2000"
excludeFromSettings="true" excludeFromExport="true" excludeFromPrint="true" />'+
  </columns>'+
  <nextLevel>'+
  <level ...>'+
  <columns>'+
  <column type="checkbox" />'+
  ...
```

```

'
                                <column
itemEditor="flexiciousNmsp.DatePicker" editable="true" editorDataField="selectedDate"
dataField="dealDate" headerText="Deal Date"
labelFunction="flexiciousNmsp.UIUtils.dataGridFormatDateLabelFunction"/>' +
                                </columns>' +
                                <nextLevel>' +
                                <level ...>' +
                                <columns>' +
                                <column type="checkbox" /
>' +
                                ...
                                <column editable="true"
dataField="lineItemAmount" headerText="Line Item Amount" textAlign="right"
footerLabelFunction2="myCompanyNameSpace.fullyLazyLoaded_getFooterLabel"
footerAlign="right"
labelFunction="flexiciousNmsp.UIUtils.dataGridFormatCurrencyLabelFunction"/>' +
                                </columns>' +
                                </level>' +
                                </nextLevel>' +
                                </level>' +
                                </nextLevel>' +
                                </level>' +
                                </nextLevel>' +
                                </level>' +
                                </grid>;

```

(Full markup here : <http://www.htmltreegrid.com/demo/examples/js/samples/Nested.js>)

OR

```

'<grid id="grid" ...>' +
'
    <level ... >' +
    <columns>' +
    <column type="checkbox" />'
    ....
    <column itemEditor="flexiciousNmsp.DatePicker" ' +
    dataField="dueDate" headerText="Due Date" filterControl="DateComboBox"'+
    labelFunction="flexiciousNmsp.UIUtils.dataGridFormatDateLabelFunction"/>' +
    </columns>' +
    <nextLevel>' +
    <level ... reusePreviousLevelColumns="true" >' +
    ' +
    <nextLevel>' +
    <level ...
reusePreviousLevelColumns="true">' +
    ' +
    </level>' +
    </nextLevel>' +
    </level>' +
    </nextLevel>' +
    </level>' +
    </grid>;

```

(Full markup here : <http://www.htmltreegrid.com/demo/examples/js/samples/GroupedData.js>)

Now, lets look at what a simple configuration with hierarchical data looks like:

Hierarchical data works very similar to flat data, with the exception that the JavaScript object at the top level have a property usually named children that is a pointer to an array of nested objects which represent the children of the top-level objects. These nested objects in turn can have children of their own which in turn can have children of their own up to any number of nested levels. The one thing to keep in mind here is that the name of the property that points to the next level children is configurable via the childrenField property. This defaults to the string "children" but can be anything as long as it's configured appropriately

Another thing to notice about this example is the use of the enableDynamicLevels flag:

For those of you who are not familiar with what a dynamic tree grid is, (Actually the term is dynamicLevels) - This means that the grid will introspect the data provider to automatically figure out how deep the tree will nest. This is in contrast to other configurations where you explicitly define how "deep" the tree will be, what columns will be at each level, etc. But in case the hierarchy is unknown at design time, the grid is capable of introspecting the data provider and automatically generating the levels at run time. You do this by setting enableDynamicLevels="true" on the grid. However, since the levels are not defined at design time, to be able to manipulate their properties at runtime, we have an event, DYNAMIC\_LEVEL\_CREATED and DYNAMIC\_ALL\_LEVELS\_CREATED. Both these events are defined on the FlexDataGridEvent class.

```
<script type="text/javascript">
```

```
    $(document).ready(function () {
        var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
        {
            configuration: '<grid id="grid" enableDynamicLevels="true"
variableRowHeight="true" horizontalScrollPolicy="on" recalculateSeedOnEachScroll="true"
enableExport="true" forcePagerRow="true" pageSize="50" enableFilters="true"
enableFooters="true" >' +
                '
                    <level enableFooters="true" ' +
                '     childrenField="children">' +
                '
                    <columns>' +
                '
                        <column
enableHierarchicalNestIndent="true" dataField="id" headerText="ID" width="100"/>' +
                '
                        <column dataField="type"
headerText="Type" width="100" wordWrap="true"/>' +
                '
                        <column dataField="type"
headerText="Type" />' +
                '
                    </columns>' +
                '
                    </level>' +
                '
            </grid>',
            dataProvider: [
                { "id": "5001", "type": "None None None None None None None None
None None None None None " , children: [
                    { "id": "5001", "type": "None 1 None 1 None 1 None 1 None 1
None 1 None 1 None 1 None 1" },
                    { "id": "5002", "type": "Glazed 1 Glazed 1 Glazed 1 Glazed 1
Glazed 1 Glazed 1 Glazed 1 Glazed 1 Glazed 1 Glazed 1 Glazed 1" },
                    { "id": "5005", "type": "Sugar 1 Sugar 1 Sugar 1 Sugar 1 Sugar
1 Sugar 1 Sugar 1 Sugar 1 Sugar 1" },
                    { "id": "5007", "type": "Powdered Sugar 1 Powdered Sugar 1
Powdered Sugar 1 Powdered Sugar 1 Powdered Sugar 1" },
                    { "id": "5006", "type": "Chocolate with Sprinkles 1 Chocolate
with Sprinkles 1 Chocolate with Sprinkles 1 Chocolate with Sprinkles 1" },
                    { "id": "5003", "type": "Chocolate 1 Chocolate 1 Chocolate 1
```

```

Chocolate 1 Chocolate 1 Chocolate 1 Chocolate 1 Chocolate 1 Chocolate 1 " },
        { "id":"5004", "type":"Maple 1 Maple 1 Maple 1 Maple 1
Maple 1 Maple 1 Maple 1 Maple 1 Maple 1 Maple 1 " }
    ]
},
{ "id":"5002", "type":"Glazed" },
{ "id":"5005", "type":"Sugar" },
{ "id":"5007", "type":"Powdered Sugar" },
{ "id":"5006", "type":"Chocolate with Sprinkles" },
{ "id":"5003", "type":"Chocolate" },
{ "id":"5004", "type":"Maple" }
    ]
});

grid.expandAll();
grid.validateNow();
});
</script>

```

In this example we can see that the top level object with the ID of 5001 has a list of children.

## Hierarchical Data - Lazy Load - Java and PHP Sample

In the previous section we looked at the structure of a hierarchical treegrid - we talked about the concept of levels, nested vs grouped and dynamic levels. Now, lets talk about another complex topic, lazy load. In this example in addition to the top level lazy load we talked about in the previous example (using `filterPageSortMode`), we introduce a concept of `itemLoadMode`.

There are also two different modes of loading hierarchical data.

`itemLoadMode=client` (default) - This assumes the parent objects and child objects are all loaded in client memory upfront.

`itemLoadMode=server` - This assumes only the top level items are loaded, and the grid will trigger an event that you will then listen for, and load children in a lazy load mechanism (or load on demand). This is more appropriate when there are very large datasets.

When `itemLoadMode` is server, you may want to set `childrenCountField`.

A property on the object that identifies if the object has children. Only needed in `itemLoadMode=server` In lazy loaded hierarchical grids levels, the child level items are loaded when the user actually expands this level for the first time. In scenarios where it is known initially that there are children, set this property to the value of the object that identifies whether there are children. If this property is set, the expand collapse icon will be drawn only when the value of this property on the object returns an integer greater than zero. Otherwise, the level will always draw the expand collapse icon.

To summarize, In client mode, the grid will assume that the children of items at this level are pre-fetched. In server mode, the grid will dispatch a `ITEM_LOAD` event (`itemLoad`) that should be used to construct an appropriate query to be sent to the back-end, to retrieve the child objects at this level. Once the results are retrieved, please call the `"setChildData"` method on the grid to set the results at this level. Please note, the `"childrenField"` is still required at this level, because that is where the `setChildData` method persists the loaded children. Future `itemOpen` events do not result in `itemLoads` because the data for this particular entity has already been loaded and persisted.

So, lets take a quick look at what this example looks like:

**1) Java Version :** <http://flexicious.com:8400/HtmlTreeGridSpring/>

**2) PHP Version :** <http://flexicious.com:8081/php-sql-demo-app>

The source code for this example can be downloaded from :

**1) Java:** <http://www.htmltreegrid.com/demo/javasample.zip>

**2) PHP:** <http://www.htmltreegrid.com/demo/phpsample.zip>

Below is the code for this example (client side only) - the server side code is included in the above zip file for you to inspect:

```
/**
 * This Example demonstrates the next level of lazy load capabilities of the
 * Flexicious DataGrid, that is each level of detail, as well as paging at each
 * level in a lazy loaded configuration.
 * There are two properties to pay attention to here, both of which are
 * defined at the column level:
 * FilterPageSortMode : The Filter/Page/Sort Mode. Can be either "server" or
 * "client". In client mode, the grid will take care of paging, sorting and
 * filtering once the dataproducer is set. In server mode, the grid will fire a
 * filterPageSortChange event that should be used to construct an appropriate
 * query to be sent to the backend.
 * ItemLoadMode : The Item Load Mode. Can be either "server" or "client". In
 * client mode, the grid will assume that the children of items at this level are
 * prefetched. In server mode, the grid will dispatch a ITEM_LOAD event
 * (itemLoad) that should be used to construct an appropriate query to be sent to
 * the backend, to retrieve the child objects at this level. Once the results
 * are retrieved, please call the "setChildData" method on the grid to set the
 * results at this level. Please note, the "childrenField" is still required at
 * this level, because that is where the setChildData method persists the loaded
 * children.
 * Future itemOpen events do not result in itemLoads because the data for this
 * particular entity has already been loaded and persisted.
 */
* In this example, we see how to wire up a partially lazy loaded Hierarchical
  DataGrid. That is, we load up the top level with no children records, and
  then lazy load them in as the user clicks on expand.
  * Please note, it is not advisable to set enableDrillDown on lazy loaded
  grids, because this will result in too many server calls being issued.
  */
var GridConFig = window.GridConFig = {
    //ApiCallBaseUrl : "http://localhost:63343/php-sql-demo-app/api/
    sever_records/", // - php call
    ApiCallBaseUrl : window.location+"api/server_records/", //- java call
    XmlConfig : {
        sampleGrid: '<grid ' +
            'height="100%" ' +
            'width="100%" ' +
            'enablePrint="true" ' +
            'enableExport="true" ' +
            'forcePagerRow="true" ' +
            'enableFilters="true" ' +
            'pagerRowHeight = "35" ' +
                'enablePreferencePersistence="true" ' +
            'rowHeight = "30" ' +
            'pageSize = "15" ' +
            'pageIndex = "1" ' +
            'horizontalScrollPolicy="auto" ' +
            'selectionColor="transparent" ' +
            'showSpinnerOnFilterPageSort="true" ' +
            'enableDrillDown = "true" ' +
            'nestIndent="36" ' +
            'filterPageSortMode="server" ' +
            'enableDefaultDisclosureIcon="false" ' +
```

```

        'enablePaging="true" ' +
        'pagerRenderer="flexiciousNmsp.CustomPagerRenderer" '
+
        'multiSortRenderer="flexiciousNmsp.CustomMultiColumnSo
rtPopupRenderer" ' +
        'filterPageSortChange="filerPageSortHandle" ' +
        'enableMultiColumnSort="true" ' +
        'enableColumnHeaderOperation="true" ' +
        'selectionMode="multipleRows"> ' +
        '<level name="Top Level" headerHeight="30"
childrenField="children" itemLoad="itemLoadHandler" itemLoadMode="server"
childrenCountField="childCounts">' +
            '<columns>' +
                '<column dataField="record_type"
width="200" headerText="Record Type" ' +
                    'enableHierarchicalNestIndent="tru
e" paddingLeft="25" enableExpandCollapseIcon="true"
filterControl="MultiSelectComboBox" filterOperation="Contains"
filterTriggerEvent="enterKeyUpOrFocusOut" filterComboBoxDataProvider=
"eval__getFilterComboBoxDP_RecordType()" />' +
                    '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="record" width="350" headerText="Record"/>' +
                    '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="site" width="100" headerText="Site"/>' +
                    '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="system" width="100" headerText="System"/>' +
                    '<column filterControl="DateComboBox"
dataField="start_time" width="350" headerText="Start Time"
labelFunction="startTimeLabelFunction"/>' +
                    '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="run_time" width="100" headerText="Run Time"
labelFunction="runTimeLabelFunction"/>' +
                    '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="status" headerRenderer="CustomHeaderRender" width="100"
headerText="status" labelFunction="statusLabelFunction"/>' +
                    '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="jenkins" width="100" headerText="jenkins" />' +
                    '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="result" width="100" headerText="result"/>' +
                        '<column
filterControl="DateComboBox" dataField="start_time" width="350"
headerText="Start Time" labelFunction="startTimeLabelFunction"/>' +
                        '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="run_time" width="100" headerText="Run Time"
labelFunction="runTimeLabelFunction"/>' +
                        '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="status" headerRenderer="CustomHeaderRender" width="100"
headerText="status" labelFunction="statusLabelFunction"/>' +
                        '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="jenkins" width="100" headerText="jenkins" />' +

```

```

        '<column filterControl="TextInput"
filterOperation="Contains" filterTriggerEvent="enterKeyUpOrFocusOut"
dataField="result" width="100" headerText="result"/>' +
        '</columns>' +
        '<nextLevel>' +
            '<level name="Second Level"
nestIndent="36" headerHeight="35" reusePreviousLevelColumns="true"
rowHeight="35" childrenField="children" filterVisible="false" />' +
            '</nextLevel>' +
            '</level>' +
        '</grid>'
    }
};
/**
// grid callbacks
//These have lookup based filters. Since at any time, we only load the top
level filter, we need to query the database for all possible values for this
pickers.
//This is not a problem with filterPageSortMode=client, because we load up the
entire dataset and run a distinct on it to figure out the values for the
picker.
//However with server based filterPageSortMode, we need to query the server to
get the entire list of possible values to pick from. In this case we are just
hardcoding the list
//look at the Dot.Net example to show how to load it from server and wire up
in the return call.
*/
function getFilterComboBoxDP_RecordType() {
    return [
        {label:'Batch',data:'batch'},
        {label:'Profile',data:'profile'},
        {label:'Testcase',data:'testcase'}
    ];
}

/**
 * The item load handler receives a filterPageSortChangeEvent, which contains
a parentObject property that represents the item being opened.
 * We basically issue a server request for the children of that time, and in
the result event, call the setChildData method passing in the parent item, the
children,
 * the level at which the parent item exists, and the total number of records
that we have, if it is different than the length of the children collection
(i.e.) if we are just pushing down a single page of data.
 * This handler basically calls out to the services layer, gets the data, and
calls the setChildData method on the grid on result.
**/
var itemLoadHandler = function(event) {
    var parentData = event.filter.parentObject;
    $.ajax({
        url : GridConFig.ApiCallBaseUrl,
        data : {name : "child_data", parent_id : parentData.id},
        type : "GET",
        success : function(res) {
            var response = JSON.parse(res);
            if(response.success) {
                var grid = event.target;
                grid.setChildData(parentData, response.data,
event.filter.level.getParentLevel());
            } else {

```



```

        alert(response.message);
    }
}
});
};
/**
The filterPageSortChangeEvent: You have to wire up the "filterPageSortChangeEvent"
event. This is the event that get dispatched when the user user clicks on the
sort header on any of the columns, or request a change using either the
page navigation buttons or the page navigation drop down in the toolbar, or
runs a filter with in any of the columns. This event has 2 properties that are
of interest:
event.filter: This object contains all the information that you would
potentially need to construct a SQL statement on the backend. Full
documentation on this object can be found here:
http://www.flexicious.com/resources/docs29/com/flexicious/grids/filters/Filter.html
event.cause - This can be one of the three values:
public static const FILTER_CHANGE:String = filterChange
public static const PAGE_CHANGE:String = pageChange
public static const SORT_CHANGE:String = sortChange
**/
var filterPageSortHandle = function(event){
    setTimeout(function () {
        var filterPageSort = {};
        var grid = event.target;
        if(event.cause == "pageChange"){
            filterPageSort.pageIndex =
event.triggerEvent.currentTarget._pageIndex;
        } else{
            filterPageSort.pageIndex = grid.getColumnLevel()._pageIndex;
        }
        filterPageSort.pageSize = grid.getPageSize();
        var sorts = event.target.getCurrentSorts();
        if(sorts && sorts.length){
            filterPageSort.sorts = [];
            for(var i = 0; i < sorts.length; i++){
                filterPageSort.sorts.push({
                    sortColumn : sorts[i].sortColumn,
                    isAscending : sorts[i].isAscending,
                    sortNumeric : sorts[i].sortNumeric
                });
            }
        }
        var filter = event.target.getRootFilter();
        if(filter.filterExpressions && filter.filterExpressions.length){
            filterPageSort.filters = [];
            for(var i = 0; i < filter.filterExpressions.length; i++){
                filterPageSort.filters.push({
                    columnName : filter.filterExpressions[i].columnName,
                    expression : filter.filterExpressions[i].expression,
                    filterOperation :
filter.filterExpressions[i].filterOperation,
                    filterComparisonType :
filter.filterExpressions[i].filterComparisonType
                });
            }
        }

$.ajax({

```

```

        url : GridConFig.ApiCallBaseUrl,
        data : {name : "top_data", filterPageSort :
JSON.stringify(filterPageSort)},
        type : "GET",
        success : function(res){
            if(res.trimLeft().indexOf("<") == 0){
                grid.hideSpinner();
                alert("Error occur while loading the data.");
                return;
            }
            var response = JSON.parse(res);
            if(response.success){
                grid.setDataProvider(response.data);
                if(event.cause == "filterChange")
                    grid.setTotalRecords(response.details.totalRecords);
            }else{
                alert(data);
            }
        }
    });
},1);
};

var runTimeLabelFunction = function(data, col){
    var totalSec = Math.round(data["run_time"]/1000);
    var hours = parseInt( totalSec / 3600 ) % 24;
    var minutes = parseInt( totalSec / 60 ) % 60;
    var seconds = totalSec % 60;

    return (hours < 10 ? "0" + hours : hours) + ":" + (minutes < 10 ? "0" +
minutes : minutes) + ":" + (seconds < 10 ? "0" + seconds : seconds);
};

var startTimeLabelFunction = function(data, col){
    var start_date = data["start_time"];
    return new Date(Date.parse(start_date)).toString();
};

var statusLabelFunction=function(data, col){
    var status = data["status"];
    if(status.toLowerCase() == "stopped")
        return "<span style='color: #ff4545; font-weight: bold'>Stopped</span>";
    else if(status.toLowerCase() == "running")
        return "<span style='color: #3434FF; font-weight: bold'>Running</span>";
    else if(status.toLowerCase() == "passed")
        return "<span style='color: #458F00; font-weight: bold'>Passed</span>";
    else if(status.toLowerCase() == "failed")
        return "<span style='color: #FF0000; font-weight: bold'>Failed</span>";
    return "";
};

```

On the server side, the key class to note is the `FilterBuilder` class

This class is responsible for taking the Filter object and build the filter

from it.

## Beyond the Basics

---

In this section, we will go over some advanced concepts, and some architectural aspects that will help you better understand how the grid works equip you to program against its API.

### OO Concepts for JavaScript Developers

This section is primarily geared towards those of our customers who are moving from our other technologies (Flex, Android, or iOS) to the JavaScript version of our product. If you are only using our HTML product, this section may not be of too much value to you, but it should be an interesting read nevertheless.

As most of you are already aware, JavaScript does not have first class OO support that ActionScript does. However, it does have an extremely versatile prototype based architecture that can be used to mimic most OO concepts. Indeed, there are numerous implementations of OO concepts in JavaScript. The goal of this section is not to go over the various mechanisms that let you emulate OO programming in JavaScript. There are numerous resources on the web that do this. Rather, in this section we will visit a few important concepts that you need to be aware of when you program against the HTMLTreeGrid API, mostly in scenarios where you need to write custom itemRenderers, itemEditors, or you are extending the core classes from the framework.

When implementing HTMLTreeGrid, we have created a small OO implementation that closely mimics ActionScript. In this section, we will cover two main concepts.

- 1) The Flexicious Typed Object class (`flexiciousNmsp.TypedObject`) : This object is the base class of all classes in the HTMLTreeGrid library. This class provides basic OO support, including Class Names, Interfaces, Type checking or interface implementations via the "implementsOrExtends" function (equivalent to "is" keyword from ActionScript), and other OO constructs like prototype chaining and polymorphism.
- 2) The Flexicious UIComponent class (`flexiciousNmsp.UIComponent`) : This class, inherits from `flexiciousNmsp.TypedObject` and is primarily responsible for encapsulating a HTML domElement, manipulating it in an API that is compatible with a number of Flash based API calls. In other words, we ported a limited sub section of the Flex SDK's UIComponent and DisplayObject class so we could implement Event Dispatching, Validation/Invalidation, Sizing and positioning similar to the analogous counterparts from Flex, so our HTMLTreeGrid code could be ported with ease. This should assist you as you port over your itemRenderers, editors, class factories and more. The other key aspect of this class is that it extends the `flexiciousNmsp.EventDispatcher` class. This affords it the ability to broadcast and listen for events.

First, lets take a quick look at the `flexiciousNmsp.TypedObject` class.

#### **flexiciousNmsp.TypedObject Class**

This class is the base class for all Flexicious classes. It defines the basis for the Object oriented design, by implementing the "implementsOrExtends" function, which is a replacement for the "is" keyword from other OO languages.

**Methods**`getClassNames`**Array**

Returns a list of strings that represent the object hierarchy for this object.

Returns:

Array:

`implementsOrExtends`

(

- name

)

Boolean

Returns true if the class name to check is in the list of class names defined for this class.

Parameters:

- name Object  
Name of the class to check

Returns:

Boolean:

As is obvious, it is very simple to use this class. Below is a sample class that “inherits” from `flexiciousNmsp.TypedObject`. Pay close attention to the code highlighted in yellow:

```

/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function(window)
{
    "use strict";
    var PreferenceInfo, uiUtil = flexiciousNmsp.UIUtils, flxConstants =
flexiciousNmsp.Constants;
    /**
     * Class added in 2.9 to support multiple preferences. Store information about the
     * name of the preference, whether or not it is a system preference (which cannot be
deleted),
     * and the preference string associated with this name.
     * @constructor
     * @class PreferenceInfo
     * @namespace flexiciousNmsp
     * @extends TypedObject
     */
    PreferenceInfo=function(){
        /**
         * Name of the preference
         * @type {String}
         * @property name
         * @default null
         */
        this.name=null;
        /**

```

```

* These cannot be deleted
* @type {Boolean}
* @property isSystemPref
* @default false
*/
this.isSystemPref=false;
/**
* The actual text (or xml) of preferences.
* @type {String}
* @property preferences
* @default null
*/
this.preferences=null;

```

**flexiciousNmsp.TypedObject.apply(this);** //this ensures that the super constructor gets called.

```

};
/**
*
* @type {Function}
*/
flexiciousNmsp.PreferenceInfo = PreferenceInfo; //add to name space
PreferenceInfo.prototype = new flexiciousNmsp.TypedObject(); //this ensures that the
prototype hierarchy is setup for inheritance.
PreferenceInfo.prototype.typeName = PreferenceInfo.typeName = 'PreferenceInfo';//for
quick inspection
/**
*
* @return {Array}
*/
PreferenceInfo.prototype.getClassNames=function(){
return ["PreferenceInfo","TypedObject"];
};//This ensures that when you call
obj.implementsOrExtends('SomeInterfaceOrClassName'), you can get the correct
result.
}(window));

```

## flexiciousNmsp.UIComponent Class

As we looked at in the introduction, this class is important because all of the Flexicious UI classes inherit from this class. You will mostly need to work with this class if you have highly interactive item renderers, that cannot be implemented using other simpler mechanisms like labelFunctions and interactive HTML (covered in later chapters). Let's take a sample itemRenderer that inherits from UIComponent:

For those of you who are not aware, and itemRenderer is a classfactory that instantiates a component that encapsulates the DOM element that sits inside each data cell. (There are analogous header,footer,filter,pager and nextLevelRenderers as well)

```

    /**
    * Flexicious
    * Copyright 2011, Flexicious LLC
    */
    (function (window) {
        "use strict";
        var CheckBoxRenderer, uiUtil =
flexiciousNmsp.UIUtils, flxConstants = flexiciousNmsp.Constants;
        /**
        * A CheckBoxRenderer is a custom item renderer,
that defines how to use custom cells with logic that you can control
        * @constructor
        * @namespace flexiciousNmsp
        * @extends UIComponent
        */
        CheckBoxRenderer = function () {
            //make sure to call constructor
            flexiciousNmsp.UIComponent.apply(this,
["input"]); //second parameter is the tag name for the dom element.
            this.domElement.type = "checkbox"; //so our
input element becomes a checkbox;

            /**
            * This is a getter/setter for the data
property. When the cell is created, it belongs to a row
            * The data property points to the item in the
grids dataprovider that is being rendered by this cell.
            * @type {*}
            */
            this.data = null;

            //the add evt listener will basically proxy
all DomEvents to your code to handle.
            this.addEventListener(this,
flxConstants.EVENT_CHANGE, this.onChange);

```

```

    };
    myCompanyNameSpace.ItemRenderers_CheckBoxRenderer
= CheckBoxRenderer; //add to name space
    CheckBoxRenderer.prototype = new
flexiciousNmsp.UIComponent(); //setup hierarchy
    CheckBoxRenderer.prototype.typeName =
CheckBoxRenderer.typeName = 'CheckBoxRenderer'; //for quick
inspection
    CheckBoxRenderer.prototype.getClassName =
function () {
        return ["CheckBoxRenderer", "UIComponent" ,
"EventDispatcher", "TypedObject"]; //this is a mechanism to replicate
the "is" and "as" keywords of most other OO programming languages
    };

    /**
    * This is important, because the grid looks for a
"setData" method on the renderer.
    * In here, we intercept the call to setData, and
inject our logic to populate the text input.
    * @param val
    */
    CheckBoxRenderer.prototype.setData = function
(val) {
        flexiciousNmsp.UIComponent.prototype.setData.
apply(this, [val]);
        var cell = this.parent; //this is an instance
of FlexDataGridDataCell (For data rows)
        var column = cell.getColumn(); //this is an
instance of FlexDataGridColumn.
        this.domElement.checked =
this.data[column.getDataField()];
    };
    /**
    * This event is dispatched when the user clicks
on the icon. The event is actually a flexicious event, and has a
trigger event
    * property that points back to the original
domEvent.
    * @param evt
    */
    CheckBoxRenderer.prototype.onChange = function
(evt) {

        //in the renderer, you have the handle to the
cell that the renderer belongs to, via the this.parent property that
you inherit from flexiciousNmsp.UIComponent.

        var cell = this.parent; //this is an instance
of FlexDataGridDataCell (For data rows)
        var column = cell.getColumn(); //this is an

```

instance of FlexDataGridColumn.

```

        this.data[column.getDataField()] =
this.domElement.checked; //we use the dom element to wire back the
value to the data object.
    };
    //This sets the inner html, and grid will try to
set it. Since we are an input field, IE 8 will complain. So we ignore
it since we dont need it anyway.
    CheckBoxRenderer.prototype.setText = function
(val) {

        };
    } (window));

```

## Configuration - XML vs API

There are two distinct mechanisms you can use to build the grid. XML, and API. The XML is easier, less verbose and can be used to cleanly separate the markup of the grid from the logic associated with it (like event handlers and function callbacks we discuss elsewhere in this guide).

The key to keep in mind however, is that the XML is ultimately passed into helper functions which then call the API methods and build out the grid. So the API is used in both cases, the XML is just a shorthand preprocessor for the API that makes it easier to configure the grid.

## XML Configuration

When you define a grid, you pass in XML configuration for the grid. We have seen many examples of this so far, but lets look at a simple example here:

```

$(document).ready(function () {
    var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
    {

        configuration: '<grid id="grid" enablePrint="true"
enablePreferencePersistence="true" enableExport="true" forcePagerRow="true" pageSize="50"
enableFilters="true" enableFooters="true" enablePaging="true">' +
            '<level>' +
                '<columns>' +
                    '<column type="checkbox"/>' +
                        '<column
dataField="id" headerText="ID" filterControl="DynamicFilterControl" footerLabel="Sum: "
footerOperation="sum" footerOperationPrecision="2"/>' +
                            '<column
dataField="type" headerText="Type" filterControl="TextInput" filterOperation="Contains" /
>' +
                                '</columns>' +
                            '</level>' +
                        '</grid>',
        dataProvider: [
            { "id": "5001", "type": "None", "active":
true },

```



```

"active": true },
"active": true },
"active": false },
Sprinkles", "active": true },
"active": false },
"active": true }
    ]
});

});

```

As you see above, we basically pass in an XML string to the constructor of the FlexDataGrid class. One very frequent question comes up is, "What are the various attributes you can specify in XML"?

The answer is simple. All of them. Each and every property mentioned in the documentation can be specified via XML. This includes both properties explicitly exposed as public attributes (e.g. enablePrint, enableExport etc) , and some that are exposed via setters (e.g. setPageRowHeight, setPageVisible) . **The XML parser is smart enough to figure out if there is a setter with the naming convention setXXXX and use it instead of a direct value set on the property. (This is in red because it often causes confusion among our customers). So `pageSize="50"` automatically translates to `setPageSize(50)`, even if there is no property called `pageSize` on FlexDataGrid, but because there is a method named `setPageSize`. It is also smart enough to figure out that a value is a number passed in as a string, so it will do that conversion as well.**

In addition to the above, the XML builder does a lot of preprocessing of the string values passed into it. These steps are described below.

- Any attribute that matches the name of an event. The XML parser assumes that any string passed in as the value for an event is a function call back and treats it as such. For example `creationComplete="myCompanyNameSpace.onCreationComplete"`. So this needs to evaluate to a name spaced function. A full list of events is available here: <http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html#events>
- For any properties with the following suffix, we eval the passed in value. This means the value needs to evaluate to a namespace qualified function. e.g. `filterRenderer="flexiciousNmsp.DynamicFilterControl"`:
  - Function
  - Renderer
  - Editor
  - Formatter
  - DateRangeOptions
- For the following properties, we wrap the value in a ClassFactory:
  - filterRenderer
  - footerRenderer
  - headerRenderer
  - pagerRenderer
  - itemRenderer

6. nextLevelRenderer
7. itemEditor
4. If any value is passed in within curly braces {}, we assume that is a function call. Like {executeFunction(1,2)}. We will execute the executeFunction passing in 1 and 2, and return with the result. This is very rarely used.
5. For any value you prefix with the word eval\_\_, we evaluate it before it gets applied
6. For any value you specify within square brackets [], we assume it is an array, and split the string by commas and convert it to an array.
7. For any value that starts with 0x, we assume its a hex code and use it as such. This is used for colors.
8. For any value that is composed solely of numbers, we parse it as such.
9. Finally, after all this preprocessing is done, we check to see if there is a setter method specified for the property name at the target object level (FlexDataGrid, FlexDataGridColumnLevel, or FlexDataGridColumn). If so, we call the setter method with the preprocessed value. Else, we set a property with that name to the preprocessed value.
1. For the grid tag, you can specify any of the properties listed here : <http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html>
2. For the level tag, you can specify any of the properties listed here : <http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html>
3. For the column tag, you can specify any of the properties listed here : <http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumn.html>

## API Configuration

In the previous topic, we saw the XML method of configuring the grid. The API method is basically what the XML configuration method uses. Many of our customers store their grid configuration in the backend. They then use it to build out the grid. Some of them choose to build out XML from their backend and then call the buildFromXML method on the grid. Others build out the grid using the API directly. This gives you more flexibility, but makes the code a lot more verbose. Let's take a quick look at the code required to build out a grid with no XML.

```
myCompanyNameSpace.SAMPLE_CONFIGS_DETAIL_URLS["DynamicColumns"]="http://
www.flexicious.com/resources/Ultimate/docs/DynamicColumns.htm";
```

```
myCompanyNameSpace.DynamicColumns_grid_creationCompleteHandler=function(evt)
{
    var grid=this;
    grid.setDataProvider(myCompanyNameSpace.FlexiciousMockGenerator.instance().getFlattOrgList());
    grid.clearColumns();

    var col=myCompanyNameSpace.DynamicColumns_addColumn("id","Company ID");
    col.setColumnLockMode(flexiciousNmsp.FlexDataGridColumn.LOCK_MODE_LEFT)
    grid.addColumn(col);
    col=myCompanyNameSpace.DynamicColumns_addColumn("legalName","Company Name");
    col.setColumnLockMode(flexiciousNmsp.FlexDataGridColumn.LOCK_MODE_RIGHT)
    grid.addColumn(col);
    grid.addColumn(myCompanyNameSpace.DynamicColumns_addColumn("headquarterAddress.line1",
    "Address Line 1"));
    grid.addColumn(myCompanyNameSpace.DynamicColumns_addColumn("headquarterAddress.line2",
    "Address Line 2"));
```

```

        grid.addColumn(myCompanyNameSpace.DynamicColumns_addCurrencyColumn("earningsPerShare"
, "EPS"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addColumn("headquarterAddress.line1"
, "Address Line 1"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addColumn("headquarterAddress.line2"
, "Address Line2"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addCurrencyColumn("earningsPerShare"
, "EPS"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addColumn("headquarterAddress.line1"
, "Address Line 1"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addColumn("headquarterAddress.line2"
, "Address Line2"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addCurrencyColumn("earningsPerShare"
, "EPS"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addColumn("headquarterAddress.line1"
, "Address Line 1"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addColumn("headquarterAddress.line2"
, "Address Line2"));
        grid.addColumn(myCompanyNameSpace.DynamicColumns_addCurrencyColumn("earningsPerShare"
, "EPS"));
        //grid.distributeColumnWidthsEqually();
        grid.reDraw();
    };
myCompanyNameSpace.DynamicColumnsCounter=0;
myCompanyNameSpace.DynamicColumns_addCurrencyColumn=function(dataField,headerText){
    var dgCol = myCompanyNameSpace.DynamicColumns_addColumn(dataField,headerText);
    dgCol.setLabelFunction(flexiciousNmsp.UIUtils.dataGridFormatCurrencyLabelFunction);
    dgCol.setStyle("textAlign","right");
    dgCol.setUniqueIdentifier(headerText+myCompanyNameSpace.DynamicColumnsCounter++);
    dgCol.footerOperation="average";
    dgCol.footerLabel="Avg: ";
    dgCol.footerAlign="right";
    dgCol.setStyle("paddingRight",15);
    dgCol.filterOperation="GreaterThan";
    dgCol.filterWaterMark = "Greater Than";
    return dgCol;
};
myCompanyNameSpace.DynamicColumns_counter=0;
myCompanyNameSpace.DynamicColumns_addColumn=function(dataField,headerText){
    var dgCol = new flexiciousNmsp.FlexDataGridColumn();
    dgCol.setDataField(dataField);
    dgCol.setHeaderText(headerText);
    //because columns are having the same header text, we need to provide unique
identifiers.
    dgCol.setUniqueIdentifier(headerText+""+myCompanyNameSpace.DynamicColumns_counter++);
    dgCol.filterControl="TextInput";
    dgCol.filterOperation="BeginsWith";
    dgCol.filterWaterMark = "Begins With";
    return dgCol;
};

myCompanyNameSpace.SAMPLE_CONFIGS["DynamicColumns"]='<grid horizontalScrollPolicy="on"
id="grid" width="800" height="500" enablePrint="true" ' +
    'enablePreferencePersistence="true" generateColumns="false" ' +
    'enableExport="true"
enableCopy="true" enableFilters="true" enableFooters="true" enablePaging="true" ' +
    'preferencePersistenceKey="dynamicColumns"' +
on'+flexiciousNmsp.Constants.EVENT_CREATION_COMPLETE
+="'myCompanyNameSpace.DynamicColumns_grid_creationCompleteHandler">' +

```

```
'      '+  
'    </grid>';
```

So, in this example, we just use the XML to initialize the grid. We use API to generate the actual columns. For a running copy of this example, please refer to [http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=Dynamic%20Columns](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=Dynamic%20Columns)

We also have a more complicated grid, with column groups and inner levels. The code for this as well as a running example is available here:

[http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=Large%20Dynamic%20Grid](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=Large%20Dynamic%20Grid)

## Event Handlers and Function Callbacks

In the previous section we embeded interactive content within cells. By interactive content, we mean things like tooltips, programmatic interactive html, etc. In this section, we will explore how to wire up event handlers and function callbacks via the XML configuration for the grid. We have explored some of this over the past few sections, but they are rich enough to warrant their own topic in the guide.

### Events

The grid, at various points in its lifecycle dispatches events. These are quite similar to HTML Dom events, and if you are coming to HTMLTreegrid from our other products should be quite straight forward to you conceptually. Even if you are new to HTMLTreeGrid in theory, it is quite simple to grasp. Something of interest happens, an event is dispatched. Anyone interested in these events can listen for them. There are different points in the grids lifecycle that it will dispatch these events, from initialization, to dataProviderSet, to rendered, to mouse interactions like mouseover, mouse out, item click, edit begin, edit end, etc.. Let's take a look at a quick example. Pay close attention to the markup in yellow highlight.

```
<!doctype html>
<html lang="en" >
<head>
  <meta charset="utf-8">
  <title>Simple</title>
  <!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-1.8.2.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-ui-1.9.1.custom.min.js"></
script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.ui.menu.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.toaster.js"></script>
  <!--End-->
  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/minified-compiled-jquery.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/examples/js/Configuration.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/themes.js"></script>
  <!--End-->
  <!--css imports-->
```

```

<link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
flexicious/css/flexicious.css"
    type="text/css" />
<link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
external/css/adapters/jquery/jquery-ui-1.9.1.custom.min.css"
    type="text/css" />
<!--End-->
<script type="text/javascript">

    $(document).ready(function () {
        myCompanyNameSpace.onChange = function (evt) {
            var grid = evt.target;
            var selectedItems = grid.getSelectedObjects();
            alert("selected objects: " +
flexiciousNmisp.UIUtils.extractPropertyValues(selectedItems,"id"));
        }
        myCompanyNameSpace.onItemClick = function (evt) {
            var grid = evt.target;
            alert('You clicked on ' + evt.item.id);
        }
        var grid = new
flexiciousNmisp.FlexDataGrid(document.getElementById("gridContainer"),
        {
            configuration: '<grid id="grid"
itemClick="myCompanyNameSpace.onItemClick"
change="myCompanyNameSpace.onChange" enablePrint="true"
enablePreferencePersistence="true" enableExport="true"
forcePagerRow="true" pageSize="50" enableFilters="true"
enableFooters="true" enablePaging="true">' +
                '                <level>' +
                '                <columns>' +
                '                <column
type="checkbox"/>' +
                '                <column
dataField="id" headerText="ID" filterControl="TextInput"
filterOperation="Contains" footerLabel="Sum: " footerOperation="sum"
footerOperationPrecision="2"/>' +
                '                <column
dataField="type" headerText="Type" filterControl="TextInput"
filterOperation="Contains" />' +
                '                </columns>' +
                '            </level>' +
                '            ' +
                '        </grid>',
            dataProvider: [
                { "id": "5001", "type": "None",
"active": true },
                { "id": "5002", "type":
"Glazed", "active": true },
                { "id": "5005", "type": "Sugar",

```

```

"active": true },
    { "id": "5007", "type":
"Powdered Sugar", "active": false },
    { "id": "5006", "type":
"Chocolate with Sprinkles", "active": true },
    { "id": "5003", "type":
"Chocolate", "active": false },
    { "id": "5004", "type": "Maple",
"active": true }
    ]
});

});

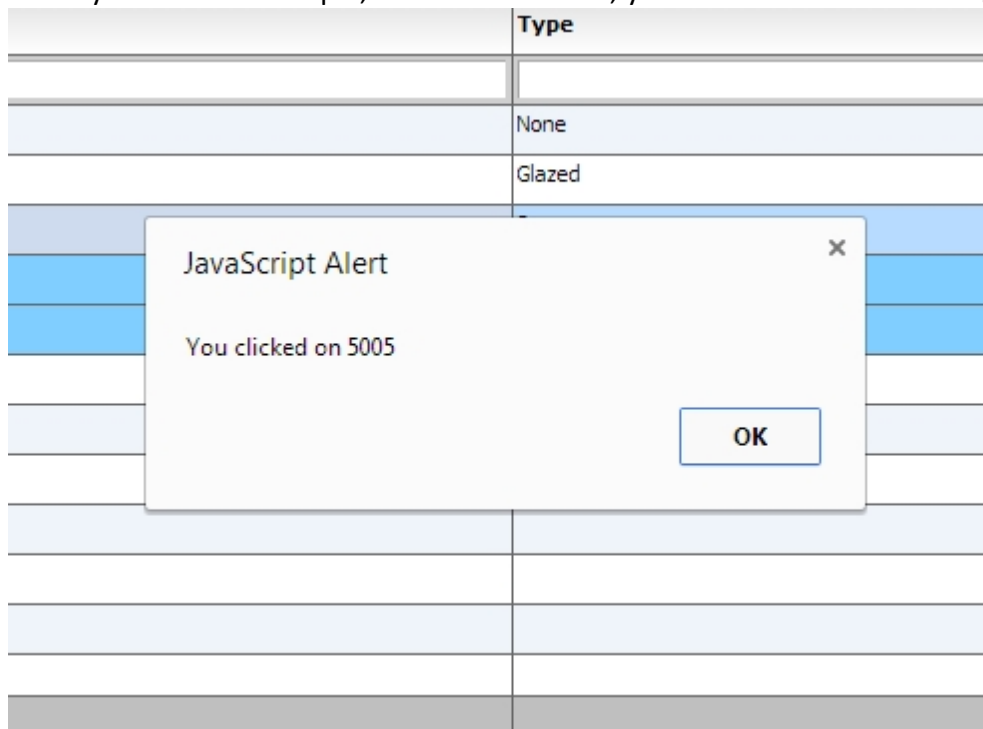
```

```

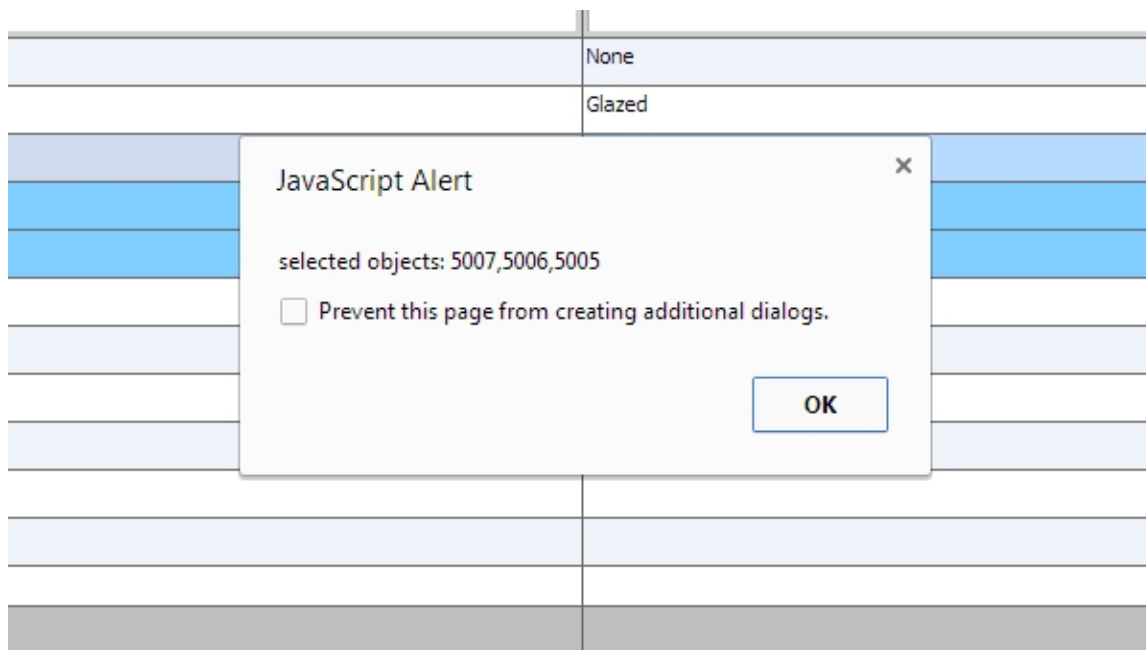
</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px; width: 100%;">
    </div>
</body>
</html>

```

When you run this example, and click on a row, you should see the following



Followed by something similar to:



What you just observed was event callbacks in action. In this example we just wired up a couple events, `itemClick` and `change`. Although they are not related, but they notify you of two different things, one that the user clicked on a row and the second that the selection of the grid has changed. Another important point to notice, is that the functions that are defined as the call backs, are defined under the “myCompanyNameSpace”. This way, all your callbacks can be defined in a way that is accessible to the grid as it navigates through the markup XML. The same concept is used for `itemRenderers` (which point to functions that are classes), and call back functions that we cover later in this chapter.

These are just a couple examples of a very rich set of events and notifications that the grid knows to dispatch. A full list of events is available below:

#### `afterExport`

Dispatched when the grid is finished exporting. At this point, the `textWritten` variable of the dispatched event is available to you, to customize the generated text if you need to. Event Type: `com.flexicious.export.ExportEvent`

#### `autoRefresh`

Dispatched when the autorefresh interval is hit. Event Type: `com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent`

#### `beforeExport`

Dispatched when the grid is about to be exported Event Type: `com.flexicious.export.ExportEvent`

#### `beforePrint`

Dispatched when the grid is about to be generated for the print, or the preview. The event has a handle to the grid that is being printed, as well as the `PrintDataGrid` instance. This lets you perform custom logic on the `PrintDataGrid` before the print occurs. Event Type: `com.flexicious.nestedtreedatagrid.events.FlexDataGridPrintEvent`



## beforePrintProviderSet

Dispatched before the beforePrint event, right prior to the data provider being set. Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridPrintEvent

## cellRendered

Dispatched when the cell is rendered completely. That is, the background and borders are drawn, the renderers are placed Use this event to perform post processing of the cell after the grid has applied all its formatting and positioning Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

## change

Dispatched when row selection or cell selection changes. Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

## clearPreferences

Fired when the grid needs to clear its preferences. Event

Type:com.flexicious.grids.events.PreferencePersistenceEvent

## columnsResized

Dispatched when the columns at this level are resized Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

## columnsShift

Dispatched when columns are dragged and dropped to change their position Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

## componentsCreated

Dispatched when all the cells snap to the calculated column widths. Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

## dynamicAllLevelsCreated

When enableDynamicLevels=true, this event is dispatched whenever all the dynamic levels are created. Event Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

## dynamicLevelCreated

When enableDynamicLevels=true, this event is dispatched whenever a new level is created. This event can be used to initialize the newly created level. Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

## filterPageSortChange

Dispatched when the grid's page, sort or filter state changes. Also Dispatched when an item, that was not previously opened is opened. Used in lazy loaded (filterPageSortMode=server) grids, to load a specific page of data from the backend.

**Event**

Type:com.flexicious.nestedtreedatagrid.events.ExtendedFilterPageSortChangeEvent  
[headerClicked](#)

Dispatched when any header cell is clicked Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[iconClick](#)

Dispatched when user clicks on an icon enabled via the enableIcon flag on a column Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[iconMouseOut](#)

Dispatched when user mouse outs on an icon enabled via the enableIcon flag on a column

Event Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[iconMouseOver](#)

Dispatched when user mouseovers on an icon enabled via the enableIcon flag on a column

Event Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[itemClick](#)

Dispatched when user clicks on a row in row selection mode or cell in cell selection mode

Event Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[itemClose](#)

Dispatched when the use clicks on the disclosure icon to collapse a previously opened item. Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

[itemClosing](#)

Dispatched when an item is about to close. If you call preventDefault() on the event, the item will not close Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

[itemDoubleClick](#)

Dispatched when user double clicks on a row in row selection mode or cell in cell selection mode Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

[itemEditBegin](#)

Dispatched when the editor is instantiated. Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridItemEditEvent

[itemEditBeginning](#)

Dispatched when the user attempts to edit an item. If you call preventDefault on this event, the edit session will not begin. Event

Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridItemEditEvent  
[itemEditCancel](#)

Dispatched when the edit session is cancelled. Event  
 Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridItemEditEvent  
[itemEditEnd](#)

Dispatched when the edit session ends. Event  
 Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridItemEditEvent  
[itemEditValueCommit](#)

Dispatched just before committing the value of the editorDataField property of the editor to the property specified by the datafield property of the column of the item being edited. If you call preventDefault on the event, the value will not be committed. Event  
 Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridItemEditEvent  
[itemFocusIn](#)

Dispatched when the item editor receives focus. Event  
 Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridItemEditEvent  
[itemLoad](#)

Dispatched only in server mode, when an item opens for the first time. Used to wire up an event handler to load the next level of data in lazy loaded grids.

Event  
 Type:com.flexicious.nestedtreedatagrid.events.ExtendedFilterPageSortChangeEvent  
[itemOpen](#)

Dispatched when the user clicks on the disclosure icon to expand a previously collapsed item. Event Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[itemOpening](#)

Dispatched when an item is about to open. If you call preventDefault() on the event, the item will not open Event Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[itemRollOut](#)

Dispatched when user rolls out of a row in row selection mode or cell in cell selection mode (only if rollover on a different item) Event  
 Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[itemRollOver](#)

Dispatched when user rolls over a row in row selection mode or cell in cell selection mode (only if rollover on a different item) Event  
 Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent

## loadPreferences

Fired In preferencePersistenceMode=server , when the grid needs to load its preferences.  
 Fired In preferencePersistenceMode=client , when the grid has successfully loaded preferences. Event Type:com.flexicious.grids.events.PreferencePersistenceEvent  
[pageSizeChanged](#)

Dispatched whenever the page size is changed.  
[pdfBytesReady](#)

Dispatched when the user clicks the 'Generate PDF' button on the Print Preview. In response to this user action, the grid simply prints it output to a byte array of page images. This byte array can then be sent to any pdf generation library either on the client or the server, for example Alive PDF. We provide integration code for AlivePDF out of the box. Event Type:com.flexicious.print.PrintPreviewEvent  
[persistPreferences](#)

Fired when the grid needs to persist its preferences. Event Type:com.flexicious.grids.events.PreferencePersistenceEvent  
[placingSections](#)

Dispatched prior to the widths, heights, x, and y of the various sections are calculated. By sections, we mean the left locked,right locked and unlocked content, header, footer, and body sections. Event Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent  
[prebuiltFilterRun](#)

Dispatched by the grid when a prebuilt filter is run. Event Type:com.flexicious.grids.events.WrapperEvent  
[preferencesLoading](#)

Fired right before preferences are being loaded and applied. Event Type:com.flexicious.grids.events.PreferencePersistenceEvent  
[printComplete](#)

Dispatched when the grid is sent to the printer. Event Type:com.flexicious.nestedtreedatagrid.events.FlexDataGridPrintEvent  
[printExportDataRequest](#)

Dispatched only in server mode, when the grid needs to print or export more data than is currently loaded in memory. Event Type:com.flexicious.grids.events.PrintExportDataRequestEvent  
[rendererInitialized](#)

Dispatched when the renderer is initialized.

Please note, any changes you made to the renderer stay in place when the renderer is recycled. So if you make any changes, please ensure that the changes are rolled back in the event handler first. For example, if you set the text to bold if a condition is met, then you should first set it to normal, check for the condition, and then set it to bold. This will ensure that if the renderer was previously used to display something that match the condition, and the current item does not, then we do not display bold text. Event Type:`com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent`  
[`rowChanged`](#)

When `enableTrackChanges` is set to true, this event is dispatched when any change (add,delete or update) is made to the grid. Only the updates made via the grid editors are tracked.  
[`scroll`](#)

Dispatched when the content is scrolled. Event Type:`mx.events.ScrollEvent`  
[`selectAllCheckBoxChanged`](#)

Dispatched when the top level select checkbox is changed

- Event Type:`com.flexicious.nestedtreedatagrid.events.FlexDataGridEvent`

[`toolbarActionExecuted`](#)

Dispatched when any toolbarAction is executed. Wrapper events' data contains the action being executed. If you call prevent default in this event, the default handling code does not execute

- Event Type:`com.flexicious.grids.events.WrapperEvent`

Some of these events are also dispatched by the Column and the Level, and you can review the entire list of events in the docs.

<http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGrid.html>

<http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumn.html>

<http://htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumnLevel.html>

All of these event names can be used in the configuration XML. Note, that the callback handler usually gets an event object, which contains a lot more information about the specific type of the event being dispatched.

## Function Callbacks

In addition to events, there are a number of API variables that are of type Function. This basically means that you can provide a function that executes your own logic. These differ from events primarily in the fact that the grid is asking something of you instead of just notifying you that something happened. For example, there is a function callback, `cellBackgroundColorFunction` that will pass you a cell, and expect a return value of a color, that you can then use to execute custom logic to provide the background color for the cells. Let's take a quick look at an example:

```
<!doctype html>
<html lang="en" >
<head>
  <meta charset="utf-8">
  <title>Simple</title>
  <!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-1.8.2.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-ui-1.9.1.custom.min.js"></
script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.ui.menu.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.toaster.js"></script>
  <!--End-->
  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/minified-compiled-jquery.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/examples/js/Configuration.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/themes.js"></script>
  <!--End-->
  <!--css imports-->
  <link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
flexicious/css/flexicious.css"
      type="text/css" />
  <link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
external/css/adapters/jquery/jquery-ui-1.9.1.custom.min.css"
      type="text/css" />
  <!--End-->
  <script type="text/javascript">

    $(document).ready(function () {
      myCompanyNameSpace.onSelectAll = function (evt) {
```

```

        var grid = evt.target;
        var selectedItems = grid.getSelectedObjects();
        alert("selected objects: " +
flexiciousNmosp.UIUtils.extractPropertyValues(selectedItems, "id"));
    }
    myCompanyNameSpace.onItemClick = function (evt) {
        var grid = evt.target;
        alert('You clicked on ' + evt.item.id);
    }
    myCompanyNameSpace.getCellBackground = function (cell) {
        if (!cell.getRowInfo().getIsDataRow()) {
            return null;
        }
        if (cell.getColumn().getDataField() == "type") {
            return 0x0000ff;
        } else if (cell.getRowInfo().getData().type ==
"Glazed") {
            return 0x00ff00;
        }
        return null; //this makes the grid use whatever color
it would have by default.
    }
    var grid = new
flexiciousNmosp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        configuration: '<grid id="grid"
itemClick="myCompanyNameSpace.onItemClick"
change="myCompanyNameSpace.onSelectAll"
cellBackgroundColorFunction="myCompanyNameSpace.getCellBackground"
enablePrint="true" enablePreferencePersistence="true"
enableExport="true" forcePagerRow="true" pageSize="50"
enableFilters="true" enableFooters="true" enablePaging="true">' +
            '
                <level>' +
            '
                <columns>' +
            '
                    <column
type="checkbox"/>' +
            '
                <column
dataField="id" headerText="ID" filterControl="TextInput"
filterOperation="Contains" footerLabel="Sum: " footerOperation="sum"
footerOperationPrecision="2"/>' +
            '
                <column
dataField="type" headerText="Type" filterControl="TextInput"
filterOperation="Contains" />' +
            '
                </columns>' +
            '
                </level>' +
            '
            ' +
            '</grid>',
        dataProvider: [
            { "id": "5001", "type": "None",

```

```

"active": true },
    { "id": "5002", "type":
"Glazed", "active": true },
    { "id": "5005", "type": "Sugar",
"active": true },
    { "id": "5007", "type":
"Powdered Sugar", "active": false },
    { "id": "5006", "type":
"Chocolate with Sprinkles", "active": true },
    { "id": "5003", "type":
"Chocolate", "active": false },
    { "id": "5004", "type": "Maple",
"active": true }
    ]
});
});

```

```

</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px; width: 100%;">
    </div>
</body>
</html>

```

When you run this, you should see:

ID	Type
5001	none
5002	none
5005	Sugar
5007	Powdered Sugar
5006	Chocolate with Sprinkles
5003	Chocolate
5004	Maple
Sum: 35028.00	

## Cells and Renderers

As we reviewed in the Grid Architecture topic, each cell in the grid is a descendant of the `FlexDataGridCell` class. The `FlexDataGridCell` has a `renderer` property, which is the actual component being displayed. The concrete classes that inherit from `FlexDataGridCell` are:

- `FlexDataGridHeaderCell`
- `FlexDataGridFilterCell`
- `FlexDataGridFooterCell`



- FlexDataGridPagerCell
- FlexDataGridLevelRendererCell
- FlexDataGridExpandCollapseHeaderCell
- FlexDataGridExpandCollapseCell
- FlexDataGridPaddingCell

Each of the above cells has a renderer property. The renderer is the actual component that is displayed on the UI. The FlexDataGridCell is responsible for sizing, positioning (based on padding), drawing the background, and drawing the borders. In case of the Header, Data or Footer cells the default renderer is a simple Label Control. For Filter, it is an instance of the IFilterControl. For the Pager, it is an IPager control. For the LevelRenderer it is an instance of the Class Factory that you specify in the nextLevelRenderer of the associated column Level. For the ExpandCollapse cells, it will draw an instance of the expand collapse icon on basis of the disclosureIcon style property. All the drawing happens in the drawCell method. It internally calls the drawBackground and drawBorder methods. Usually specifying the style attributes or the cellBackground/rowBackground/cellBorder/rowBorder functions is sufficient, but in case its needed, these methods can be overridden in a custom implementation, and this custom implementation can then be hooked in via the following properties on either the column or the level:

- dataCellRenderer
- headerCellRenderer
- footerCellRenderer
- pagerCellRenderer
- filterCellRender
- expandCollapseHeaderCellRenderer
- nestIndentPaddingCellRenderer
- expandCollapseCellRenderer

It is extremely rare to modify the built in renderers for the actual cells. More often than not, you will be providing custom renderers for the content of the cells. These renderers are the components that render the actual cell, or the container for each cell. Inside each cells is the actual component that renders the content of the cell. These are represented by the following properties on the column or the level

- itemRenderer
- headerRenderer
- footerRenderer
- pagerRenderer
- filterRender
- nextLevelRenderer

The above properties point to class factories for item renderer instances that display the data for each item in the column. You can specify a ClassFactory custom item renderer component as the value of these properties. The default item renderer is a Label class, which displays the item data as text.

If specifying custom html is not enough for your needs, you can use itemRenderers, which is a very powerful concept. Item Renderers are basically UIComponents that sit inside the various cells of the grid. The key about item renderers is that they should expose setData and getData methods. This method is called when the grid instantiates an itemRenderer and prepares it for display. Finally, another important thing to keep in mind is that each item renderer gets a parent property. This points to the parent FlexDataGridCell object. The actual type is FlexDataGridDataCell for itemRenderers, FlexDataGridHeaderCell for headerRenderers, FlexDataGridFilterCell for filterRenderers, and FlexDataGridFooterCell for footerRenderers.

### **Summary:**

The key points to remember about renderers:

- They sit inside the actual cell.
- They have to expose a setData method, which will get a value, which is the object being rendered by that renderer (the row data)
- They have a handle to the parent (via the this.parent) which is a pointer to a [IFlexDataGridCell](#) instance. Depending on whether it is a header, footer, filter, data, or level renderer, the parent will be a FlexDataGridCell of the appropriate type.
- The cell itself has a few important properties, namely the grid, the column, the level, and others which you can use to navigate the hierarchy.
- Renderers are complicated, but also powerful. If you want just some custom HTML in cells, you can easily accomplish that using labelFunctions. See this [topic](#) for more details.
- Usually, the cell is responsible for drawing the background and the border. You can use the cellBackgroundColorFunction if you just need to change cell colors. Renderers are only used when you have complex interaction within each cell.

### **Recycling, Virtualization and Buffering**

To understand itemRenderers, you have to understand why they are what they are and what our intentions were when we designed them. Suppose you have 1,000 records you want to show. If you think the list control creates 1,000 itemRenderers, you are incorrect. If the list is showing only 10 rows, the list creates about 12 itemRenderers—enough to show every visible row, plus a couple for buffering and performance reasons. The list initially shows rows 1–10. When the user scrolls the list, it may now be showing rows 3–12. But those same 12 itemRenderers are still there: no new itemRenderers are created, even after the list scrolls.

Here's what the grid does. When the list is scrolled, those itemRenderers that will still be showing the same data (rows 3–10) are moved upward. Aside from being in a new location, they haven't changed. The itemRenderers that were showing the data for rows 1 and 2 are now moved below the itemRenderer for row 10. Then those itemRenderers are given the data for rows 11 and 12. In other words, unless you resize the list, those same itemRenderers are recycled to a new location and are now showing new data.

This behavior complicates the situation for certain programming scenarios. For instance, if you want to change the background color of the cell in the fourth column of the fifth row, be aware that the itemRenderer for that cell may now be showing the contents of the twenty-first row if the user has scrolled the list.

So how do you make changes like this?

The itemRenderers must change themselves based on the data they are given to show. If the itemRenderer for the list is supposed to change its color based on a value of the data, then it must look at the data it is given and change itself. The way you do this, is expose a setData method. In the next section, we will take a look at a simple item renderer.

One thing many people try to do is access an itemRenderer from outside of the list. For example, you might want to make the cell in the fourth column of the fifth row in a DataGrid turn green because you've just received new data from the server. Getting that itemRenderer instance and modifying it externally would be a huge breach of the framework. Instead, you should perform the logic associated in the setData method. You cannot assume the renderer has been freshly created. When it is time to update the renderer, the components might be holding old values from the item it was rendering before. You cannot assume that there are only as many renderers created as you see on-screen. You cannot assume that the first renderer created is for the first visible row. The most common error when using renderers is in not resetting values. For example, if you have code that makes the text red if the value is below zero, you might write:

```
if (data.price < 0)
    this.domElement.style.color=0xFF0000;
```

This is incorrect. It will work the first time the renderer is created since the default color is black and will work if values are greater than 0, and if the value is less than zero. But once the renderer displays a value less than zero and turns the text red, if it is later asked to render a different value that is greater than zero, there is no code to turn the color back to black again. What will happen is that the data will look fine until you start scrolling or receiving updates, then the red will start to appear in funny places. The correct way is to write code like this:

```
if (data.price < 0)
    this.domElement.style.color=0xFF0000;
else
    this.domElement.style.color=0x000000;
```

### **A word about performance:**

Since we draw just the visible area, the memory/performance will depend more on the height and width of the grid and the number of columns you pack into the visible area. A couple of pointers to improve performance:

- 1) If you have a LOT of columns, set horizontalScrollPolicy to auto or on, which will use horizontal renderer recycling, improving performance.
- 2) If you have custom item renderers, be careful with them, if you nest too many heavy weight containers, you will run into performance issues. The renderers should be as light weight as possible, however, we have not yet seen any major issues with even complex item renderers. The key to keep in mind is that the setData method should do as little works as possible, since it is the one that gets called as you scroll.
- 3) If you have a very large amount of data (say hundreds of thousands of records), you should consider server side paging. Please review the [filterPageSortMode](#) topic for more informaion.

In the next section, we will take a look at how custom item renderers are written.

## Custom Renderers

Lets take a quick look at what a custom item renderer looks like. Note, this is a renderer that displays a data cell, but the same concept can be used to render custom interactive content in header, footer, filter, pager, as well as level renderer cells:

This is a renderer that uses a text input to render the data, instead of HTML label. Technically, you could just use a labelFunction and write out the HTML for the text input, but by having a renderer class do this, you have more control over interaction, as well as a handle to the cell that is rendering the data.

```
/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function(window)
{
    "use strict";
    var TextInputRenderer, uiUtil = flexiciousNmsp.UIUtils, flxConstants =
flexiciousNmsp.Constants;
    /**
     * A TextInputRenderer is a custom item renderer, that defines how to use custom
cells with logic that you can control
     * @constructor
     * @namespace flexiciousNmsp
     * @extends UIComponent
     */
    TextInputRenderer=function(){
        //make sure to call constructor
        flexiciousNmsp.UIComponent.apply(this,["input"]); //second parameter is the tag
name for the dom element.
    /**
     * This is a getter/setter for the data property. When the cell is created, it
belongs to a row
     * The data property points to the item in the grids dataprovider that is being
rendered by this cell.
     * @type {*}
     */
        this.data=null;
        //the add event listener will basically proxy all DomEvents to your code to
handle.
        this.addEventListener(this, flxConstants.EVENT_CHANGE, this.onChange);
    };
    myCompanyNameSpace.ItemRenderers_TextInputRenderer = TextInputRenderer; //add to name
space
    TextInputRenderer.prototype = new flexiciousNmsp.UIComponent(); //setup hierarchy
    TextInputRenderer.prototype.typeName = TextInputRenderer.typeName =
'TextInputRenderer'; //for quick inspection
    TextInputRenderer.prototype.getClassNames=function(){
        return ["TextInputRenderer", "UIComponent"]; //this is a mechanism to replicate
the "is" and "as" keywords of most other OO programming languages
    };

    TextInputRenderer.prototype.setWidth=function(w){
        flexiciousNmsp.UIComponent.prototype.setWidth.apply(this, [w]);
    }
    /**
     * This is important, because the grid looks for a "setData" method on the renderer.
```

```

    * In here, we intercept the call to setData, and inject our logic to populate the
text input.
    * @param val
    */
    TextInputRenderer.prototype.setData=function(val){
        flexiciousNmsp.UIComponent.prototype.setData.apply(this,[val]);
        this.domElement.value=val.legalName;
    };
    /**
    * This event is dispatched when the user clicks on the icon. The event is actually a
flexicious event, and has a trigger event
    * property that points back to the original domEvent.
    * @param event
    */
    TextInputRenderer.prototype.onChange=function(evt){
        this.data.legalName=this.domElement.value;//we use the dom element to wire back
the value to the data object.
        var cell = this.parent; //this is an instance of FlexDataGridDataCell (For data
rows)
        var column = cell.getColumn();//this is an instance of FlexDataGridColumn.
        column.level.grid.refreshCells();//this will re-render the cells.

    }
    //This sets the inner html, and grid will try to set it. Since we are an input
field, IE 8 will complain. So we ignore it since we dont need it anyway.
    TextInputRenderer.prototype.setText=function(val){

    };

}(window));

```

The way you would associate this renderer to a column is this:

```

'                                     <column headerText="Editable Name"
dataField="legalName" '+'
'                                     filterControl="TextInput"
filterOperation="BeginsWith" paddingLeft="5" paddingBottom="5" '+'
'                                     paddingRight="8" enableCellClickRowSelect="false"
itemRenderer="myCompanyNameSpace.ItemRenderers_TextInputRenderer">' +
'                                     </column>' +

```

Just like you associate renderers with data cells, you can do the same with header, footer, as well as filter cells.

Lets take a look at what a header renderer looks like:

```

/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function(window)
{
    "use strict";
    var CheckBoxHeaderRenderer, uiUtil = flexiciousNmsp.UIUtils, flxConstants =
flexiciousNmsp.Constants;
    /**
    * A CheckBoxHeaderRenderer is a custom item renderer, that you can use in a header
cell. In this case, we customize the header
    * so that instead of showing a header label, we show a checkbox that switches the

```

```

dataField flag on all items.
    * @constructor
    * @namespace flexiciousNmosp
    * @extends UIComponent
    */
    CheckBoxHeaderRenderer=function(){
        //make sure to call constructor
        flexiciousNmosp.UIComponent.apply(this,["input"]); //second parameter is the tag
name for the dom element.
        this.domElement.type = "checkbox"; //so our input element becomes a checkbox;
        this.domElement.checked=true;
        //the add event listener will basically proxy all DomEvents to your code to
handle.
        this.addEventListener(this,flxConstants.EVENT_CHANGE,this.onChange);
    };
    myCompanyNameSpace.ItemRenderers_CheckBoxHeaderRenderer = CheckBoxHeaderRenderer; //
add to name space
    CheckBoxHeaderRenderer.prototype = new flexiciousNmosp.UIComponent(); //setup
hierarchy
    CheckBoxHeaderRenderer.prototype.typeName = CheckBoxHeaderRenderer.typeName =
'CheckBoxHeaderRenderer'; //for quick inspection
    CheckBoxHeaderRenderer.prototype.getClassName=function(){
        return ["CheckBoxHeaderRenderer","UIComponent"]; //this is a mechanism to
replicate the "is" and "as" keywords of most other OO programming languages
    };

    /**
    * This event is dispatched when the user clicks on the icon. The event is actually a
flexicious event, and has a trigger event
    * property that points back to the original domEvent.
    * @param event
    */
    CheckBoxHeaderRenderer.prototype.onChange=function(event){

        //in the renderer, you have the handle to the cell that the renderer belongs to,
via the this.parent property that you inherit from flexiciousNmosp.UIComponent.

        var cell = this.parent; //this is an instance of FlexDataGridDataCell (For data
rows)
        var column = cell.getColumn(); //this is an instance of FlexDataGridColumn.
        //var dp = cell.level.getGrid().getDataProvider(); //this is a pointer back to the
grid and its dataprovider.
        var dp=this.data; //for header cells, specifically in case of nested grids, the
data property is a pointer back to the top level array, or the children array

        if(this.data.hasOwnProperty("deals")){
            //this means we are at a inner level checkbox header
            dp=this.data.deals;
        }
        //based upon which level this renderer appears.
        for (var i=0;i<dp.length;i++){
            dp[i][column.getDataField()] = this.domElement.checked;
        }

        column.level.grid.refreshCells(); //this will re-render the cells.
    };
    //This sets the inner html, and grid will try to set it. Since we are an input
field, IE 8 will complain. So we ignore it since we dont need it anyway.
    CheckBoxHeaderRenderer.prototype.setText=function(val){

    };
}(window));

```

Again, you can specify HTML for the headerText, but you may choose to use a JavaScript based renderer if you need more control over the interaction of the component within the HTML.

In the Demo Console, you can review the "Item Renderers" example for a running demo of how to use renderers.

ID	Editable Name	Website	Stock Price
20800	3M Co	3M Co	
<input type="checkbox"/> Deal Description Deal Amount Deal Date			
<input type="checkbox"/> Project # 1 - 3M Co - 9/2014 78,663 September-15-2014			
<input type="checkbox"/> Project # 2 - 3M Co - 10/2012 119,243 October-03-2012			
Count:2.00		Total:\$197,906	
20805	AFLAC Inc	AFLAC Inc	
<input type="checkbox"/> Deal Description Deal Amount Deal Date			
<input type="checkbox"/> Project # 1 - AFLAC Inc - 4/2011 111,416 April-22-2014			
<input type="checkbox"/> Project # 2 - AFLAC Inc - 4/2011 144,705 April-16-2013			
Count:2.00		Total:\$256,121	
20802	AK Steel Holding	AK Steel Holding	
20805	AT&T Inc	AT&T Inc	
20801	Abbott Laboratories	Abbott Laboratories	
20802	Adobe Systems	Adobe Systems	
20803	Advanced Micro De	Advanced Micro De	
20804	Aetna Inc	Aetna Inc	
20805	Affiliated Computer	Affiliated Computer Sys	

The same concept applies to other kinds of renderers, including:

- itemRenderer
- headerRenderer
- footerRenderer
- pagerRenderer
- filterRender
- nextLevelRenderer

For example, lets take a quick look at how we define the nextLevelRenderer. In the demo console, there is a running version of this example, that shows the inner level custom display:

```
/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function(window)
{
    "use strict";
    var NextLevelRenderer2, uiUtil = flexiciousNmsp.UIUtils, flxConstants =
flexiciousNmsp.Constants;
    /**
     * A NextLevelRenderer2 is a custom item renderer, that defines how to use custom
     cells with logic that you can control
     * @constructor
     * @namespace flexiciousNmsp
     * @extends UIComponent
     */
})
```

```

NextLevelRenderer2=function(){
    //make sure to call constructor
    flexiciousNmsp.UIComponent.apply(this); //second parameter is the tag name for the
    dom element.
    this.setHeight(50);
    /**
     * This is a getter/setter for the data property. When the cell is created, it
    belongs to a row
     * The data property points to the item in the grids dataprovider that is being
    rendered by this cell.
     * @type {*}
     */
    this.data=null;
};
myCompanyNameSpace.LevelRenderers2_NextLevelRenderer2 = NextLevelRenderer2; //add to
name space
NextLevelRenderer2.prototype = new flexiciousNmsp.UIComponent(); //setup hierarchy
NextLevelRenderer2.prototype.typeName = NextLevelRenderer2.typeName =
'NextLevelRenderer2'; //for quick inspection
NextLevelRenderer2.prototype.getClassNames=function(){
    return ["NextLevelRenderer2","UIComponent"]; //this is a mechanism to replicate
the "is" and "as" keywords of most other OO programming languages
};

/**
 * This is important, because the grid looks for a "setData" method on the renderer.
 * In here, we intercept the call to setData, and inject our logic to render the html
for the renderer.
 * @param val
 */
NextLevelRenderer2.prototype.setData=function(val){
    flexiciousNmsp.UIComponent.prototype.setData.apply(this,[val]);

    var html = "<fieldset><legend>Orgainzation Information</legend><table
style='width:100%'><tr>" +
        "<td style='border:solid 1px #000000'>Organization Name "+val.legalName+" </
td>" +
        "<td style='border:solid 1px #000000'>Sales Contact
"+val.salesContact.getDisplayName()+" </td>" +
        "<td style='border:solid 1px #000000'>Sales Contact
Phone:"+val.salesContact.telephone+" </td>" +
        "</tr><tr>" +
        "<td style='border:solid 1px #000000'>Annual
Revenue:"+flexiciousNmsp.UIUtils.formatCurrency(val.annualRevenue)+" </td>" +
        "<td style='border:solid 1px
#000000'>EPS:"+flexiciousNmsp.UIUtils.formatCurrency(val.earningsPerShare)+" </td>" +
        "<td style='border:solid 1px #000000'>Last Stock
Price:"+flexiciousNmsp.UIUtils.formatCurrency(val.lastStockPrice)+" </td>" +
        "</tr><tr>" +
        "<td style='border:solid 1px #000000'>Employees:"+val.numEmployees+" </td>"
+
        "<td colspan='2' style='border:solid 1px
#000000'>Address:"+val.headquarterAddress.toDisplayString()+" </td>" +
        "</tr></table></fieldset>";

    this.setInnerHTML(html);
}
}(window));

myCompanyNameSpace.levelRenderers2_creationCompleteHandler =function (evt){
    grid.validateNow();
    grid.expandAll();
}

```



```

myCompanyNameSpace.SAMPLE_CONFIGS["LevelRenderers2"]='<grid id="grid" enablePrint="true"
enableDrillDown="true"' +
,
enablePreferencePersistence="true"'+
,
enableCopy="true"'+
,
enableExport="true"
,
preferencePersistenceKey="levelRenderers2"
on'+flexiciousNmsp.Constants.EVENT_CREATION_COMPLETE
+'="myCompanyNameSpace.levelRenderers2_creationCompleteHandler">' +
,
<level enableFilters="true" enablePaging="true"
rendererHorizontalGridLines="true" ' +
,
rendererVerticalGridLines="true" pageSize="20" childrenField="deals"
enableFooters="true" selectedKeyField="id" ' +
,
nextLevelRenderer="myCompanyNameSpace.LevelRenderers2_NextLevelRenderer2"
levelRendererHeight="120">' +
,
<columns>' +
,
<column type="checkbox" />' +
,
<column enableCellClickRowSelect="false"
columnWidthMode="fitToContent" selectable="true" dataField="id" headerText="ID"
filterControl="TextInput"/>' +
,
<column truncateToFit="true"
enableCellClickRowSelect="false" columnWidthMode="fitToContent" selectable="true"
dataField="legalName" headerText="Legal Name"/>' +
,
<column dataField="headquarterAddress.line1"
headerText="Address Line 1" footerLabel="Count:" footerOperation="count"/>' +
,
<column dataField="headquarterAddress.line2"
headerText="Address Line 2"/>' +
,
<column dataField="headquarterAddress.city.name"
headerText="City" filterControl="MultiSelectComboBox" filterComboBoxBuildFromGrid="true"
filterComboBoxWidth="150"/>' +
,
<column dataField="headquarterAddress.state.name"
headerText="State" filterControl="MultiSelectComboBox" filterComboBoxBuildFromGrid="true"
filterComboBoxWidth="150"/>' +
,
<column dataField="headquarterAddress.country.name"
headerText="Country" filterControl="MultiSelectComboBox"
filterComboBoxBuildFromGrid="true" filterComboBoxWidth="150"/>' +
,
</columns>' +
,
</level>' +
,
</grid>';

```

<input type="checkbox"/>	ID	Legal Name	Address Line 1	Address Line 2	City	State	Country	
<input type="checkbox"/>					All	All	All	
▼	<input type="checkbox"/>	20800	3M Co	337 Gardner Rd	Suite #942	Stroudsburg	Ohio	United States
Organization Information								
Organization Name: 3M Co			Sales Contact: ROSETTA PEREZ			Sales Contact Phone: 732-133-3064		
Annual Revenue: 26,478			EPS: 6.96			Last Stock Price: 25.32		
Employees: 4693			Address: 337 Gardner Rd Suite #942 Suite #942, Stroudsburg, Ohio, United States					

## Class Factories

Everytime you have a property with the suffix "renderer", that usually means it is of type "ClassFactory". What this means is that the property itself points to an object that exposes a `newInstance()` method. This method is responsible for generating an "Instance" of the class. These are usually renderer instances. So for example, the following properties on the grid (or the level or column) are of type ClassFactory:

- `dataCellRenderer`
- `headerCellRenderer`

- footerCellRenderer
- pagerCellRenderer
- filterCellRender
- expandCollapseHeaderCellRenderer
- nestIndentPaddingCellRenderer
- expandCollapseCellRenderer
  
- itemRenderer
- headerRenderer
- footerRenderer
- pagerRenderer
- filterRender

In XML, you provide class factory values by using a fully qualified class name. For example:

```
'
                                <column headerText="Editable Name"
dataField="legalName" '+'
                                filterControl="TextInput"
filterOperation="BeginsWith" paddingLeft="5" paddingBottom="5" '+'
                                paddingRight="8" enableCellClickRowSelect="false"
itemRenderer="myCompanyNameSpace.ItemRenderers_TextInputRenderer">'+
                                </column>'+
```

However, in API, you could do the same thing in the following manner (this is what the XML builder does internally anyway):

```
col.itemRenderer=(new
flexiciousNmsp.ClassFactory(myCompanyNameSpace.ItemRenderers_TextInputRenderer));
```

Although you may not have to directory work with the ClassFactory class, below is the code for it. It is a very simple class responsible for generating the instances of renderers.

```
/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function () {
  "use strict";
  var ClassFactory;
  /**
   * A generator class that instantiates new classes of type classConstruct.
   * @constructor
   * @class ClassFactory
   * @namespace flexiciousNmsp
   * @extends TypedObject
   * @param classConstruct
   * @param [props]
   * @param [passPropertiesToConstructor]
   */
  ClassFactory = function (classConstruct, props, passPropertiesToConstructor) {
    flexiciousNmsp.TypedObject.apply(this);
    /**
     * The constructor to instantiate
     * @type Function
     */
    this.classConstruct = classConstruct;
    /**
     * The properties to apply to this constructors
```

```

    */
    this.properties = props;
    /**
     * If true, passes the properties to constructor, if false, loops through
properties and
     * sets them individually on the bean
     */
    this.passPropertiesToConstructor=passPropertiesToConstructor;

};
flexiciousNmsp.ClassFactory = ClassFactory; //add to name space
ClassFactory.prototype = new flexiciousNmsp.TypedObject(); //setup hierarchy
ClassFactory.prototype.typeName = ClassFactory.typeName = "ClassFactory";//for quick
inspection
ClassFactory.prototype.getClassNames = function () { //for support of "is" keyword
    return ["TypedObject", this.typeName];
};
/**
 * Creates a new instance of the object specified by the class construct
 * @return {*}
 */
ClassFactory.prototype.newInstance=function(){
    var obj;

    if(this.passPropertiesToConstructor){
        obj = new this.classConstruct(this.properties);
    }else{
        obj = new this.classConstruct();
        if(this.properties ){
            for (var prop in this.properties){
                obj[prop] = this.properties[prop];
            }
        }
    }

    return obj;
};

}());

```

## Filter Options - Deep Dive

Filters are a key feature of the grid product. Although most use cases for filtering are handled elegantly by the built in functionality, it is often times necessary to implement customizations. In this section, we will review some of the concepts associated with filtering data.

### Filter Page Sort Mode

The grid exposes a key property, `filterPageSortMode`. This property plays a very important role in how the filtering (as well as paging and sorting) mechanism behaves.

This is a key property to understand. There are two values for this property, “server” and “client”. The default mode is client. Let us assume you are working on a Human Resource Management Application. You have a DataGrid that shows a list of Employees. You have a few hundred Employees to show. This can be easily accomplished by getting the list of employees from the server, and setting the data provider property of the grid to an Array that contains this list of Employees. As you need, you enable filters, footers, paging, export, print, etc, and all is well and good. You will be up and running in no time with `filterPageSortMode=client`. The grid

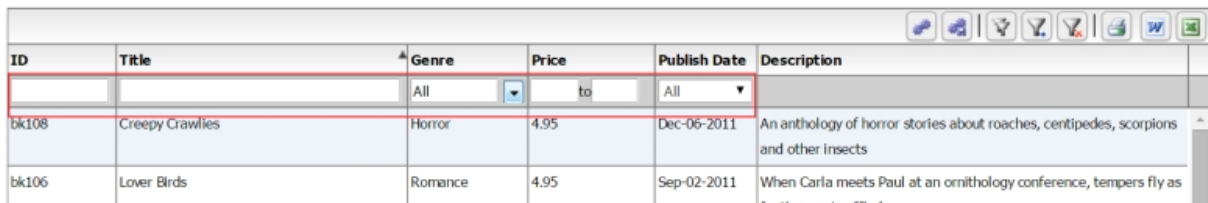
will take care of paging, filtering and cross page sorting for you. However, now consider a scenario where you have to display a time sheet search page. Each employee creates hundreds of timesheets each year, and multiply that by the number of employees, you are looking at thousands, even hundreds of thousands of records of data. Although it may be possible to load thousands of records in any DataGrid (including ours) with no noticeable drag, this is not recommended, and often unnecessary. What we suggest in this scenario is you use the `filterPageSortMode = "server"`. What Flexicious does, in this setup, is it assumes that the current Array is a part of a much larger record set, and you are showing just the current page of data. So if there are 100,000 records in the database, and the `pageSize` is 50, grid will show the 50 records, and provide paging UI for the user to navigate to any record in the result set. At any point in time, no more than 50 records will be loaded on client memory. This setup, will require a little more effort in terms of configuration, but it will be considerably easier to do this with Flexicious as opposed to building it yourself, because the product is designed to cater to a scenario like this.

The Filter/Page/Sort Mode. Can be either "server" or "client". In client mode, the grid will take care of paging, sorting and filtering once the dataprovider is set. In server mode, the grid will fire a `FilterPageSortChangeEvent` named `filterPageSortChange` that should be used to construct an appropriate query to be sent to the back end.

For example of the server mode, please review the Fully Lazy Loaded example in the demo console, here : [http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=Fully%20Lazy%20Loaded](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=Fully%20Lazy%20Loaded)

### Built in filter controls

Filters are UI controls embedded in the top of the grid to narrow down the grid data.



ID	Title	Genre	Price	Publish Date	Description
		All	to	All	
bk108	Creepy Crawlies	Horror	4.95	Dec-06-2011	An anthology of horror stories about roaches, centipedes, scorpions and other insects
bk106	Lover Birds	Romance	4.95	Sep-02-2011	When Carla meets Paul at an ornithology conference, tempers fly as feathers not ruffled

There are a number of built in filter controls, and you can even write your own by implementing the `IFilterControl` interface. To be implemented by any control that can participate in the filtering mechanism. There are the following controls available out of the box:

- `TextInput`
- `TriStateCheckBox`
- `ComboBox`
- `MultiSelectComboBox`
- `DateComboBox`
- `DateRangeBox`
- `NumericRangeBox`
- `NumericTextInput`

There may be situations where you might need to either extend the built in functionality of each of these filter controls or write your own filter controls, both of which are possible.

You can write your own custom filter controls by extending any of these, or by implementing the `IFilterControl`, `ICustomMatchFilterControl`, or `IDynamicFilterControl` interfaces.

In the rest of this chapter, we will go over some of these concepts.

### Built in filter control options

As we discussed earlier, there are the following filter controls available out of the box with the grid:

- `TextInput`
- `TriStateCheckBox`
- `ComboBox`
- `MultiSelectComboBox`
- `DateComboBox`
- `DateRangeBox`
- `NumericRangeBox`
- `NumericTextInput`

Each of these is a stand alone control in itself, but has some key properties that can be used to customize its behavior:

#### Text Input Filter

The Text Input filter is the most commonly used filter, so lets take a quick look at the available options.

First, the way you use these controls is that you define which filter you want on the column:

```
'
    <column id="colId" dataField="id" headerText="ID"
    filterControl="TextInput"
```

With each of the filter controls, there are pass through properties that enable various options on the filter controls. For example, for `TextInput`, the properties in red can be used to customize its behavior:

```
'
    <column id="colId" dataField="id" headerText="ID"
    filterControl="TextInput" filterWaterMark="Search"'+
    columnLockMode="left" filterIcon="'+ myCompanyNameSpace.IMAGE_PATH +'/'
    search_clear.png"'+
    enableFilterAutoComplete="true" clearFilterOnIconClick="true"/>'+
```

As you can imagine, we set the following properties here:

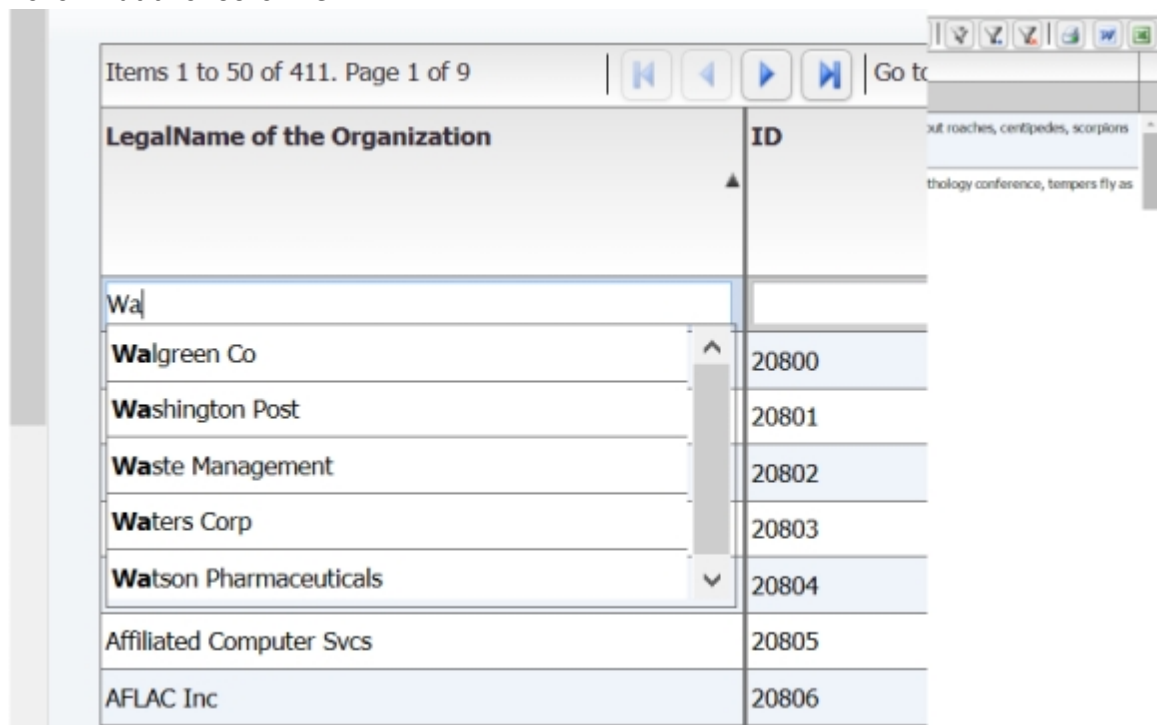
1. `filterControl="TextInput"` which made the column have a text input filter.
2. `filterWaterMark="Search"` which made the text input have a Search watermark
3. `filterIcon= ...` which made the clear icon visible
4. `enableFilterAutoComplete="true"` which made the text input auto complete enabled, so the filter can drill down.
5. and finally, `clearFilterOnIconClick="true"`, which makes the filter clear out as soon as the user hits the clear button.

In addition, there is also the `filterOperation` which you can use to control the behavior. This is the operator to apply when doing the conversion. See `FILTER_OPERATION_TYPE` constant values

from `com.flexicious.gridfilters.FilterExpression`. Here are a list of options:

- Equals
- NotEquals
- BeginsWith
- EndsWith
- Contains
- DoesNotContain
- GreaterThan
- LessThan
- GreaterThanEquals
- LessThanEquals
- InList
- NotInList
- Between

This is what this looks like:



LegalName of the Organization	ID
Walgreen Co	20800
Washington Post	20801
Waste Management	20802
Waters Corp	20803
Watson Pharmaceuticals	20804
Affiliated Computer Svcs	20805
AFLAC Inc	20806

One final aspect of the TextInput Filter, is that it implements `IDelayedChange`. This is a marker interface to indicate that the object support a delayed changed event Used in

filtering - so we dont trigger the filter event for every key stroke in the textbox. It has a delay duration to control how long to wait before we trigger the filter.

```
//BEING DELAYED CHANGE PROPERTIES
/**
 * The amount of time (in milliseconds) to wait before dispatching the
DELAY_CHANGE event.
 * @property delayDuration
 * @type int
 * @default 500
 */
this.delayDuration= 500; // delay in milliseconds before "delayedChange" event is
dispatched.
/**
 * Whether or not to enable the DELAY_CHANGE event.
 * @property enableDelayChange
 * @type int
 * @default 500
 */
this.enableDelayChange= true;
//END DELAYED CHANGE PROPERTIES
//=====DELAYED CHANGE INCLUDE=====//
TextInput.prototype.dispatchEvent=function(evt){
    if(this.enableDelayChange)
    {
        if((evt.type == flxConstants.EVENT_KEY_UP &&
(this.textBoxValue!=this.getTextBox().value)) || evt.keyCode==8)
        {
            // if change event, intercept
            if (this.timerInstance == null )
            {
                this.timerInstance = new
flexiciousNmsp.Timer(this.delayDuration,1);
                this.timerInstance.addEventListener(this,flxConstants.EVENT_TIMER
_COMPLETE, this.onTimerComplete,false,0,true);
            }
            this.timerInstance.reset(); // reset if already set...
            this.timerInstance.repeatCount = 1;
            // starts the timer ticking
            this.timerInstance.start();
        }
    }
    return flexiciousNmsp.UIComponent.prototype.dispatchEvent.apply(this,[evt]);
};

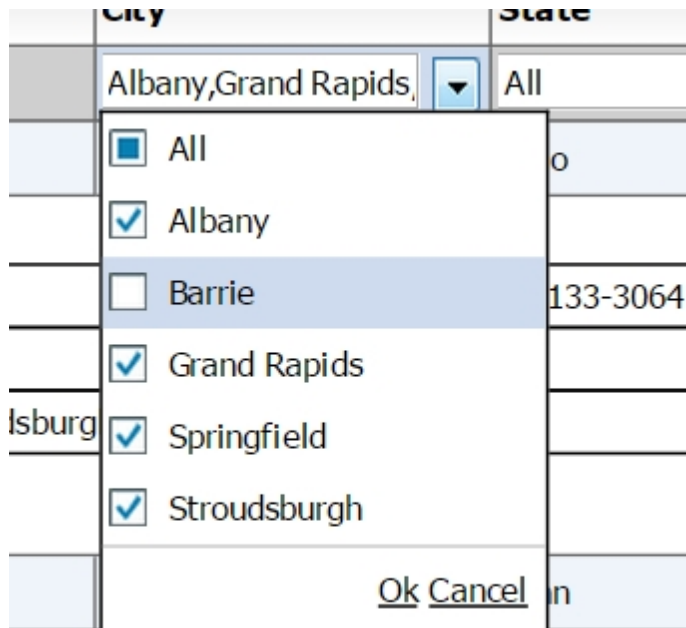
TextInput.prototype.onTimerComplete=function (event){
    this.dispatchEvent(new flexiciousNmsp.FlexDataGridEvent("delayedChange"));
    if(this.timerInstance){
        this.timerInstance.stop();
        this.timerInstance=null;
    }
};
//=====END DELAYED CHANGE INCLUDE=====//
```

Finally, it does support autocomplete, and by default the list of items in the auto complete are a distinct list of values from the column. However, you can also control the list of items by specifying a filterComboBoxDataProvider, filterComboBoxLabel field and filterComboBoxDataField properties.

## Multi Select Combo Box Filter

Another very useful filter is the MultiSelectComboBox:

This allows you to select one or more values to search across the column.



The way you configure a column to have a MultiSelectComboBox as the filter is:

```
<column dataField="headquarterAddress.city.name"
headerText="City" filterControl="MultiSelectComboBox" filterComboBoxBuildFromGrid="true"
filterComboBoxWidth="150"/>+
```

The following properties are useful when you are using the MultiSelectComboBox (Some of these also apply to the ComboBox filter control)

1. **filterComboBoxDataProvider**: Dataprovider to use to build the list of values to display in the filter control, only applicable if the filterControl is a ISelectFilterControl, IMultiSelectFilterControl, ISingleSelectFilterControl inherit from ISelectFilterControl, and MultiSelectComboBox RadioButtonList ComboBox DateComboBox all inherently implement ISelectFilterControl. It is not required to set this field. You may set the filterComboBoxBuildFromGrid property to true which will automatically build this collection on basis of distinct values in the grid.
2. **filterComboBoxLabelField**: Used in conjunction with the filterComboBoxDataProvider field, used to set the value of the label field for the associated ISelectFilterControl. The name of the field in filterComboBoxDataProvider array objects to use as the label field. Property; specifies a field in each item to be used as display text. This property takes the value of the field and uses it as the label. The default value is "label". For example, if you set the labelField property to be "name". "Nina" would display as the label for this {name: "Nina", id: 123};
3. **filterComboBoxDataField**: Used in conjunction with the filterComboBoxDataProvider field, used to set the value of the dataField for the associated ISelectFilterControl. The name of the field in filterComboBoxDataProvider array objects to use as the data field. Property;

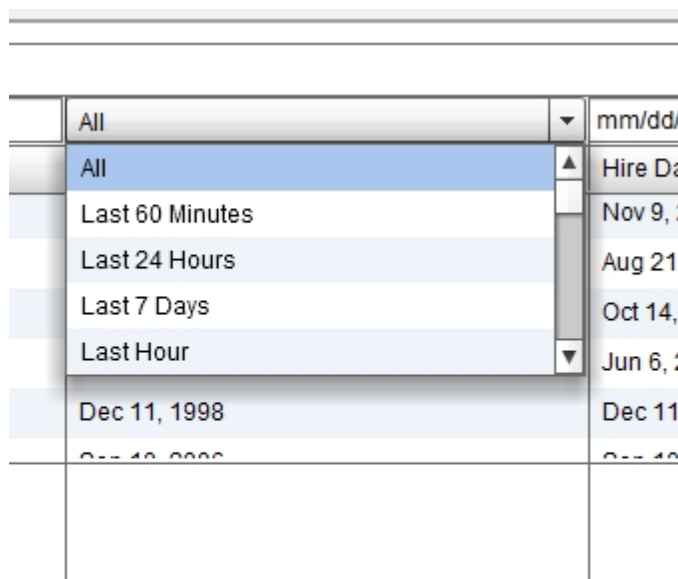


specifies a field in each item to be used as display text. This property takes the value of the field and uses it as the label. The default value is "data". For example, if you set the labelField property to be "id". 123 would be stored as the value {name: "Nina", id: 123};

4. **filterComboBoxBuildFromGrid**: Flag, if set to true, will automatically build this collection on basis of distinct values in the grid. Usually, you would want to set this flag to true, because it automatically builds out the list of distinct items for you, sets the appropriate filterComboBoxDataField, filterComboBoxLabelField, and you do not have to specify a filterComboBoxdataProvider. The only scenario where you would want to set this to false, is when you explicitly specify a filterComboBoxDataProvider. This is usually done only when filterPageSortMode=server, because not all the data across the entire dataset is present when you have server paging. The other situation is where you have very large datasets and you don't want the grid to spend time iterating through the dataprovider to identify the unique values.

### Date Combo Box Filter

Another very useful filter is the DateComboBox filter. This basically lists a series of predefined set of DateFilter options.



The way you configure a column to have a DateComboBox as the filter is:

```
<column dataField="hireDate" headerText="Date"
  filterControl="DateComboBox">
```

The following properties are useful when you are using the DateComboBox:

**filterDateRangeOptions**: List of options that are available in the DateComboBox filter. The default is a blank array which makes all available Date Ranges show up. For a full list of DateRanges, see the DateRange class

Another point to note is that when the user picks the "Custom" date range, the grid will show a

popup with two date pickers to pick the date range. You can control the format of the Date Picker by setting the

```
flexiciousNmsp.Constants.DEFAULT_DATE_FORMAT = "M-dd-yy";
```

Note that the date format is specific to the library being used. If you use moment.js, the format will be different. For jQuery, we use the built in JQDatePicker, which has the format described above.

## Other Controls

Aside from the above, there are a group of other filter controls that you may use for filtering data. Most of these are not as extensively used, so we are going to just mention them here. The key to keep in mind is that all these controls are classes. You can extend these classes and customize their behavior.

## Anatomy of a filter control

The Grid exposes a robust mechanism in which you can create and plug in your own filter controls.

Step1: Choose the appropriate IFilterControl Interface OR extend an existing FilterControl.

Filter Controls in most cases, need to implement a derivative of the IFilterControl interface (See the end of this article if none of the below interfaces apply to your scenario). This gives you the following options:

- IDateComboBox
- IDynamicFilterControl
- IRangeFilterControl
- ISelectedBitFilterControl
- ISelectFilterControl
- ITextFilterControl
- ITriStateCheckBoxFilterControl

Now consider what you wish your filter control to behave like. Are you going to be executing Text based search? Then your best bet is ITextFilterControl (Or extending TextInput). Are you performing Date Range Searches? Then you want the IRangeFilterControl. Do you need complete control over the FilterExpression that gets generated? Then you need the Dynamic Filter Control. Do you need even more control and have a requirement to build in custom logic in your filter control? Then you can use the ICustomMatchFilterControl.

Step 2:

Now, there are a bunch of properties and methods that the IFilterControl comes along with. Most of these are simply getters and setters, for example, below is the code from our TriStateCheckBox filter control:

```
TriStateCheckBox=function(){

    flexiciousNmsp.UIComponent.apply(this,["span"]);
    //BEGIN FILTER PROPERTIES
    /**
     * Whether or not there is an active search
     * @property hasSearch
     * @type Boolean
     * @default false
     */
    this.hasSearch = false;
    /**
     * Whether or not this control has been registered. This should not be set by
```

your code.

```

    * @property registered
    * @type Boolean
    * @default false
    */
    this.registered = false;
    /**
     * This is usually automatically set, you don't have to manually set it,
     * unless you're sending strings as Date objects. When set, will attempt
     * to first convert the current value to the type you specified and then
     * do the conversion.
     * Values : auto,string,number,boolean,date
     * @property filterComparisionType
     * @type String
     * @default auto
     */
    this.filterComparisionType = "auto";
    /**
     * The field to search on, usually same as the data field.
     * @property searchField
     * @type String
     * @default null
     */
    this.searchField = null;
    /**
     * The filter operation to apply to the comparison
     * See the FilterExpression class for a list.
     * Please note, for CheckBoxList and MultiSelectComboBox, this field
     * defaults to "InList" and is ignored when set.
     * Valid values are :
    "Equals,NotEquals,BeginsWith,EndsWith,Contains,DoesNotContain,GreaterThan,LessThan,Greate
    rThanEquals,LessThanEquals,InList,NotInList,Between"
     * @property filterOperation
     * @type String
     * @default Equals
     */
    this.filterOperation = null;
    /**
     * The event that the filter triggers on. Defaults to "change", or if the
     * filterRenderer supports com.flexicious.controls.interfaces.IDelayedChange,
then
     * the delayedChange event.
     * @property filterTriggerEvent
     * @type String
     * @default change
     */
    this.filterTriggerEvent = "change";
    /**
     * The grid that the filter belongs to - can be null
     * if filter is used outside the grid
     * @property grid
     */
    this.grid = null;
    /**
     * The grid column that the filter belongs to - can be null
     * if filter is used outside the grid
     * @property gridColumn
     * @return
     */
    this.gridColumn = null;
    //END FILTER PROPERTIES

```

Step 3:

Dispatch the appropriate event: The Flexicious DataGrid is setup so that the filter control should dispatch an event (which is configurable) that the datagrid listens for, and when this event is dispatched, a filter is run. By default, this event is the "change" event. For controls that implement IDelayedChange, it is the delayed change event. So, this is something you need to keep in mind while implementing your own filter control. The change event being dispatched by default will trigger the filter. You can control this via setting the filterTriggerEvent on the column, if you want a specific event to trigger the filter. For example, our TristateCheckBox dispatches a delayed change, which basically lets the user flip through multiple states of the checkbox before running the filter, so a filter is not run on every state flip. Same thing applies to our TextInput Filter, so a filter is not run on every key stroke, rather it runs when the user pauses typing for a predetermined interval.

Below is the code that does this:

```
//=====DELAYED CHANGE INCLUDE=====//
TriStateCheckBox.prototype.dispatchEvent=function(event){
    if(this.enableDelayChange)
    {
        if(event.type == flxConstants.EVENT_CHANGE)
        {
            // if change event, intercept
            if (this.timerInstance == null)
            {
                this.timerInstance = new flexiciousNmsp.Timer(this.delayDuration);
                this.timerInstance.addEventListener(this,flxConstants.EVENT_TIMER_COM
PLETE, this.onTimerComplete,false,0,true);
            }
            this.timerInstance.reset(); // reset if already set...
            this.timerInstance.repeatCount = 1;
            // starts the timer ticking
            this.timerInstance.start();
        }
    }
    return flexiciousNmsp.UIComponent.prototype.dispatchEvent.apply(this,[event]);
};

TriStateCheckBox.prototype.onTimerComplete=function (event){
    this.dispatchEvent(new flexiciousNmsp.FlexDataGridEvent("delayedChange"));
    this.timerInstance.stop();
    this.timerInstance=null;
};
//=====END DELAYED CHANGE INCLUDE=====//
```

#### Step 4:

Implement the Filter Interface methods: There are a few methods that we need to implement based on which filter interface we are implementing: For example ,if you are implementing IRangeFilterControl, we need to provide searchRangeStart and searchRangeEnd. This is then used by the filter logic to build a BETWEEN expression. If either searchRangeEnd or searchRangeStart is empty, the entire filter expression for this control is ignored. Please review the documentation of the particular interface to identify which methods you need to implement.

The other important methods are: setValue, getValue and clear. These methods are called by our API, when we rebuild the filter row in response to a refresh of the layout. So we call getValue, store it temporarily, destroy the filter component, create a new one, and then call setValue with the previously persisted filter Value. It is the responsibility of the filter control, to

restore its state, from information it we give it in setFilterValue (which it gave us in getFilterValue). For example, below is the code from our TriStateCheckBox that does this. Note that each filter control would have its own way of handling these methods.

```
//=====IFilterControl Methods=====/  
TriStateCheckBox.prototype.clear = function () {  
    this.setSelectedState(flexiciousNmsp.TriStateCheckBox.STATE_MIDDLE);  
};  
/**  
 * Generic function that returns the value of a IFilterControl  
 */  
TriStateCheckBox.prototype.getValue = function () {  
    return this.getSelectedState();  
};  
/**  
 * Generic function that sets the value of a IFilterControl  
 * @param val  
 */  
TriStateCheckBox.prototype.setValue = function (val) {  
    this.setSelectedState(val);  
};  
//=====End IFilterControl Methods=====/
```

#### Step 5:

Using the Filter Control : Once we have a filter control implemented, its just a matter of setting its name (along with the package) in the filterRenderer property of the column.

All of that said, there may be scenarios where you need to massage the value of the filter before processing it, or you may need to apply your own filtering logic to the filter control. There are ways to do this:

1) Implement IConverterControl: This interface is used, when you need to process the value of the item being compared. For example. A database sends down date columns as Strings. You have a DateComboBox as a filter. By default, the filter mechanism will try to compare two dates against a string. To combat this situation, you implement IConverterControl, and in the convert method, convert the string to a Date. We do this in the sample, look at MyDateComboBox.as (The code is commented, but its there).

2) Implement ICustomMatchFilterControl : This lets you completely encapsulate the logic for performing a filter. Please note, only use a FilterControl that implements this interface in filterPageSortMode=client.

Note : If you are writing your own custom filter controls, you MUST inherit from the flexiciousNmsp.UIComponent class. Please review the guide on understanding the structure of the UIComponent class.

### Sample Custom Filter Control

One of the most powerful features of the HTMLTreegrid is the ability to plug-in custom, interactive and rich content within each cell of the Grid. And not just that, we enable API hooks for these highly functional components to interact with the grid seamlessly, almost as if they were a part of the grid to begin with. Just scroll down to the bottom of this post and look at the UI, it appears as if the filter picker is an intrinsic part of the grid, where in fact, it is a

completely custom plugged in component with rich functionality encapsulated within its bounds.

One of the most powerful features of the HTMLTreegrid is the ability to plug-in custom, interactive and rich content within each cell of the Grid. And not just that, we enable API hooks for these highly functional components to interact with the grid seamlessly, almost as if they were a part of the grid to begin with. Just scroll down to the bottom of this post and look at the UI, it appears as if the filter picker is an intrinsic part of the grid, where in fact, it is a completely custom plugged in component with rich functionality encapsulated within its bounds.

In this example, the DynamicFilterControl and DynamicFilterPopup are a custom filter component. If you look at the code for DynamicFilterControl you can see how this gets plugged in. Although this example demonstrates a fairly complicated piece of functionality, the concepts are quite simple - That is the use of renderers to implement custom filter functionality. Most of the code is actually related to implementation of the filter itself. This example implements the ICustomMatchFilterControl interface. This is applicable in scenarios where you filter control needs to perform additional logic beyond what is covered by the built in filter operators. You have to implement the isMatch function to decide whether an object in the dataprovider matches the filter controls filter criteria.

On running index.html you will see the following screen with "All" option at the top on column.

On clicking it will show a popup for selecting criteria, select any criteria and click ok to apply on grid. "Remove filter" will remove the criteria.

Name	Start Date	End Date	Monthly Fees	Weekly Fees	Conc
	All	All	All	All	
Learn Java Swing	2014-08-18	2014-08-19	743	269	Brahi
Learn Angular Js	2014-08-18	2014-08-19	668	85	Jev
Learn AS	2014-08-18	2014-08-19	181	237	Lio
Learn Java Spring	2014-08-18	2014-08-19	216	295	Jack
Learn Ajax	2014-08-18	2014-08-19	183	101	Cour
Learn Assemblies	2014-08-18	2014-08-19	601	187	Willie
Learn JDK 1.8	2014-08-18	2014-08-19	380	145	Billa
Learn Xpath	2014-08-18	2014-08-19	330	92	Vicky

For the files associated with this help topic, please refer to <http://blog.htmltreegrid.com/post/Setting-up-dynamic-filter-control-for-html-grid.aspx>

## External Filters and Filter Functions

In addition to embedded filters, you can also have a function external to the grid, that determines if a particular object matches a filter. So, for example, you could have an external filter panel, that would filter the records on basis of a column that is not shown in the grid, or some custom logic that cannot be easily wrapped in a filter control.

The screenshot shows a hierarchical tree grid. The top level contains employee records. One employee, Tony Devanberry, is expanded to show a 'Project' record (Mapping). This project is further expanded to show 'TimeSheet' records. A 'Settings' dialog box is open over the 'TimeSheet' records, showing checkboxes for 'Time Sheet-1' and 'Time Sheet-2', and a 'Search' button. The grid columns include Employee Name, Title, Email Address, Department, Project, Role On Project, Project Start, TimeSheet Title, Hours, and Rate.

Employee Name	Title	Email Address	Department
Tony Devanberry	Architect	tdony@email.com	IT
Project			
Mapping	Lead Developer		08/08/2008
TimeSheet			
TimeSheet Title	Hours	Rate	
Time Sheet-1		100	
Time Sheet-2		100	
Time Sheet-2	48	100	
Jason Parker	Programmer	jason@email.com	Support
Project			
Grid Support	Developer		06/07/2008
TimeSheet			
TimeSheet Title	Hours	Rate	
Time Sheet-1	42	100	
Time Sheet-2	49	100	
Mapsharp	Architect		06/14/2009
TimeSheet			
TimeSheet Title	Hours	Rate	

The way you do this, is to specify a filterFunction for the level you are interested in filtering. So if you want to filter the top level records, you would set the filterFunction and the top level. If you are interested in filtering inner level records, you would navigate to the level you are interested in, and then specify a filterFunction there. This is particularly applicable for hierarchical grids (as opposed to flat grids) because hierarchical grids don't have a filter bar for sub grids. For a running example of this functionality, please review [http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=External%20Filter](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=External%20Filter)

```

grid.getColumnLevel().nextLevel.nextLevel.filterFunction=myCompanyNameSpace.externalFilter_filterDeviceTypes;
myCompanyNameSpace.externalFilter_filterDeviceTypes=function(item){
    if(SOME_LOGIC_HERE)
        return true;
    else
        return false;
};

```

## Export Options - Deep Dive

Flexicious, out of the box, offers you the ability to Export the data grid data to Word, Excel, HTML, Text and XML. We also provide a plug-in based mechanism, where in you can add your own exporters.

There are certain aspects of the Export mechanism that require some additional insight, which we will cover in this chapter.

### Built in Exporters

Out of the box, there are the following Exporters available:

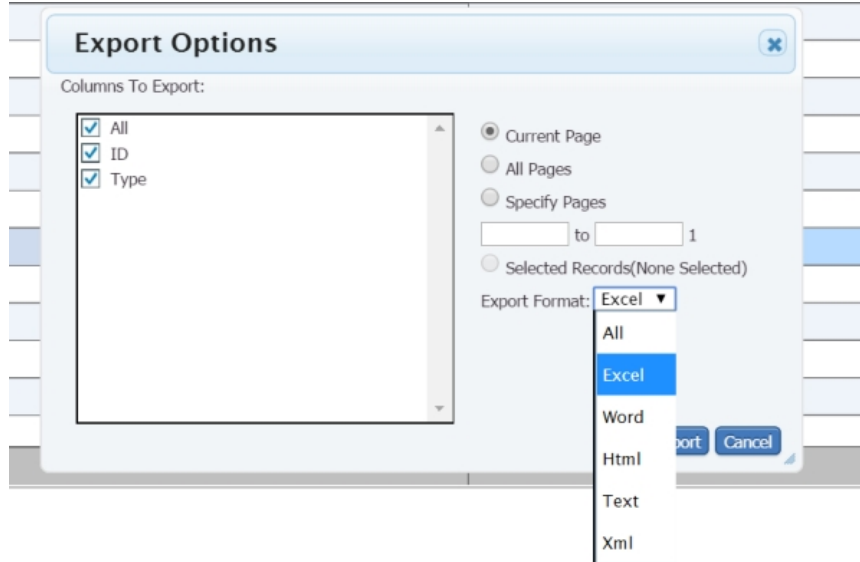
- CsvExporter
- DocExporter
- HtmlExporter
- XmlExporter
- TxtExporter
- ExcelExporter, (which basically creates HTML, but names the file as XLS, which excel will



open)

All these exporters can be found in the export namespace. They all inherit from the `Exporter` class.

By default when you do an export, you will see the following popup:



The grid has two properties, `grid.excelOptions` and `grid.wordOptions`, which you can use to customize the behavior of the exporters. Although the grid does a lot more than just word and excel, these are the only two that get their own buttons on the grid's toolbar, to save space and reduce clutter. The `grid.excelOptions` and `grid.wordOptions` both are associated with the toolbar icons for excel export and word export. The other exporters inherit their options from these two, depending on which one was used to launch the window.

The `ExportOptions` class is quite useful to customize the behavior of the export mechanism. Full documentation here : <http://www.flexicious.com/resources/docs29/com/flexicious/export/ExportOptions.html>. This class inherits from `PrintExportOptions`, which contain common properties between the printing and exporting mechanism. The `PrintOptions` class is documented here: <http://www.flexicious.com/resources/docs29/com/flexicious/grids/events/PrintExportOptions.html>

## Echo URL

By default, we use the `downloadify` library to generate the file on the client and save it on disk. However, since `downloadify` depends on flash player, in environments without the flash plugin, we depend on a server echo mechanism. You can choose to only use the server echo if you want consistent behavior. You can do so by setting `grid.excelOptions.enableLocalFilePersistence=false;`

Here is how that works:

When you do a export (or pdf - more on that later), we basically build a string on the client application. Since a browser without Flash player cannot write to your hard drive (except

cookies and such), we need to go through the traditional http File Buffering process. So what we do is this : We build the excel/word/html string, and send it to the server, which simply writes it back, in addition to setting the content type. Now, we advise that you implement your own url to buffer this string back, but do provide our own url - the one mentioned above to perform the buffering. The server side code for this is very simple:

//This is C# code, you could just as easily do the same thing in Java or PHP or whatever it is that sits on your server.

```
var extension = Request["extension"];
var contentType = Request["contentType"];
var body= Request["body"];
Response.ClearContent();
Response.AddHeader("content-disposition", string.Format("attachment;
filename=Download.{0}",extension));
Response.ContentType = contentType;
Response.Write(body);
Response.End();
```

And then the final part, of letting the DataGrid know that you have your own server url.

To do this, just set `grid.exportOptions.echoUrl=yourUrl;`

## Export with Server Paging

### **Additional information When filterPageSortMode=server:**

When filterPageSortMode=server, print/export works a little differently when you select “all pages” or “selected pages”. In this case, you need to wire up the PrintExportDataRequest event

```
//This event is only fired when the user requests to print data that is
currently
    //not loaded in the grid. It is only applicable when
filterPageSortMode is set to
    //'server', because in this mode, the grid only requests the
current page of data
    //from the server.
    //For example, if the user asks to print pages 1-4 of a 10
    //page grid, and we're currently on page #4, on the client, we
only have the records
    //from page #4. So we have to get the records from pages 1-4 from
the server.
    //when this data comes back, we just set the grids.printExportData
property, and the
    //print/export mechanism continues as usual.
private function
onPrintExportDataRequest(event:PrintExportDataRequestEvent):void
{
    //here we build our custom filter object and send it to the
server
    ...

    //Function to call when server call returns.
private function onPrintExportDataResponse(event:ResultEvent):void
{
    //This response came back as a result of a print request. so set
    // the print export data on the grid and have the print/export
    //mechanism continue on its way
    dgEmployeesServer.printExportData=event.result.records;
    ...
}
```

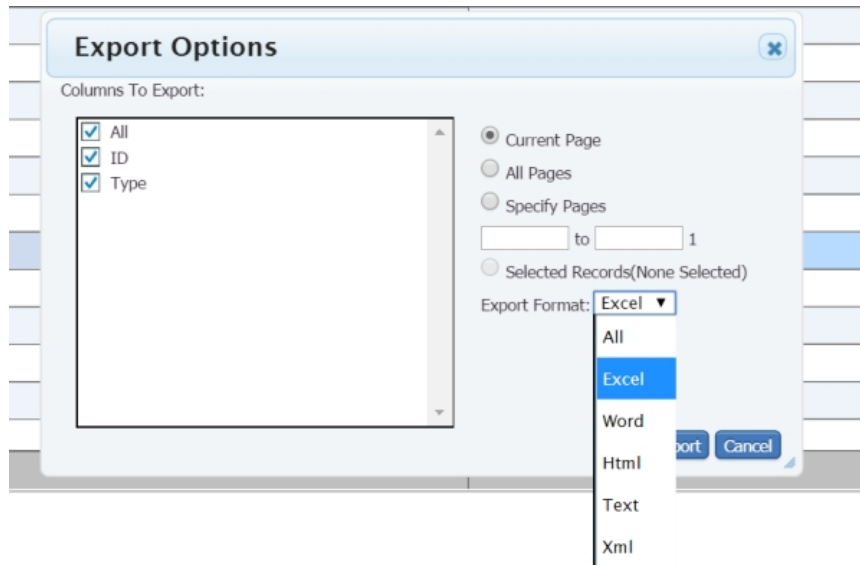
Review the code here:

<http://www.flexicious.com/resources/Flex/srcview/source/com/sample/examples/ServerGridDotNet.mxml.html>

This code demonstrates how to handle the printExportDataRequest event, and is heavily commented to explain what is going on.

## Custom Export Popup

Often times, you want to customize the popup that shows up when you click on the export button.



Like we saw in the renderers topic, the `grid.popupFactoryExportOptions` gives you a class factory that should return an instance of `ExportOptionsView`. This can be your own class.

```
grid.popupFactoryExportOptions = new
flexiciousNmsp.ClassFactory(flexiciousNmsp.CustomExportOptionsView);
```

The `CustomExportOptionsView` is a class that renders the popup for the export.

A running example of how this can be done is demonstrated in this blog post: [http://blog.htmltreegrid.com/post/How-to-Customize-the-Pager-\(Toolbar\)-Export-Options-Settings-and-Save-Settings-Popups.aspx](http://blog.htmltreegrid.com/post/How-to-Customize-the-Pager-(Toolbar)-Export-Options-Settings-and-Save-Settings-Popups.aspx)

## Custom Exporter

Since each of the built in exporters simply inherits from the base Exporter class and implements its own "customized" way of exporting data, it is quite easy to actually plugin your own exporters. Let's take a look at how this is done.

First, we need to write our own custom exporter. This is actually quite simple. In this example, we will use an exporter that writes out OO XML (Open Office Excel Markup Language). This is XML that Microsoft Excel 2007 and up can open. This allows you to write out documents that are excel compatible, and allow you to add formatting, formulas and more. It saves an xml file, but this file can be opened in excel without the dreaded "The file you are trying to open is in a different format than specified by the file extension" message. The big upside of the excel 2007 exporter is that it lets you achieve a combination of no extension message, PLUS the ability to define styles and fonts. Finally, double clicking on this file will automatically open excel on systems that have Office 2007 and above installed

```
/**
 * Flexicious
 * Copyright 2011, Flexicious LLC
 */
(function(window)
{
    "use strict";
    var Excel2007Exporter, uiUtil = flexiciousNmsp.UIUtils, flxConstants =
flexiciousNmsp.Constants;
    /**
     * Exports the grid in CSV format
     * @constructor
     * @namespace
     * @extends Exporter
     */
    Excel2007Exporter=function(){

        /**
         * Writes the header of the grid (columns) in csv format
         * @param grid
         * @return
         */

        this.strTable = "";

    };
    flexiciousNmsp.Excel2007Exporter = Excel2007Exporter; //add to name space
    Excel2007Exporter.prototype = new flexiciousNmsp.Exporter(); //setup hierarchy
    Excel2007Exporter.prototype.typeName = Excel2007Exporter.typeName =
'Excel2007Exporter';//for quick inspection
    Excel2007Exporter.prototype.getClassName=function(){
        return ["Excel2007Exporter", "Exporter"];
    };
    Excel2007Exporter.prototype.writeHeader=function(grid){

        this.buildHeader(grid);
        return "";
    };

```

```

};
/**
 * @private
 * @param grid
 * @return
 *
 */
Excel2007Exporter.prototype.buildHeader=function (grid){

    var colIndex=0;

    this.strTable += "<Row ss:StyleID='s68'>";
    while(colIndex++<this.nestDepth)
        this.strTable += "<Cell><Data ss:Type='String'></Data></Cell>";
    for(var i=0;i<grid.getExportableColumns().length;i++){
        var col=grid.getExportableColumns()[i];
        if(!this.isIncludedInExport(col))
            continue;

        this.strTable += "<Cell><Data ss:Type='String'>" +
            flexiciousNmsp.Exporter.getColumnHeader(col,colIndex) + "</Data></Cell>";
        colIndex++;
    }
    this.strTable += "</Row>";

};
Excel2007Exporter.prototype.uploadForEcho=function(body, exportOptions){
    var strWorkbook = "<?xml version='1.0' encoding='ISO-8859-1'?><?mso-application
progid='Excel.Sheet'?>" +
        "<Workbook xmlns='urn:schemas-microsoft-com:office:spreadsheet' " +
        "xmlns:ss='urn:schemas-microsoft-com:office:spreadsheet'
xmlns:x='urn:schemas-microsoft-com:office:excel' " +
        "xmlns:o='urn:schemas-microsoft-com:office:office' xmlns:html='http://
www.w3.org/TR/REC-html40'>" +
        "<Styles>" +
        "<Style ss:Name='Normal' ss:ID='Default'>" +
        "<Alignment ss:Vertical='Bottom' />" +
        "<Borders />" +
        "<Font />" +
        "<Interior />" +
        "<NumberFormat />" +
        "<Protection />" +
        "</Style>" +
        "<Style ss:ID='s68'>" +
        "<Borders>" +
        "<Border ss:Position='Bottom' ss:LineStyle='Continuous' ss:Weight='2'/>" +
        "<Border ss:Position='Top' ss:LineStyle='Continuous' ss:Weight='2'/>" +
        "</Borders>" +
        "<Interior ss:Color='#4F81BD' ss:Pattern='Solid'/>" +
        "</Style>" +
        "</Styles>" +
        "<Worksheet ss:Name='Sheet1'>" +
        "<Table>";
    strWorkbook += this.strTable + "</Table></Worksheet></Workbook>";
    flexiciousNmsp.Exporter.prototype.uploadForEcho.apply(this,[strWorkbook,
this.exportOptions]);
    this.strTable="";
};
/**
 * Writes an individual record in csv format
 * @param grid
 * @param record

```

```

* @return
*
*/
Excel2007Exporter.prototype.writeRecord=function(grid, record){

    var colIndex=0;
    this.strTable += "<Row>";
    if(!this.reusePreviousLevelColumns){
        while(colIndex++<this.nestDepth)
            this.strTable += "<Cell><Data ss:Type='String'></Data></Cell>";
    }
    for(var i=0;i<grid.getExportableColumns().length;i++){
        var col=grid.getExportableColumns()[i];
        if(!this.isIncludedInExport(col))
            continue;
        var str = col.itemToLabel(record);
        this.strTable += "<Cell><Data ss:Type='String'>" + str + "</Data></Cell>";
    }
    this.strTable += "</Row>";
    return "";
};

/**
 * Writes the footer in CSV format
 * @param grid
 * @param dataProvider
 */
Excel2007Exporter.prototype.writeFooter=function(grid, dataProvider){

    return "";

};

/**
 * Extension of the download file.
 * @return
 */
Excel2007Exporter.prototype.getExtension = function() {
    return "xml";
};

/**
 * Returns the content type so MS Excel launches
 * when the exporter is run.
 * @return
 */
Excel2007Exporter.prototype.getContentType = function() {
    return "text/xml"
};

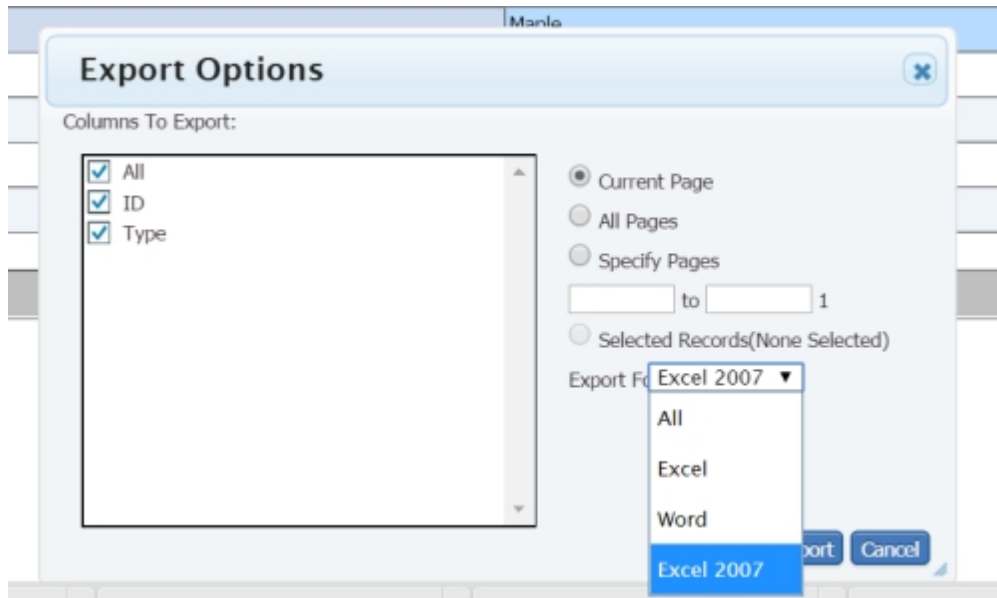
/**
 * Name of the exporter
 * @return
 */
Excel2007Exporter.prototype.getName = function() {
    return "Excel 2007";
};
}(window));

```

And the way you wire this up is :

```
grid.excelOptions.exporters = exporters = [new  
flexiciousNmsp.CsvExporter(), new flexiciousNmsp.DocExporter(),  
new flexiciousNmsp.Excel2007Exporter()  
];
```

Once you click on the excel export button, you should see:



Once you do the export using Excel 2007 exporter, you should see the below:



File Home Insert Page Layout Formulas Data Review View Load Test Team										
Clipboard		Font			Alignment			General		
Paste		Arial 10			Wrap Text			\$ %		
Cut Copy		B I U			Merge & Center					
Format Painter										
I11										
	A	B	C	D	E	F	G	H	I	J
1	ID	Legal Nam	Address -	Address -	Address -	Address -	Address -	Financials	Financials	Financials
2	20800	3M Co	597 Newar	Suite #352	Stroudsbui	New Jerse	United Sta	29,045.00	43,816.00	4.47
3		Deal Desc	Deal Amou	Deal Date						
4		Project #	1	625,555.0	Dec-27-2005					
5		Invoice Num	Invoice Arr	Invoice Sta	Invoice Dal	Due Date				
6		2080000	144,241.0	Approved	Apr-26-201	May-26-2010				
7		Line Item [ Line Item Amount								
8			Profession	18,538.00						
9			Profession	31,617.00						
10			Profession	27,537.00						
11			Profession	44,272.00						
12			Profession	22,277.00						
13		2080001	123,770.0	Approved	Nov-04-20	Dec-04-2009				
14		Line Item [ Line Item Amount								
15			Profession	18,754.00						
16			Profession	38,325.00						
17			Profession	35,211.00						
18			Profession	14,047.00						
19			Profession	17,433.00						
20		2080002	113,069.0	Cancelled	Jul-05-200	Aug-04-2008				
21		Line Item [ Line Item Amount								
22			Profession	11,226.00						
23			Profession	12,745.00						
24			Profession	40,708.00						

For a running example of this functionality, please review this blog post :

<http://blog.htmltreegrid.com/post/Customizing-Excel-Output.aspx>

## Preference Persistence

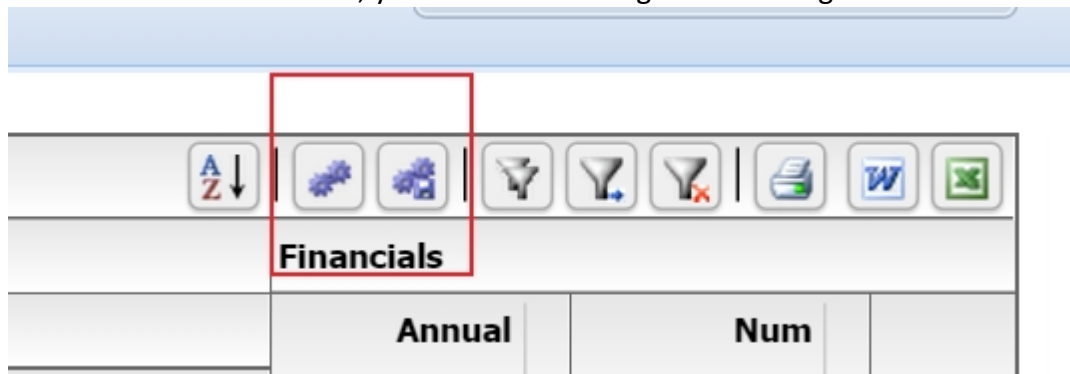
### Introduction

Preference Persistence is the feature of "maintaining state" for the grid between page reloads. The state includes the following :

- `PERSIST_COLUMN_ORDER:String="columnOrder"` : The order in which columns appear. User can drag and drop columns and then save the state. When the grid is loaded in the future, the order of the columns will be preserved.
- `PERSIST_COLUMN_LOCKMODES:String="columnLockModes"`: The lock modes of the column - left, right or unlocked.
- `PERSIST_COLUMN_VISIBILITY:String="columnVisiblity"` : Whether or not the column is hidden.
- `PERSIST_COLUMN_WIDTH:String="columnWidth"`: The width of the columns.
- `PERSIST_FILTER:String="filter"`: The filter values that the user currently has entered into the filter controls.
- `PERSIST_SORT:String="sort"` : The sort settings.
- `PERSIST_VERTICAL_SCROLL:String="verticalScroll"`: The vertical scroll position of the grid.
- `PERSIST_HORIZONTAL_SCROLL:String="horizontalScroll"`: The horizontal scroll position of the grid.
- `PERSIST_FOOTER_FILTER_VISIBILITY:String="footerFilterVisiblity"` : Whether or not the user has chosen to show or hide the footer and filter bars.
- `PERSIST_PAGE_SIZE:String="pageSize"` : The page size that the user has chosen.
- `PERSIST_PRINT_SETTINGS:String="printSettings"`: Print settings, currently not applicable for HTML version of grid.

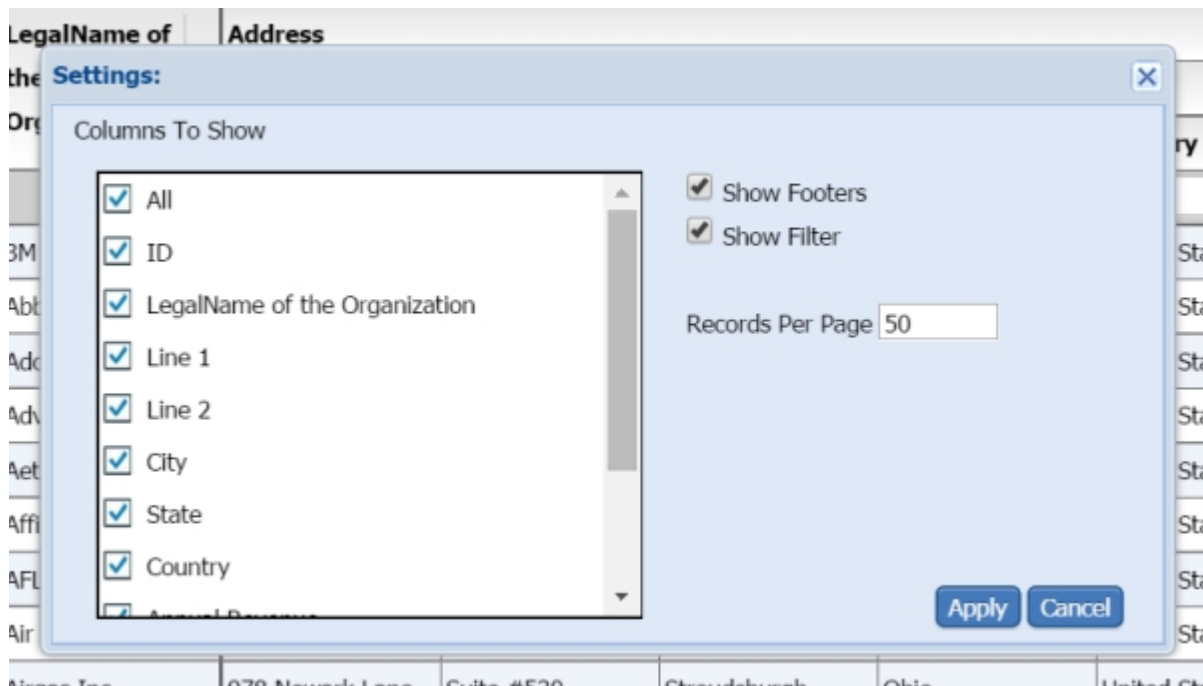
### Built in features

If you set `enablePreferences=true`, you see the following icons in the grid:

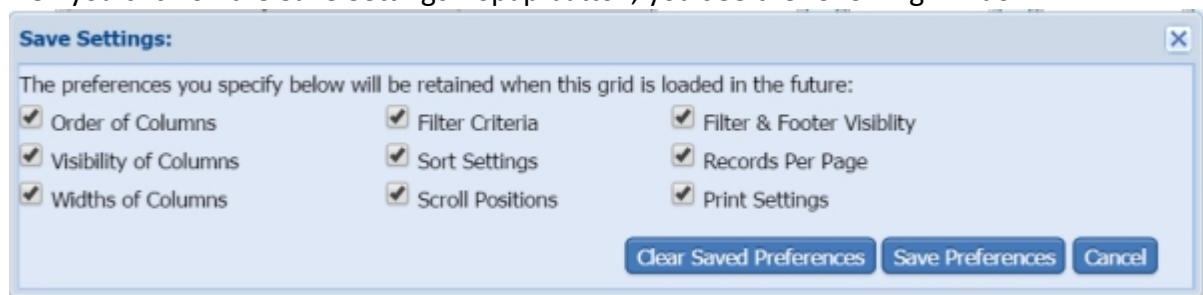


These are the settings and save settings pop-ups.

When you click on the Settings Popup button, you see the below popup.



When you click on the Save Settings Popup button, you see the following window:



Together, these two windows allow you to configure the grid, as well as save the state of the grid.

With client preferences, these are stored in local storage on the browser. For server based preference persistence, please refer to the next chapter.

## Persisting Preferences on the Server

When you save preferences, where are the preferences persisted to??

The grid has a robust persistence preference mechanism that "just works" out of the box, but the preferences are stored on the client machine's localStorage by default. This enables the preference persistence mechanism to work without any additional coding on the part of the developers utilizing the library. While this may be sufficient for most people, this will not work in scenarios when the same user uses multiple machines to access your application, or if multiple users access the application using the same machine. For this, you will need to store preferences in the backend, and this post aims to show you how this is done. Attached is a simple example that you can plug into the demo console you got when you downloaded a trial (or purchased a license).

This depends on `preferencePersistenceMode`. String value "server" or "client". When this property is set to client(default), the grid uses a local storage on the client to store preference settings. When it is set to server, the grid fires an event, `preferencesChanged`, which contains a string representation of the preference values. This can then be persisted on the back-end, tied to a specific user id.

While you are at it, also review `preferencePersistenceKey`:

String value that uniquely identifies this grid across the application. If you have multiple grids' with `enablePreferencePersistence`, and they happen to share the same value for the id field, e.g. `id="grid1"` they might overwrite each others' preferences. To combat this situation, we provide a property, which defaults to the id of the grid, but you can override to provide a globally unique key.

Saving preferences on the server:

If you look at `ServerPreferences.js` it shows the JS code require to enable server based persistence of preferences: Please note the declaration of the grid - we wire up 3 events, `loadPreferences`, `persistPreferences`, and `clearPreferences`. Each of these events call a service method, which we will talk about in a bit. But the idea is simple, on `persistPreferences`, we send up the preferences string, the name of the grid, and in your service method, you figure out who the logged in user is (on basis of your security implementation) and store a record in the database with `UserID`, `gridName`, and preferences (Make this a large text field (`nvarchar(max)` in SQL Server Lingo), it could get pretty verbose). the `loadPreferences` will call your service method to load the preferences you persisted, and set the grids preferences property. Once you set the property, the grid will parse everything and get back to the state it was when the preferences were stored. And then the final method is the `clearPreferences`, which will basically just wipe out the preferences that were previously stored. Sound like a lot, but really pretty straight forward. In this example, we just store the preferences in a global variable instead of the database.

Lets take a look at the code that does this:

```
var myCompanyNameSpace = {};
myCompanyNameSpace.SAMPLE_CONFIGS = {};
myCompanyNameSpace.SAMPLE_CONFIGS["ServerPreferences"] = '<grid
```

```

creationComplete="myCompanyNameSpace.serverPreferences_CreationComplete"
id="grid" enablePrint="true" preferencePersistenceMode="server"
enablePreferencePersistence="true" enableExport="true" forcePagerRow="true"
pageSize="50" enableFilters="true" enableFooters="true" >' +
    '
        <level>' +
        '
            <columns>' +
            '
                <column dataField="id" headerText="ID" />' +
            '
                <column dataField="type" headerText="Type"/>' +
            '
                <column dataField="" headerText="" />' +
            '
            </columns>' +
        '
        </level>' +
    '
    ' +
' </grid>';

myCompanyNameSpace.Preferences = {}
myCompanyNameSpace.Preferences.save = function (str, fn) {
    $.post('Preferences/set', { str: str }, function (o) {
        fn(o);
    });
}

myCompanyNameSpace.Preferences.load = function (fn) {
    $.get('Preferences/get', function (o) {
        fn(o);
    });
}

myCompanyNameSpace.serverPreferences_CreationComplete = function (event) {
    var grid = event.target;
    grid.setDataProvider([
        { "id": "5001", "type": "None" },
        { "id": "5002", "type": "Glazed" },
        { "id": "5005", "type": "Sugar" },
        { "id": "5007", "type": "Powdered Sugar" },
        { "id": "5006", "type": "Chocolate with Sprinkles" },
        { "id": "5003", "type": "Chocolate" },
        { "id": "5004", "type": "Maple" }
    ]);
    grid.addEventListener(this,
flexiciousNmsp.FlexDataGrid.EVENT_LOADPREFERENCES,
    //Called when the grid needs preferences (on creation complete)
    function () { //onLoadPreferences()
        //here you will call a server method to load preferences, for this
        demo, we are going to mock this.
        flexiciousNmsp.UIUtils.showMessage("Loading Preferences!");
        myCompanyNameSpace.Preferences.load(function (o) {
            if (o) {
                flexiciousNmsp.UIUtils.showMessage("Preferences found,
applying!");
                grid.setPreferences(o);
            } else flexiciousNmsp.UIUtils.showMessage("No preferences
found, please save preferences!");
        });
    }
);
    grid.addEventListener(this,
flexiciousNmsp.FlexDataGrid.EVENT_PERSISTPREFERENCES,
    //Called when the user clicks on "save preferences" in the preferences

```

```

dialog box
    function () { //onPersistPreferences()
        //here you will call a server method to save preferences, for this
        demo, we are going to mock this.
        flexiciousNmsp.UIUtils.showMessage("Saving Preferences!");
        myCompanyNameSpace.Preferences.save(grid.getPreferences(),
function (o) {
    flexiciousNmsp.UIUtils.showMessage("Preferences Saved!");
    })
    }
    );

    //Called when the user clicks on "clear preferences" in the preferences
    dialog box
    grid.addEventListener(this,
flexiciousNmsp.FlexDataGrid.EVENT_CLEARPREFERENCES,
    function () { //onClearPreferences()
        //here you will call a server method to clear preferences, for
        this demo, we are going to mock this.
        flexiciousNmsp.UIUtils.showMessage("Clearing Preferences!");
        myCompanyNameSpace.Preferences.save("", function (o) {
            flexiciousNmsp.UIUtils.showMessage("Preferences Cleared!");
        })
    }
    );
    grid.loadPreferences();
};

$(document).ready(function () {
    var grid = new
    flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
    {
        configuration:
    myCompanyNameSpace.SAMPLE_CONFIGS["ServerPreferences"]
    });
});

```

Basically, what we do here is that we wire up three event listeners:

- EVENT\_LOADPREFERENCES
- EVENT\_PERSISTPREFERENCES
- EVENT\_CLEARPREFERENCES

Each of these event handlers, we go back to the server, passing in the relevant information and loading the preferences from the server. The data structure to store preferences looks like this

(MY-SQL):

delimiter \$\$

```

CREATE TABLE `gridpreference` (
  `idGridPreference` int(11) NOT NULL,
  `gridName` varchar(254) COLLATE utf8_unicode_ci DEFAULT NULL,
  `idUser` varchar(254) COLLATE utf8_unicode_ci DEFAULT NULL,

```

```

`createdBy` varchar(254) COLLATE utf8_unicode_ci DEFAULT NULL,
`createdDate` datetime DEFAULT NULL,
`updatedBy` varchar(254) COLLATE utf8_unicode_ci DEFAULT NULL,
`updatedDate` datetime DEFAULT NULL,
`preferenceString` text COLLATE utf8_unicode_ci,
PRIMARY KEY (`idGridPreference`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci$$

```

SQL Server:

```

USE [Flexicious]
GO

```

```

/***** Object: Table [dbo].[Preferences]      Script Date: 11/27/2014 19:11:57
*****/

```

```

SET ANSI_NULLS ON
GO

```

```

SET QUOTED_IDENTIFIER ON
GO

```

```

CREATE TABLE [dbo].[Preferences] (
    [PreferenceId] [int] NOT NULL,
    [GridName] [nvarchar] (50) NOT NULL,
    [PreferenceString] [nvarchar] (max) NOT NULL,
    [UserId] [nvarchar] (255) NOT NULL
) ON [PRIMARY]

```

```

GO

```

And basically on the back-end, we have code that persists the preferences to the server.

Sample C# Code:

```

/// <summary>
/// Returns previously saved preferences for the provided
/// Grid Name. This name corresponds to the preferencePersistenceKey
/// from Flex
/// </summary>
/// <param name="gridName">The Grid To get preferences for</param>
/// <returns>The persisted preference values</returns>
public ActionResult GetSavedPreferences(string gridName)
{
    //Check the database if we have previously persisted this
    preference.
    Preferences pref = entites.Preferences.Where(p =>
    p.GridName.Equals(gridName) &&
    p.UserId.Equals(SESSION_USER_ID)).FirstOrDefault();
    if (pref != null)
        return Content(pref.PreferenceString); //if we have
    preferences. return them
    else
        return Content(""); //else return a blank string
}
/// <summary>
/// Persists the preferences for the provided grid name
/// </summary>
/// <param name="gridName">Grid to persist the preferences for</param>

```

```

    /// <param name="prefXml">The preferences to persist</param>
    /// <returns>Success code.</returns>
    [ValidateInput(false)]
    public ActionResult PersistPreferences(string gridName, string
prefXml)
    {
        //Check the database if we have previously peristed this
preference.
        Preferences pref = entites.Preferences.Where(p =>
p.GridName.Equals(gridName) &&
p.UserId.Equals(SESSION_USER_ID)).FirstOrDefault();
        //If no, then create a new one..
        if (pref == null)
        {
            pref = new Preferences();
            entites.AddToPreferences(pref);
        }
        //Set the preference information
        pref.GridName = gridName;
        pref.PreferenceString = prefXml;
        //Here we're tying the preferences to your IP, but in your code,
        //you would tie it to your USER ID
        pref.UserID = SESSION_USER_ID;

        //save the preferences to the database

        entites.SaveChanges();
        return Content("Preferences Persisted!");
    }
    /// <summary>
    /// Clears previously saved preferences for the provided
    /// Grid Name. This name corresponds to the preferencePersistenceKey
    /// from Flex
    /// </summary>
    /// <param name="gridName">The Grid To clear preferences for</param>
    /// <returns>The persisted preference values</returns>
    public ActionResult ClearPreferences(string gridName)
    {
        //Check the database if we have previously peristed this
preference.
        Preferences pref = entites.Preferences.Where(p =>
p.GridName.Equals(gridName) &&
p.UserId.Equals(SESSION_USER_ID)).FirstOrDefault();
        if (pref != null)
        {
            pref.PreferenceString = string.Empty; //we could alternatively
delete the record..
            entites.AddToPreferences(pref);
            entites.SaveChanges();
        }
        return Content("Preferences Removed!");
    }
}

```

Java Code:

```

    public static void createGridPreference(String gridName, String prefXml,
String idUser)
        throws OSException {

```



```

Connection c = null;
PreparedStatement ps = null;

int id = -1;
long now = System.currentTimeMillis();

try {

    c = createDBConnection();
    ps = prepareStatement(c, "INSERT INTO GridPreference
(idGridPreference, gridName, idUser, createdBy, createdDate, updatedBy,
updatedDate, preferenceString) values (?, ?, ?, ?, ?, ?, ?, ?)");

    int i = 1;

    ps.setInt(i++,
SQLSequencer.getNextValue(SQLSequencer.GRID_PREFERENCE_TABLE_CODE));
    ps.setString(i++, gridName);
    ps.setString(i++, GridPreference.ALL_USERS);
    ps.setString(i++, idUser);
    ps.setTimestamp(i++, new
java.sql.Timestamp(SettingManager.offsetTime(now)));
    ps.setString(i++, idUser);
    ps.setTimestamp(i++, new
java.sql.Timestamp(SettingManager.offsetTime(now)));
    ps.setString(i++, prefXml);

    ps.executeUpdate();

} catch (SQLException x) {
    throw new OSEException("Unable to create new grid preference.",
        x);
} finally {
    try {ps.close();} catch (Exception ignored) {}
    try {c.close();} catch (Exception ignored) {}
}

}

public static void updateGridPreferences(String gridName, String prefXml,
String idUser)
    throws OSEException {

    Connection c = null;
    PreparedStatement ps = null;

    int id = -1;
    long now = System.currentTimeMillis();

    try {

        c = createDBConnection();

        ps = prepareStatement(c, "UPDATE GridPreference SET updatedBy=?,
updatedDate=?, preferenceString=? WHERE idUser=? AND gridName=? ");

```

```

        int i = 1;
        ps.setString(i++, idUser);
        ps.setTimestamp(i++, new
java.sql.Timestamp(SettingManager.offsetTime(now)));
        ps.setString(i++, prefXml);
        ps.setString(i++, GridPreference.ALL_USERS);
        ps.setString(i++, gridName);

        int numberUpdated = ps.executeUpdate();

        if (numberUpdated < 1) {
            throw new OSEException("No grid preference was updated!  name:
" + gridName + " does not exist under user: " + idUser + ".");
        } else if (numberUpdated > 1) {
            throw new OSEException("Too many grid preferences update!
name: " + gridName + " belongs to more than 1 grid.");
        }

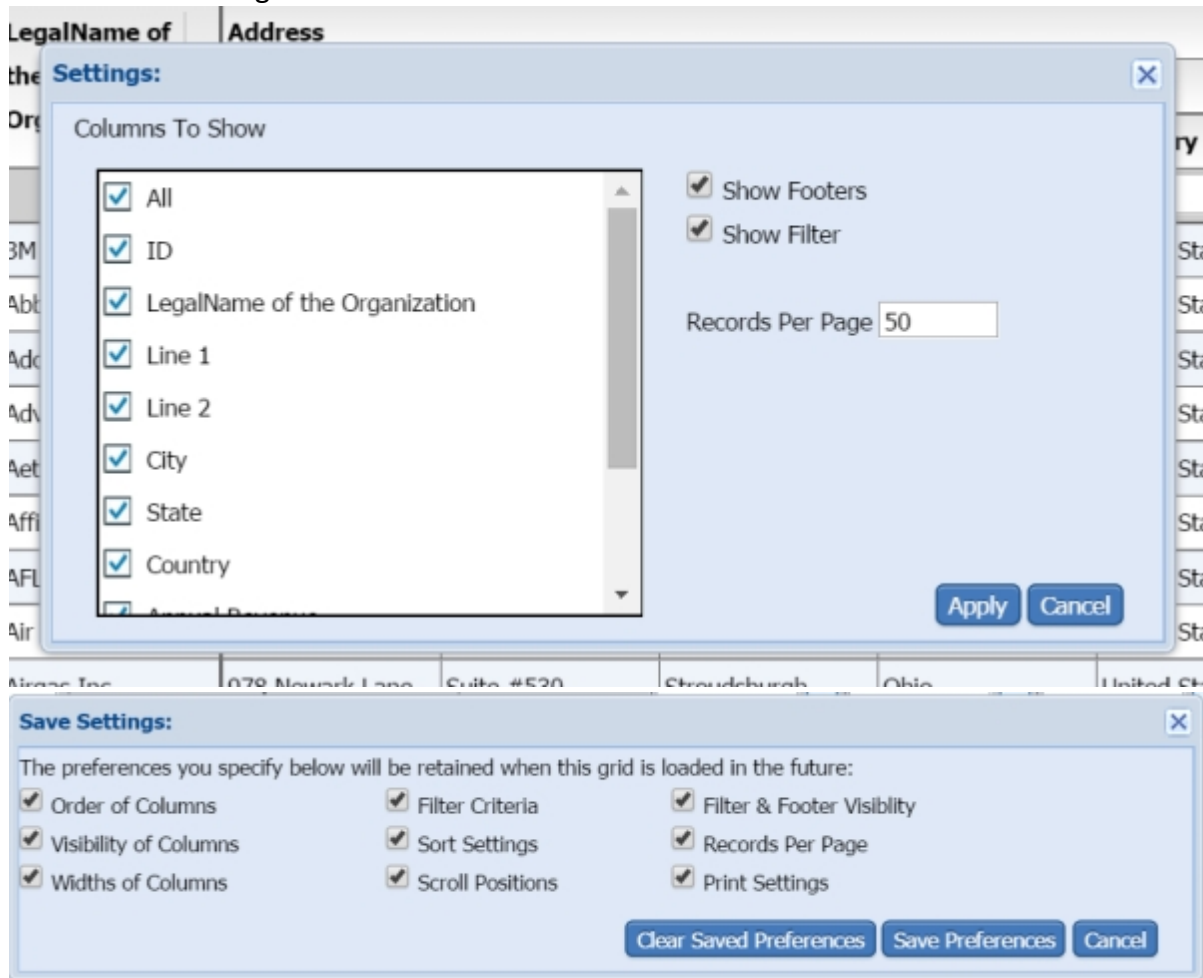
    } catch (SQLException x) {
        throw new OSEException("Unable to update grid preferences.",
            x);
    } finally {
        try {ps.close();} catch (Exception ignored) {}
        try {c.close();} catch (Exception ignored) {}
    }
}

```

For a running demo, please refer <http://blog.htmltreegrid.com/post/Adding-Preferences-Support.aspx>

## Customizing the Preference Popups

Often times, you want to customize the popup that shows up when you click on the Save Preferences or Settings Button.



Like we saw in the renderers topic, the `grid.popupFactoryExportOptions` gives you a class factory that should return an instance of `ExportOptionsView`. This can be your own class.

```
grid.popupFactorySaveSettingsPopup = new
flexiciousNmsp.ClassFactory(flexiciousNmsp.CustomSaveSettingsPopup);
grid.popupFactorySettingsPopup = new
flexiciousNmsp.ClassFactory(flexiciousNmsp.CustomSettingsPopup);
```

The `CustomSaveSettingsPopup` and `CustomSettingsPopup` are a classes that renders the popup for the settings.

A running example of how this can be done is demonstrated in this blog post: [http://blog.htmltreegrid.com/post/How-to-Customize-the-Pager-\(Toolbar\)-Export-Options-Settings-and-Save-Settings-Popups.aspx](http://blog.htmltreegrid.com/post/How-to-Customize-the-Pager-(Toolbar)-Export-Options-Settings-and-Save-Settings-Popups.aspx)

## Hierarchical Grids

As discussed elsewhere in this guide, display of hierarchical data is among the most powerful features of the HTMLTreeGrid. There are two different types of hierarchical grids:

- Nested (Multiple sets of columns, each at one hierarchical level, or grid within grid) display
- Grouped (Tree with a single set of top level columns).

Nested:

Items 1 to 20 of 411. Page 1 of 21										Go to Page: 1										Edit X Delete																			
	<input type="checkbox"/>	ID	Legal Name				Annual Revenue				Num Employees				EPS				Stock Price																				
	<input type="checkbox"/>																																						
▼	<input type="checkbox"/>	20800	3M Co				53,985				42,810				1.44				13.33																				
	<input type="checkbox"/>	Deal Description					Deal Amount					Deal Date																											
▼	<input type="checkbox"/>	Project # 1 - 3M Co - 6/2013					138,573					Jun-26-2013																											
	<input type="checkbox"/>	Invoice Number			Invoice Amount			Invoice Status			Invoice Date			Due Date																									
▼	<input type="checkbox"/>	2080000			45,255			Paid			Feb-04-2014			Mar-06-2014																									
	<input type="checkbox"/>	Line Item Description							Line Item Amount																														
	<input type="checkbox"/>	Professional Services - Jason Bourne														20,949																							
	<input type="checkbox"/>	Professional Services - Tarah Silverman														24,306																							
		Count:2.00														Total:\$45,255																							
▼	<input type="checkbox"/>	2080001			93,318			Transmitted			Oct-01-2012			Oct-31-2012																									
	<input type="checkbox"/>	Line Item Description							Line Item Amount																														
	<input type="checkbox"/>	Professional Services - Jason Bourne														47,413																							
	<input type="checkbox"/>	Professional Services - Kristian Donovan														45,905																							
		Count:2.00														Total:\$93,318																							
		Count:2.00			Total:\$138,573																																		
Items 1 to 2 of 2. Page 1 of 1																				Go to Page: 1										Edit X Delete									

Grouped:

Items 1 to 20 of 411. Page 1 of 21									Go to Page: 1 ▼			
<input type="checkbox"/>	Name	Amount	Invoice Number	Invoice Status	Invoice Date	Due Date						
				All ▼	All ▼	All ▼						
▼	<input type="checkbox"/> 3M Co	242,314										
▼	<input type="checkbox"/> Project # 1 - 3M Co - 6/2013	138,573										
	<input type="checkbox"/> 2080000	45,255		Paid	Feb-04-2014	Mar-06-2014						
	<input type="checkbox"/> 2080001	93,318		Transmitted	Oct-01-2012	Oct-31-2012						
		Total:\$138,573	Count:2.00									
Items 1 to 2 of 2. Page 1 of 1									Go to Page: 1 ▼			
▼	<input type="checkbox"/> Project # 2 - 3M Co - 6/2013	103,741										
	<input type="checkbox"/> 2080010	74,248		Approved	Oct-06-2013	Nov-05-2013						
	<input type="checkbox"/> 2080011	29,493		Paid	Jul-04-2012	Aug-03-2012						
		Total:\$103,741	Count:2.00									
Items 1 to 2 of 2. Page 1 of 1									Go to Page: 1 ▼			
▼	<input type="checkbox"/> AFLAC Inc	225,609										
▼	<input type="checkbox"/> Project # 1 - AFLAC Inc - 7/2014	115,732										
	<input type="checkbox"/> 2080600	48,234		Draft	Jun-20-2014	Jul-20-2014						
	<input type="checkbox"/> 2080601	67,498		Paid	Jun-02-2014	Jul-02-2014						
		Total:\$115,732	Count:2.00									
Items 1 to 2 of 2. Page 1 of 1									Go to Page: 1 ▼			

One of the most important concepts behind the Architecture of the grid arose from the fundamental requirement that we created the product for - that is display of Hierarchical Data. The notion of nested levels is baked in to the grid via the "columnLevel" property. This is a property of type "FlexDataGridColumnLevel". This grid always has at least one column level.

This is also referred to as the top level, or the root level. In flat grids (non hierarchical), this is the only level. But in nested grids, you could have any number of nested levels. The columns collection actually belongs to the columnLevel, and since there is one root level, the columns collection of the grid basically points to the columns collection of this root level. So if there is a grid like the Nested example above, where the top level shows Organizations, the next one shows Projects, and the next one Invoices, and the final inner most shows Invoice Line Items, we are basically looking at a grid with 4 levels. Below is the configuration for a grid like this one.

```
myCompanyNameSpace.SAMPLE_CONFIGS["Nested"]='<grid id="grid"
enablePrint="true" enableMultiColumnSort="true"'+'
    '+'
    <level enableFilters="true" enablePaging="true"
initialSortField="legalName"'+'
    '+'
    pageSize="20"
childrenField="deals" enableFooters="true" selectedKeyField="id"'+'
    '+'
    '+'
    <columns>'+'
    ... '+'
    </columns>'+'
    <nextLevel>'+'
    <level childrenField="invoices"
enableFooters="true" selectedKeyField="id" nestIndent="30"'+'
    '+'
    initialSortField="dealDate" initialSortAscending="false"
parentField="customer">'+'
    <columns>'+'
    ... '+'
    </columns>'+'
    <nextLevel>'+'
    <level childrenField="lineItems"
enableFooters="true" enablePaging="true" pageSize="3"'+'
    '+'
    selectedKeyField="id" parentField="deal" nestIndent="30">'+'
    <columns>'+'
    ... '+'
    </columns>'+'
    <nextLevel>'+'
    <level
enableFooters="true" selectedKeyField="id" parentField="invoice"
nestIndent="30">'+'
    <columns>'+'
    ... '+'
    </columns>'+'
    </level>'+'
    <nextLevel>'+'
    </level>'+'
    <nextLevel>'+'
    </level>'+'
    <nextLevel>'+'
    </level>'+'
</grid>;'
```

On the other hand, for the "grouped" grids, the inner levels do not have a column of their own, instead using the top level columns using the `reusePreviousLevelColumns` property.

```

myCompanyNameSpace.SAMPLE_CONFIGS["GroupedData"]='<grid id="grid"
enablePrint="true" horizontalGridLines="true" ...'
|
|           <level enableFilters="true" enablePaging="true"
pageSize="20" childrenField="deals" selectedKeyField="id"'+
|
|           reusePreviousLevelColumns="true" >'+
|               <columns>'+
|               ...'+
|               </columns>'+
|               <nextLevel>'+
|                   <level childrenField="invoices"
selectedKeyField="id" reusePreviousLevelColumns="true" >'+
|                       '+
|                       <nextLevel>'+
|                           <level enableFooters="true"
enablePaging="true" pageSize="5"'+
|
|                           selectedKeyField="id"
reusePreviousLevelColumns="true">'+
|                               '+
|                               </level>'+
|                               </nextLevel>'+
|                               </level>'+
|                               </nextLevel>'+
|                               </level>'+
|                               '+
|                               </grid>';

```

There is one field that is crucial to provide on the level: The childrenField. This is The property of the parent level object, that identifies the children that should be displayed on the next level. This is only required if the collection is not a hierarchical view. Please note, in server mode, this property is also the "storage" for the lazy loaded children.

The FlexDataGridColumnLevel class has a "nextLevel" property, which is a pointer to another instance of the same class, or a "nextLevelRenderer" property, which is a reference to a ClassFactory the next level. Please note, currently, if you specify nextLevelRenderer, the nextLevel is ignored. This means, at the same level, you cannot have both a nested subgrid as well as a level renderer. Bottom line - use nextLevelRenderer only at the innermost level. For an example, refer to [http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=Level%20Renderers](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=Level%20Renderers) or [http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=Level%20Renderers2](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=Level%20Renderers2)

#### Sample configuration for a level renderer

```

myCompanyNameSpace.SAMPLE_CONFIGS["LevelRenderers2"]='<grid id="grid"
enablePrint="true" enableDrillDown="true"'+
|
|           enablePreferencePersistence="true"'+
|
|                                           enableExport="true"
enableCopy="true"'+
|
|           preferencePersistenceKey="levelRenderers2"
on'+flexiciousNmosp.Constants.EVENT_CREATION_COMPLETE
+'="myCompanyNameSpace.levelRenderers2_creationCompleteHandler">'+
|           <level enableFilters="true" enablePaging="true"

```

```

rendererHorizontalGridLines="true" ' +
    '         rendererVerticalGridLines="true" pageSize="20"
childrenField="deals" enableFooters="true" selectedKeyField="id" ' +
    '
nextLevelRenderer="myCompanyNameSpace.LevelRenderers2.NextLevelRenderer2"
levelRendererHeight="120">' +
    '         <columns>' +
    '             <column type="checkbox" />' +
    '             <column enableCellClickRowSelect="false"
columnWidthMode="fitToContent" selectable="true" dataField="id"
headerText="ID" filterControl="TextInput"/>' +
    '             <column truncateToFit="true"
enableCellClickRowSelect="false" columnWidthMode="fitToContent"
selectable="true" dataField="legalName" headerText="Legal Name"/>' +
    '             <column dataField="headquarterAddress.line1"
headerText="Address Line 1" footerLabel="Count:" footerOperation="count"/>' +
    '             <column dataField="headquarterAddress.line2"
headerText="Address Line 2"/>' +
    '             <column dataField="headquarterAddress.city.name"
headerText="City" filterControl="MultiSelectComboBox"
filterComboBoxBuildFromGrid="true" filterComboBoxWidth="150"/>' +
    '             <column
dataField="headquarterAddress.state.name" headerText="State"
filterControl="MultiSelectComboBox" filterComboBoxBuildFromGrid="true"
filterComboBoxWidth="150"/>' +
    '             <column
dataField="headquarterAddress.country.name" headerText="Country"
filterControl="MultiSelectComboBox" filterComboBoxBuildFromGrid="true"
filterComboBoxWidth="150"/>' +
    '         </columns>' +
    '     </level>' +
    ' </grid>';

```

This generates a grid like this:

Items 1 to 20 of 411. Page 1 of 21

<input type="checkbox"/>	ID	Legal Name	Address Line 1	Address Line 2	City	State
<input type="checkbox"/>	20800	3M Co	209 Newark St	Suite #689	Barrie	Penn
Organization Information						
Organization Name 3M Co		Sales Contact RYAN WHITE		Sales Contact Phone:732-575-2941		
Annual Revenue:3,499		EPS:6.57		Last Stock Price:12.75		
Employees:1911		Address:209 Newark St Suite #689 Suite #689, Barrie, Penn, United States				
<input type="checkbox"/>	20806	APLAC Inc	169 Newark Rd	Suite #600	Albany	New Jersey
Organization Information						
Organization Name APLAC Inc		Sales Contact ROSETTA LOPEZ		Sales Contact Phone:646-522-3082		
Annual Revenue:39,099		EPS:2.51		Last Stock Price:24.59		
Employees:44141		Address:169 Newark Rd Suite #600 Suite #600, Albany, New Jersey, United States				
<input type="checkbox"/>	20809	AK Steel Holding	904 Park Rd	Suite #371	Barrie	Penn
Organization Information						
Organization Name AK Steel Holding		Sales Contact HAROLD ANDERSON		Sales Contact Phone:732-99		
Annual Revenue:25,437		EPS:2.71		Last Stock Price:23.98		
Employees:31088		Address:904 Park Rd Suite #371 Suite #371, Barrie, Penn, United States				

There are also two different modes of loading hierarchical data.

- itemLoadMode=client (default) - This assumes the parent objects and child objects are all loaded in client memory upfront.
- itemLoadMode=server - This assumes only the top level items are loaded, and the grid will trigger an event that you will then listen for, and load children in a lazy load mechanism (or load on demand). This is more appropriate when there are very large datasets.

The itemLoadMode is not to be confused with [filterPageSortMode](#), which determines the top level records only.

When itemLoadMode is server, you may want to set childrenCountField.

A property on the object that identifies if the object has children. Only needed in itemLoadMode=server In lazy loaded hierarchical grids levels, the child level items are loaded when the user actually expands this level for the first time. In scenarios where it is known initially that there are children, set this property to the value of the object that identifies whether there are children. If this property is set, the expand collapse icon will be drawn only when the value of this property on the object returns an integer greater than zero. Otherwise, the level will always draw the expand collapse icon.

## Paging Options & Toolbar

---

Out of the box, the grid offers up a pager bar (or a tool bar) on top. This bar includes paging buttons, and on basis of the various enableXXX flags, shows other buttons to manage filters, footers, preferences, export, etc.

If you set enablePaging=true, the bar will show up. If you do not want paging, but still want the bar to show up, you can set forcePagerRow=true.

Often times, you may want to customize the appearance of the tool bar. This is quite easy to do. The grid has a pager renderer property. Renderers are discussed more in detail [here](#). The way you set pager renderers is below:

```
grid.setPagerRenderer(new
flexiciousNmsp.ClassFactory(flexiciousNmsp.CustomPagerControl));
```

You can see a running example of this here: [http://blog.htmltreegrid.com/post/How-to-Customize-the-Pager-\(Toolbar\)-Export-Options-Settings-and-Save-Settings-Popups.aspx](http://blog.htmltreegrid.com/post/How-to-Customize-the-Pager-(Toolbar)-Export-Options-Settings-and-Save-Settings-Popups.aspx)

Finally, there is a difference in server paging and client paging. We discuss more about the [filterPageSortMode](#) [here](#). Client paging is the default and will work without you having to do anything other than setting enablePaging=true. For server paging, you have to listen to the filterPageSortChange event and send a server request on basis of the filter object. A running example can be found here: [http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=Fully%20Lazy%20Loaded](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=Fully%20Lazy%20Loaded)

## Angular JS Integration

---

For those of you who do not know Angular, it is among the most promising JavaScript application development frameworks. It is sponsored by Google, and has been witnessing explosive growth in the Enterprise space, where most of our customers are. For our Flex customers moving to JavaScript, angular will be a breath of fresh air. In fact, the feature set provided by angular are so similar to the Flex features, you would be pleasantly surprised by the number of concepts that Flex had that also exist in Angular. Data Binding, MVVM, Templates, Form Validation, to name a few.



The purpose of this topic is not to go into details of Angular JS, but there are some excellent resources on line that do this. There is also a book written by Jeffrey Houser, Life After Flex which is a good read if you are a Flex Developer moving to Angular JS : <https://www.lifeafterflex.com/>

AngularJSForFlexDevelopers/ In this topic, we are going to focus on the work that we have been doing make use of Angular JS features to make the lives of our customers easier. For Flex developers, the concept of MXML within view is second nature. Angular directives bring this same power to HTML developers. Seeing is believing , so let's take a look at a quick example of what this means:

Let's quickly look at the steps involved:

- Ensure that AngularJs is included
- Add HTMLtreeGrid directive after angular-js. This is available here: [www.htmltreegrid.com/demo/flexicious/js/htmltreegrid-angular-directive.js](http://www.htmltreegrid.com/demo/flexicious/js/htmltreegrid-angular-directive.js)
- Create an Angular module that depends on treegrid module : `angular.module('appName', ['some dependency', 'other dependency', 'fdGrid', 'more and more'])`
- Create your controllers and templates and use the fd-grid as attribute like this `<div fd-grid="" ng-model="gridOptions" style="height: 400px; width: 100%;"></div>`
- Note that ng-model is required and must be not null, and fd-grid is also required but it has no value
- In your controller access your \$scope and set the what ever model you need and pass it to the grid see the example above added gridOptions
- The grid model should have dataProvider that contains array of data
- Each attribute that has multiple upper case (camel case) should be separated by ( - ) and started with xi like enableMultiColumnSort should be xienable-multi-column-sort
- Each tag name that has multiple upper case (camel case) should be separated by ( - ) - for example : nextLevel should be next-level
- For events and function callbacks : With angular, you can still follow the regular mechanism of providing namespaced function callbacks, but there is another option, should you choose to use it. You can define the callbacks on the scope itself. The scope becomes the delegate for the grid, and all callbacks are then called on the scope

There are two key advantages of using Angular directive for HTMLTreeGrid.

1. You get to use the excellent modularization of the grid markup, and keep it separate from scripting logic. This makes code organization a lot cleaner, since markup for the grid usually becomes quite verbose, and mixing it with callback functions, event handlers tends to get bulky.
2. You get to define callback functions on your scope, because the scope automatically becomes the delegate for the grid. So the functions need not be added to the flexicious namespace (or have a custom namespace - like myCompanyNameSpace used in many of our examples). For example, you can do something like `:$scope.onGridCreationComplete = function (evt),` and use it in the markup as `xicreation-complete="onGridCreationComplete"`, and the scope will be inspected for that function name first.

That said, lets take a look at an example:

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
  <title>Tree Grid Sample</title>

  <!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript"
    src="http://htmltreegrid.com/demo/external/js/adapters/jquery/jquery-
1.8.2.js"></script>
  <script type="text/javascript"
    src="http://htmltreegrid.com/demo/external/js/adapters/jquery/jquery-ui-
1.9.1.custom.min.js"></script>
  <script type="text/javascript"
    src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript"
    src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.watermarkinput.js"></script>
  <script type="text/javascript"
    src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.ui.menu.js"></script>
  <script type="text/javascript"
    src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.toaster.js"></script>
  <!--End-->

  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://htmltreegrid.com/demo/minified-compiled-
jquery.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/examples/js/
Configuration.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/themes.js"></script>

  <!--End-->

  <!--css imports-->
  <link rel="stylesheet" href="http://htmltreegrid.com/demo/flexicious/css/
flexicious.css" type="text/css"/>
  <link rel="stylesheet"
    href="http://htmltreegrid.com/demo/external/css/adapters/jquery/jquery-ui-
1.9.1.custom.min.css"
    type="text/css"/>
  <!--End-->

  <!--AngularJs -->
  <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/
angularjs/1.2.16/angular.min.js"></script>
  <script type="text/javascript" src="http://htmltreegrid.com/demo/
flexicious/js/htmltreegrid-angular-directive.js"></script>

  <!--End-->

  <script>

    angular.module('app', ['fdGrid'])
      .factory('localStorage',function(){
        return {};
      })
      .controller('myCtrl', function ($scope) {

        $scope.gridOptions = {

          dataProvider: [
            { "id": "5001", "type": "None" },

```

```

        { "id": "5002", "type": "Glazed" },
        { "id": "5005", "type": "Sugar" },
        { "id": "5007", "type": "Powdered Sugar" },
        { "id": "5006", "type": "Chocolate with Sprinkles" },
        { "id": "5003", "type": "Chocolate" },
        { "id": "5004", "type": "Maple" }
    ],
    delegate: $scope

};

$scope.onGridCreationComplete = function (evt) {
    var grid = evt.target;
    grid.validateNow();
    alert(grid.getDataProvider().length);
};

    })
</script>
</head>
<body>
<div ng-app="app">
    <div ng-controller="myCtrl">

        <!-- Tree Grid -->
        <div id="gridContainer" fd-grid="" ng-model="gridOptions" style="height:
400px;width: 100%;" xicreation-complete="onGridCreationComplete"
        xienable-Print="true" xienable-Preference-Persistence="true" xienable-
Export="true" xiforce-PagerRow="true"
        xipage-Size="50" xienable-Filters="true" xienable-Footers="true">
            <level>
                <columns>
                    <!-- Don't forget <column xidata-Field="id" xiheader-Text="ID"> will
not work -->
                    <column xidata-Field="id" xiheader-Text="ID"></column>
                    <column xidata-Field="type" xiheader-Text="Type"></column>
                </columns>
            </level>
        </div>

    </div>
</div>
</body>
</html>

```

And here is one with support for hierarchy:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Tree Grid Sample</title>

    <!--These are jquery and plugins that we use from jquery-->
    <script src="//www.parsecdn.com/js/parse-1.3.0.min.js"></script>

    <script type="text/javascript"
        src="http://htmltreegrid.com/demo/external/js/adapters/jquery/jquery-
1.8.2.js"></script>
    <script type="text/javascript"

```

```

        src="http://htmltreegrid.com/demo/external/js/adapters/jquery/jquery-ui-
1.9.1.custom.min.js"></script>
        <script type="text/javascript"
            src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.maskedinput-1.3.js"></script>
        <script type="text/javascript"
            src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.watermarkinput.js"></script>
        <script type="text/javascript"
            src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.ui.menu.js"></script>
        <script type="text/javascript"
            src="http://htmltreegrid.com/demo/external/js/adapters/jquery/
jquery.toaster.js"></script>
        <!--End-->

        <!--These are specific to htmltreegrid-->
        <script type="text/javascript" src="http://htmltreegrid.com/demo/minified-compiled-
jquery.js"></script>
        <script type="text/javascript" src="minified-compiled-jquery.js"></script>
        <script type="text/javascript" src="http://htmltreegrid.com/demo/examples/js/
Configuration.js"></script>
        <script type="text/javascript" src="http://htmltreegrid.com/demo/themes.js"></script>

        <!--End-->

        <!--css imports-->
        <link rel="stylesheet" href="http://htmltreegrid.com/demo/flexicious/css/
flexicious.css" type="text/css"/>
        <link rel="stylesheet"
            href="http://htmltreegrid.com/demo/external/css/adapters/jquery/jquery-ui-
1.9.1.custom.min.css"
            type="text/css"/>
        <!--End-->

        <!--AngularJs -->
        <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/
angularjs/1.2.16/angular.min.js"></script>
        <script type="text/javascript"
            src="http://htmltreegrid.com/demo/flexicious/js/htmltreegrid-angular-
directive.js"></script>

        <!--End-->

        <script>
            angular.module('app', ['fdGrid'])
                .factory('localStorage', function () {
                    return {};
                })
                .controller('myCtrl', function ($scope, gridService) {

                    var dummyServerPreferences = "";

                    $scope.gridOptions = {

                        dataProvider: [
                            { "id": "5001", "type": "None" },
                            { "id": "5002", "type": "Glazed", "sub": [
                                { "id": "1", "price": "10" }, { "id": "2", "price": "11" },
                                { "id": "3", "price": "9.5" }, { "id": "4", "price": "10" }
                            ] },
                            { "id": "5005", "type": "Sugar", "sub":

```

```

[{"id": "1", "price": "10"}, {"id": "2", "price": "11"}, {"id": "3", "price": "9.5"},
{"id": "4", "price": "10"}]],
    { "id": "5007", "type": "Powdered Sugar", "sub":
[{"id": "1", "price": "10"}, {"id": "2", "price": "11"}, {"id": "3", "price": "9.5"},
{"id": "4", "price": "10"}]],
    { "id": "5006", "type": "Chocolate with Sprinkles", "sub":
[{"id": "1", "price": "10"}, {"id": "2", "price": "11"}, {"id": "3", "price": "9.5"},
{"id": "4", "price": "10"}]],
    { "id": "5003", "type": "Chocolate", "sub":
[{"id": "1", "price": "10"}, {"id": "2", "price": "11"}, {"id": "3", "price": "9.5"},
{"id": "4", "price": "10"}]],
    { "id": "5004", "type": "Maple", "sub":
[{"id": "1", "price": "10"}, {"id": "2", "price": "11"}, {"id": "3", "price": "9.5"},
{"id": "4", "price": "10"}]]
    ],
    delegate: $scope
};

$scope.onGridCreationComplete = function (event) {
    var grid = event.target;
    grid.validateNow();
};

    })
</script>
</head>
<body>
<div ng-app="app">

    <div ng-controller="myCtrl">

        Tree Grid
        <div id="gridContainer" fd-grid="" ng-model="gridOptions" style="height:
400px; width: 100%;
        xicreation-complete="onGridCreationComplete"
        xienable-Print="true" xienable-Preference-Persistence="true" xienable-
Export="true" xiforce-PagerRow="true"
        xienable-Drill-Down="true"
        xipage-Size="50" xienable-Filters="true" xienable-Footers="true">
            <level xichildren-Field="sub">
                <columns>
                    <!-- Don't forget <column xidata-Field="id" xiheader-Text="ID"> will
not work -->
                    <column xidata-Field="id" xiheader-Text="ID"></column>
                    <column xidata-Field="type" xiheader-Text="Type" xiheader-
Align="middle"
                    xifilter-Control="MultiSelectComboBox" xifilter-Combo-Box-
Build-From-Grid="true"
                    xienable-Filter-AutoComplete="true"></column>
                </columns>

                <next-Level>
                    <level >
                        <columns>

                            <column xidata-Field="id" xiheader-Text="ID" xihide-Header-
Text="true"></column>
                            <column xidata-Field="price" xiheader-Text="Price" xihide-
Header-Text="true"></column>
                        </columns>
                    </level>
                </next-Level>

```

```

        </level>

    </div>

</div>
</div>
</body>
</html>

```

For hierarchical grids, we add a the next-level tag [next-level] after columns. The XML structure would be the same as any of our examples in the demo, just with angular-style property names.

You can download these files from <http://blog.htmltreegrid.com/post/Angular-JS-Support.aspx>

## Miscellaneous

### Advanced Configuration Options – Single Level Grids

So far, we have looked at a fairly simplistic example of the power and features of the HTMLTreeGrid. In this section, let's review what a real grid looks like in a typical LOB application. This example is a part of the Sample project you get when you request a trial. It is the very first example that you will see on our demo pages as well.

Lets start with a screen shot:

Items 1 to 50 of 411. Page 1 of 9							
<div> <div> <div>Go to Page: 1</div> <div> <div></div> <div></div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> <div></div> <div></div> </div> </div>							
ID	LegalName of the Organization	Address					Financials
		Lines		Region			Annual Revenue
		Line 1	Line 2	City	State	Country	
				All	All	All	to
20800	3M Co	863 Newark St	Suite #913	Springfield	North Carolina	United States	12,000
20801	Abbott Laboratori...	502 Park Rd	Suite #31	Grand Rapids	North Carolina	United States	51,000
20802	Adobe Systems	448 King Ave	Suite #790	Springfield	Ohio	United States	28,000
20803	Advanced Micro ...	884 West Lane	Suite #776	Springfield	Penn	United States	31,000
20804	Aetna Inc	295 Newark Blvd	Suite #64	Stroudsburch	Ohio	United States	18,000
20805	Affiliated Comput...	957 Gardner Ave	Suite #897	Grand Rapids	Penn	United States	1,000
20806	AFLAC Inc	736 West St	Suite #19	Stroudsburch	New York	United States	51,000
20807	Air Products & Ch...	519 Gardner Blvd	Suite #266	Springfield	North Carolina	United States	25,000
20808	Airgas Inc	573 Newark Lane	Suite #213	Albany	Michigan	United States	42,000
		Count:411.00					Avg:\$30,055.79

As compared to the earlier simple example grids, you will notice there are a number of additional features that you will observe.

- 1) Left locked columns
- 2) Various different kinds of filters
- 3) Grouped Columns
- 4) The grid appears initially sorted
- 5) The text of the Legal Name column is truncated
- 6) The alignment of numeric columns is right aligned.
- 7) Formatters have been applied to both data and footer fields for the numeric columns
- 8) Headers are word wrap enabled

These are all configuration options that are available to you, to save you time, and help you build applications quickly.

Let's look at the markup used to generate this grid. A lot of this looks quite familiar now that you have gone through the simple examples. But you will notice quite a few new properties, each with their own functionality:

```

    <grid id="grid" enablePrint="true"
enablePreferencePersistence="true"
                                enableExport="true"
preferencePersistenceKey="simpleGrid"
useCompactPreferences="true"
horizontalScrollPolicy="auto" footerDrawTopBorder="true">
        <level enablePaging="true" pageSize="50"
enableFilters="true"
enableFooters="true" initialSortField="legalName"
initialSortAscending="true">
            <columns>
                <column id="colId" dataField="id"
headerText="ID" filterControl="TextInput" filterWaterMark="Search"
                selectable="true" filterComparisionType="number"/>
                <column id="colLegalName"
dataField="legalName" sortCaseInsensitive="true" selectable="true"
                headerText="LegalName of the Organization"
                headerWordWrap="true" truncateToFit="true"
columnLockMode="left" filterComparisionType="string"/>
                <column id="colLine2"
dataField="headquarterAddress.line2" selectable="true"
headerText="Line 2"
                filterComparisionType="string"/>
            </columns>
        </columnGroup>
        <columnGroup
headerText="Region">
            <columns>
                <column id="colCity"
dataField="headquarterAddress.city.name" headerText="City"
filterControl="MultiSelectComboBox" filterComboBoxWidth="150"
filterComboBoxBuildFromGrid="true" filterComparisionType="string"/>
                <column id="colState"

```

```

dataField="headquarterAddress.state.name" headerText="State"
filterControl="MultiSelectComboBox" filterComboBoxWidth="150"
filterComboBoxBuildFromGrid="true" filterComparisionType="string"/>
        <column
id="colCountry" dataField="headquarterAddress.country.name"
headerText="Country" filterControl="MultiSelectComboBox"
filterComboBoxWidth="150" filterComboBoxBuildFromGrid="true"
filterComparisionType="string"/>
                </columns>
        </columnGroup>
</columnGroups>
</columnGroup>
<columnGroup headerText="Financials">
    <columns>
        <column headerAlign="right"
id="colAnnRev" dataField="annualRevenue" headerText="Annual Revenue"

                headerWordWrap="true" textAlign="right"
        footerLabel="Avg:" footerOperation="average" footerAlign="center"
        footerOperationPrecision="2"

        labelFunction="flexiciousNmsp.UIUtils.dataGridFormatCurrencyLabelFu
nction"

                filterControl="NumericRangeBox" sortNumeric="true"
        footerFormatter="new flexiciousNmsp.CurrencyFormatter"

        paddingRight="10"

        headerPaddingRight="20" filterComparisionType="number"/>
                <column headerAlign="right"
id="colNumEmp" headerWordWrap="true" sortNumeric="true"
dataField="numEmployees" headerText="Num Employees" textAlign="right"
footerLabel="Avg:" footerOperation="average"
footerOperationPrecision="2"
        labelFunction="flexiciousNmsp.UIUtils.dataGridFormatCurrencyLabelFun
ction"

                headerPaddingRight="20
" filterComparisionType="number"/>
                <column headerAlign="right"
id="colEPS" headerWordWrap="true" sortNumeric="true"
dataField="earningsPerShare" headerText="EPS"

                textAlign="right"
        footerLabel="Avg:" footerOperation="average"

                footerFormatter="flexi
ciousNmsp.CurrencyFormatter"
        labelFunction="flexiciousNmsp.UIUtils.dataGridFormatCurrencyLabelFunc
tion"

                headerPaddingRight="20

```



```

" filterComparisionType="number"/>
                                <column headerAlign="right"
id="colStockPrice" headerWordWrap="true" sortNumeric="true"
dataField="lastStockPrice" headerText="Stock Price"

footerFormatter="flexiciousNmsp.CurrencyFormatter" textAlign="right"
footerLabel="Avg:" footerOperation="average"
footerOperationPrecision="2"
                                labelFunction="flexici
ousNmsp.UIUtils.dataGridFormatCurrencyLabelFunction"
headerPaddingRight="20" filterComparisionType="number"/>
                                </column>
                                </columnGroup>
                                </columns>
                                </level>

</grid>;

```

Among the first things you should notice about this markup is the introduction of the “level” tag. We have not yet covered this in detail, but among the most powerful features of the HTMLTreegrid product is its support for hierarchical data. We do not display hierarchical data in this example, but the level introduces an important concept that is universal. One of the most important concepts behind the Architecture of the grid arose from the fundamental requirement that it was created for - that is display of Heterogeneous Hierarchical Data (nested DataGrids - or DataGrid inside DataGrid). The notion of nested levels is baked in to the grid via the “columnLevel” property. This is a property of type “FlexDataGridColumnLevel”. This grid always has at least one column level. This is also referred to as the top level, or the root level. In flat grids (non hierarchical), this is the only level. But in nested grids, you could have any number of nested levels. The columns collection actually belongs to the columnLevel, and since there is one root level, the columns collection of the grid basically points to the columns collection of this root level.

In this example, we just have 1 level, which is also the top level. In fact, the columns you see in the grid are defined on the level object, not the grid. This is because for nested grids, you could have levels inside levels, each with its own set of columns.

```

<grid id="grid" . . . gridAttributes . . .>

    <level . . . levelAttributes . . .>

```

As we will see in a future example, for grids with multiple levels of hierarchy, it is possible to embed levels inside levels. If the number of levels is unknown at the outset, you can also set enableDynamicLevels to true, and the grid will inspect your data provider to figure out the depth needed.

The other important concept here is that of column groups. When you have a large set of columns, it helps to group them inside column groups. As you can observe, we can also nest column groups within columns groups:

```
<columnGroup headerText="Address" enableExpandCollapse="true" >
    <columnGroups>
        <columnGroup headerText="Lines"
    >
        <columns>
            <column . . .
```

Other than these, most of the properties in this markup are fairly self-explanatory. Each of these properties has a description in the API documentation. Please NOTE – since some properties have getters and setters, the documentation will most likely be associated with the corresponding getter OR setter, depending on which is most likely to be used.. For example:

```
labelFunction="flexiciousNmsp.UIUtils.dataGridFormatCurrencyLabelFunc
tion"
```

The documentation in this case is associated with labelFunction appears under  
getLabelFunction: [http://www.htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumn.html#method\\_getLabelFunction](http://www.htmltreegrid.com/docs/classes/flexiciousNmsp.FlexDataGridColumn.html#method_getLabelFunction)

### getLabelFunction

A function that determines the text to display in this column. By default the column displays the text for the field in the data that matches the column name. However, sometimes you want to display text based on more than one field in the data, or display something that does not have the format that you want. In such a case you specify a callback function using labelFunction.

For the DataGrid control, the method signature has the following form:

```
labelFunction(item:Object, column:DataGridColumn):String
```

Where item contains the DataGrid item object, and column specifies the DataGrid column.

A callback function might concatenate the firstName and lastName fields in the data, or do some custom formatting on a Date, or convert a number for the month into the string for the month.

The labelFunction also introduces an important concept, that of associating JavaScript callback functions to various APIs of the grid. The string that you pass into the value for this property is usually something that can be evaluated to a named javascript function. As an example:  
labelFunction="flexiciousNmsp.UIUtils.dataGridFormatCurrencyLabelFunc  
tion"

This function looks like:

```
/**
```

```

    * Datagrid function to format the date column in a grid.
    * @return A formatted date string
    *
    */
    UIUtils.dataGridFormatDateLabelFunction = function (item,
dgColumn) {
        var num = UIUtils.resolveExpression(item,
dgColumn.getDataField());
        var date = typeof num == 'string' ? new
Date(Date.parse(num.toString())) : num;
        if (date)
            return UIUtils.formatDate(date);
        return null;
    };

```

Note that each function that you associate with the string has to have a set contract. This contract is usually defined in the documentation of the property. In this example, the labelFunction has been documented as labelFunction(item:Object, column:DataGridColumn):String, which is what we have done in our UIUtils.dataGridFormatDateLabelFunction function.

## Advanced Configuration Options – Hierarchical Grids

Display of hierarchical data is the most powerful feature of the HTMLTreeGrid. It was designed from the ground up to display such complex data and relationships between the various data elements.






There are two types of configurations that we refer to in various examples: Nested and Grouped. In the nested data mode, also known as Heterogeneous Hierarchical Data we display related, but separate Business entities (usually parent child entities) in a nested hierarchical UI. Each level has its own set of columns. Each level can also have its own footers, paging bar etc. You will notice that enableFilters, enablePaging and enableFooters can be defined on each level. This gives you the Flexibility to have headers, footers, paging and filters at each level. Columns are also defined at each level. So each level can have an independent set of columns.

The second type of hierarchical data is grouped data (by using "reusePreviousLevelColumns"=true), which we also refer to as Homogeneous Hierarchical Data. This means that we're displaying properties of a single entity, just grouped along some attributes of that entity.

A picture is worth a thousand words, so here is the difference. The top image is nested data, the bottom is grouped data:

Notice the key difference is that Nested Data has columns at each level, while the Grouped Data just has a single set of top level columns.

Items 1 to 20 of 411. Page 1 of 21										Go to Page: 1										Edit Delete																													
<input type="checkbox"/> ID Legal Name										Annual Revenue										Num Employees										EPS										Stock Price									
<input type="checkbox"/>																																																	
▼ <input type="checkbox"/> 20800 3M Co										53,985										42,810										1.44										13.33									
<input type="checkbox"/> Deal Description										Deal Amount										Deal Date																													
▼ <input type="checkbox"/> Project # 1 - 3M Co - 6/2013										138,573										Jun-26-2013																													
<input type="checkbox"/> Invoice Number										Invoice Amount										Invoice Status										Invoice Date										Due Date									
▼ <input type="checkbox"/> 2080000										45,255										Paid										Feb-04-2014										Mar-06-2014									
<input type="checkbox"/> Line Item Description										Line Item Amount																																							
<input type="checkbox"/> Professional Services - Jason Bourne																														20,949																			
<input type="checkbox"/> Professional Services - Tarah Silverman																														24,306																			
										Count:2.00																				Total:\$45,255																			
▼ <input type="checkbox"/> 2080001										93,318										Transmitted										Oct-01-2012										Oct-31-2012									
<input type="checkbox"/> Line Item Description										Line Item Amount																																							
<input type="checkbox"/> Professional Services - Jason Bourne																														47,413																			
<input type="checkbox"/> Professional Services - Kristian Donovan																														45,905																			
										Count:2.00																				Total:\$93,318																			
										Count:2.00										Total:\$138,573																													
Items 1 to 2 of 2. Page 1 of 1										Go to Page: 1										Edit Delete																													
										Avg:\$32,573.47										Avg:\$29,978.44										Avg:\$4.09										Avg:\$20.72									

Items 1 to 20 of 411. Page 1 of 21					Go to Page: 1 ▾				
<input type="checkbox"/>	Name	Amount	Invoice Number	Invoice Status	Invoice Date	Due Date			
▼	<input type="checkbox"/> 3M Co	242,314		All ▾	All ▾	All ▾			
▼	<input type="checkbox"/> Project # 1 - 3M Co - 6/2013	138,573							
	<input type="checkbox"/> 2080000	45,255		Paid	Feb-04-2014	Mar-06-2014			
	<input type="checkbox"/> 2080001	93,318		Transmitted	Oct-01-2012	Oct-31-2012			
		Total:\$138,573	Count:2.00						
Items 1 to 2 of 2. Page 1 of 1					Go to Page: 1 ▾				
▼	<input type="checkbox"/> Project # 2 - 3M Co - 6/2013	103,741							
	<input type="checkbox"/> 2080010	74,248		Approved	Oct-06-2013	Nov-05-2013			
	<input type="checkbox"/> 2080011	29,493		Paid	Jul-04-2012	Aug-03-2012			
		Total:\$103,741	Count:2.00						
Items 1 to 2 of 2. Page 1 of 1					Go to Page: 1 ▾				
▼	<input type="checkbox"/> AFLAC Inc	225,609							
▼	<input type="checkbox"/> Project # 1 - AFLAC Inc - 7/2014	115,732							
	<input type="checkbox"/> 2080600	48,234		Draft	Jun-20-2014	Jul-20-2014			
	<input type="checkbox"/> 2080601	67,498		Paid	Jun-02-2014	Jul-02-2014			
		Total:\$115,732	Count:2.00						
Items 1 to 2 of 2. Page 1 of 1					Go to Page: 1 ▾				

When you download the full Sample, you will receive the configuration as well as the data for above examples to play with them further.

A key point to notice here, is that in the configuration of each grid, you will always find "Column Levels". Each "Level" is basically a holder for information that pertains to the hierarchical level. For flat grids (no hierarchy), there is just one column level, accessible via the `grid.getColumnLevel()`. For a grid with one data one level deep, there will be two levels, accessible via `grid.getColumnLevel()` and `grid.getColumnLevel().nextLevel`.

`FlexDataGridColumnLevel` is a class that contains information about a nest level of grid. This includes the columns at this level, information about whether or not to enable paging, footers, filters, the row sizes of each, the property of the dataprovider to be used as the key for

selection, the property of the data provider to be used as the children field, the renderers for each of the cells, etc. The Grid always contains at least one level. This is the top level, and is accessible via the `columnLevel` property of the grid.

One of the most important concepts behind the Architecture of Flexicious Ultimate arose from the fundamental requirement that the product was created for - that is display of Heterogeneous Hierarchical Data.

The notion of nested levels is baked in to the grid via the `"columnLevel"` property. This is a property of type `"FlexDataGridColumnLevel"`. The grid always has at least one column level. This is also referred to as the top level, or the root level. In flat grids (non hierarchical), this is the only level. But in nested grids, you could have any number of nested levels.

The columns collection actually belongs to the `columnLevel`, and since there is one root level, the columns collection of the grid basically points to the columns collection of this root level. The `FlexDataGridColumnLevel` class has a `"nextLevel"` property, which is a pointer to another instance of the same class, or a `"nextLevelRenderer"` property, which is a reference to a `ClassFactory` the next level. Please note, currently, if you specify `nextLevelRenderer`, the `nextLevel` is ignored. This means, at the same level, you cannot have both a nested subgrid as well as a level renderer. Bottom line - use `nextLevelRenderer` only at the innermost level. Our examples demonstrate this.

## Setting Column Widths

Usually, you can set `column width="xx"` and the column will take that many pixels of width. However, there are certain features and some nuances that make column widths a rather intriguing topic. The key here is the `columnWidthMode` property and how it interacts with the `horizontalScrollPolicy` property.

The `columnWidthMode` property on the column specifies how the column widths are applied. This property defaults to `"none"`. The Grid provides a rich mechanism to control column widths. Column widths are a complicated topic because there are a number of scenarios and rules that we need to account for

- When there is a horizontal scroll (`horizontalScrollPolicy=on` or `auto`): In this case, the columns are free to take as much width as they need. Below is how the column widths should handle in this case:
  - When `columnWidthMode=none` or `fixed`: The column will basically take the width specified by the width property
  - When `columnWidthMode=fitToContent`: The column will take the width calculated by its contents. The grid identifies the longest string to be displayed in this column, and sets the width of the column to this value.
  - When `columnWidthMode=percent`: This is not a valid setting when `horizontalScrollPolicy` is `on` or `auto`. The setting will be ignored and the column will take the width specified by the width property. When `horizontalScrollPolicy` is set to `auto` or `on`, `columnWidthMode=percent` holds no meaning, since there are

no fixed bounds to squish the columns within.

- When there is no horizontal scroll (`horizontalScrollPolicy=off` - this is the default):
  - When `columnWidthMode=none`: The column will take the width specified by the width property, and adjust for width (see sum of Column Widths exceeds Grid Width below).
  - When `columnWidthMode=fixed`: The column will take the width specified by the width property, and not adjust for width.
  - When `columnWidthMode=fitToContent`: The column will take the width calculated by its contents, and adjust for width (see sum of Column Widths exceeds Grid Width below).
  - When `columnWidthMode=percent`: For these columns, the grid divides the remaining width after allocating all fixed and fitToContent columns, on a percentage basis among all columns that have `columnWidthMode` set to percent. PLEASE NOTE : If you set `columnWidthMode='percent'`, also set `percentWidth`. Also, ensure that the `percentWidth` of the columns adds up to a 100.

Finally, there are the below calculations once the column widths are allocated:

- Grid Width exceeds Sum of Column Widths: The situation where the calculated column widths do not add up to the grid width is also handled on basis of the `horizontalScrollPolicy`.
  - When there is a horizontal scroll (`horizontalScrollPolicy=on` or `auto`): The last column extends to fill up all the remaining space. If you do not want your last column to resize, add a dummy column that has the following property (order is important) `minWidth="1" width="1" paddingLeft="0" paddingRight="0"`
  - When there is no horizontal scroll (`horizontalScrollPolicy=off` - this is the default): The extra width is divided proportionally between all the columns where `columnWidthMode` does not equal fixed.
- Sum of Column Widths exceeds Grid Width: Similarly, The situation where the allocated column widths exceed the width of the grid width is also handled on basis of the `horizontalScrollPolicy`.
  - When there is a horizontal scroll (`horizontalScrollPolicy=on` or `auto`): We simply show a scroll bar, and no column widths are changed.
  - When there is no horizontal scroll (`horizontalScrollPolicy=off` - this is the default): This excess width is reduced proportionally between columns where `columnWidthMode` does not equal fixed.

Left and right locked columns do not support column width mode, it is ignored for these. Finally, with multi level grids, if the hierarchy's columns width in the top level is smaller than the next level's width (and the horizontal scroll policy of the grid is "on"/"auto"), the horizontal scroller will be calculated only by the top level's width, making some columns in the next level unreachable. The recommendation is to give a large column width to the last top level column,

which is large enough so that sum of column widths at top level is larger than the sum of column widths at the bottom level.

Values : none,fixed,percent,fitToContent

## Selected Key Field

The `selectedKeyField` is a property on the object that identifies the object uniquely. Similar to a surrogate key, or a business key, as long as it uniquely identifies the object. When this property is set, the `selectedKeys` returns the ID values of the objects that were selected. When a row is selected in the grid, we store the `selectedKeyField` property of the selected object in this array collection. This allows for multiple pages of data that comes down from the server and not maintained in memory to still keep track of the ids that were selected on other pages.

If you use Flexicious in `filterPageSortMode=client`, this really does not apply to you, but in server mode, each page of data potentially represents a brand new dataprovider. Let's assume you have a Database table of 100,000 records, with the `pageSize` property set on Flexicious to 50. You load page 1, select a few items, and move on to page 2. The grid exposes a property called `selectedItems`, which will be lost when the new page of data is loaded. This is why we have the `selectedObjects` and `selectedKeys` property, that is, to keep the selection that was loaded in memory on prior pages of data. Now, in most LOB applications, each record can be identified by a surrogate key (or some unique identifier). This surrogate key is then used to uniquely identify different instances of at the same Object. For example, when the page 1 is loaded for the first time, there is a Employee OBJECT with `EmployeeId=1`. When the user selects this record, navigates to a different page, and switches back to page 1, the Employee with ID 1 is still there, and need to stay selected, but it could be an altogether different INSTANCE of the same record in the database. This is why we have the `selectedKeyField` property, which would in this case, be "EmployeeID" so we can uniquely identify the selection made by the user across multiple pages.

## Row and Column Span

The grid supports row and column spans inherently. Basically, the grid has two callback functions, `rowSpanFunction` and `colSpanFunction`.

The `rowSpanFunction` is a function that takes in a data object, and a column, and returns a number. -1 indicates that the row should span the entire height of the grid. 1 indicates the cell should occupy just its own spot. Any number greater than one would position the cell so it covers the width of that number of cells. Please note, row spans and col spans are only supported for data rows.

The `colSpanFunction` is a function that takes in a data object, and a column, and returns a number. -1 indicates that the row should span the entire width of the grid. Please note, row spans and col spans are only supported for data rows. Any number greater than one would position the cell so it covers the height of that number of cells. Since this function is defined on the grid, it will get a `IFlexDataGridCell` object that you should use to return a `colSpan`.

This is what a grid with Row Span looks like:



Survey Question	Answer	Freshman		Sophomore		Junior		Senior		Total	
		Count	Perccr	Count	Perccr	Count	Perccr	Count	Perccr	Count	Perccr
▼ Please rate your level of satisfaction with the sense of safety and security as experienced in your residential college/housing campus		955	22.48	1291	30.38	874	20.57	1129	26.57	4249	100
	Very Satisfied	106	13.70	274	35.40	130	16.80	264	34.11	774	18.22
	Moderately Satisf	244	38.67	144	22.82	143	22.66	100	15.85	631	14.85
	No Opinion/NA	228	27.18	166	19.79	175	20.86	270	32.18	839	19.75
	Moderately Dissati	169	18.57	343	37.69	272	29.89	126	13.85	910	21.42
	Very Satisfied	208	19.00	364	33.24	154	14.06	369	33.70	1095	25.77
		955		1291		874		1129		4249	
▼ Please rate your level of satisfaction with the availability of public transportation to and from the University Campus		1448	25.92	1379	24.68	1370	24.52	1390	24.88	5587	100
	Very Satisfied	278	30.75	101	11.17	247	27.32	278	30.75	904	16.18
	Moderately Satisf	386	31.08	278	22.38	288	23.19	290	23.35	1242	22.23
	No Opinion/NA	340	31.25	298	27.39	210	19.30	240	22.06	1088	19.47
	Moderately Dissati	335	28.44	347	29.46	249	21.14	247	20.97	1178	21.08
	Very Satisfied	109	9.28	355	30.21	376	32.00	335	28.51	1175	21.03
		1448		1379		1370		1390		5587	
▼ Please rate your level of satisfaction with the quality of the Intramural sports and recreation programs		1071	23.95	994	22.23	1376	30.77	1031	23.05	4472	100
	Very Satisfied	279	28.18	327	33.03	211	21.31	173	17.47	990	22.14
	Moderately Satisf	232	27.26	104	12.22	223	26.20	292	34.31	851	19.03
	No Opinion/NA	100	12.22	100	12.22	100	12.22	100	12.22	400	12.22
		4848		4958		4929		4656		19391	

This is what a grid with column span looks like:

Survey Question	Answer	Freshman		Sophomore		Junior		Senior		Total	
		Count	Perccr	Count	Perccr	Count	Perccr	Count	Perccr	Count	Perccr
▼ Please rate your level of satisfaction with the sense of safety and security as experienced in your residential college/housing campus		955	22.48	1291	30.38	874	20.57	1129	26.57	4249	100
	Very Satisfied	106	13.70	274	35.40	130	16.80	264	34.11	774	18.22
	Moderately Satisf	244	38.67	144	22.82	143	22.66	100	15.85	631	14.85
	No Opinion/NA	228	27.18	166	19.79	175	20.86	270	32.18	839	19.75
	Moderately Dissati	169	18.57	343	37.69	272	29.89	126	13.85	910	21.42
	Very Satisfied	208	19.00	364	33.24	154	14.06	369	33.70	1095	25.77
		955		1291		874		1129		4249	
▼ Please rate your level of satisfaction with the availability of public transportation to and from the University Campus		1448	25.92	1379	24.68	1370	24.52	1390	24.88	5587	100
	Very Satisfied	278	30.75	101	11.17	247	27.32	278	30.75	904	16.18
	Moderately Satisf	386	31.08	278	22.38	288	23.19	290	23.35	1242	22.23
	No Opinion/NA	340	31.25	298	27.39	210	19.30	240	22.06	1088	19.47
	Moderately Dissati	335	28.44	347	29.46	249	21.14	247	20.97	1178	21.08
	Very Satisfied	109	9.28	355	30.21	376	32.00	335	28.51	1175	21.03
		1448		1379		1370		1390		5587	
▼ Please rate your level of satisfaction with the quality of the Intramural sports and recreation programs		1071	23.95	994	22.23	1376	30.77	1031	23.05	4472	100
	Very Satisfied	279	28.18	327	33.03	211	21.31	173	17.47	990	22.14
	Moderately Satisf	232	27.26	104	12.22	223	26.20	292	34.31	851	19.03
	No Opinion/NA	100	12.22	100	12.22	100	12.22	100	12.22	400	12.22
		4848		4958		4929		4656		19391	

Lets take a quick look at the markup:

```
myCompanyNameSpace.SAMPLE_CONFIGS["RowSpanColSpan"]='<grid fontFamily="tahoma"
```



```

    fontSize="11" id="grid" enableDynamicLevels="true"
    rowSpanFunction="myCompanyNameSpace.rowSpanColSpan_getRowSpan"
    colSpanFunction="myCompanyNameSpace.rowSpanColSpan_getColSpan"
    enableDefaultDisclosureIcon="false"'+
    ,
    preferencePersistenceKey="rowSpanColSpan"
    on'+flexiciousNmisp.Constants.EVENT_CREATION_COMPLETE
    +'="myCompanyNameSpace.rowSpanColSpan_CreationComplete"'+
    ,
    cellBackgroundColorFunction="myCompanyNameSpace.rowSpanColSpan_getColor"
    horizontalGridLines="true"'+
    ,
    alternatingItemColors="[0xFFFFFFFF,0xE7F3FF]"
    headerColors="[0x298EBD,0x298EBD]"
    headerRollOverColors="[0x298EBD,0x298EBD]"'+
    ,
    columnGroupColors="[0x298EBD,0x298EBD]" footerColors="[0x298EBD,0x298EBD]"
    headerStyleName="whiteText" footerStyleName="whiteText"
    columnGroupStyleName="whiteText" '+
    ,
    footerRollOverColors="[0x298EBD,0x298EBD]" lockedSeperatorThickness="1"
    lockedSeperatorColor="0x6f6f6f" >'+
    ,
        <level childrenField="answers" enableFooters="true" >'+
    ,
        <columns>'+
    ,
            <column columnTextColor="0x17365D"
id="questionColumn" width="200" columnWidthMode="fixed" headerText="Survey
Question" dataField="question" wordWrap="true" enableExpandCollapseIcon="true"
paddingLeft="20" expandCollapseIconTop="4" expandCollapseIconLeft="4"/>'+
    ,
            <column width="150" headerText="Answer"
dataField="answerOption"/>'+
    ,
            <columnGroup headerText="Freshman">'+
    ,
                <columns>'+
    ,
                    <column dataField="freshmanCount"
headerText="Count" footerOperation="sum" footerOperationPrecision="0"
textAlign="right" headerAlign="right" footerAlign="right" paddingRight="5"/>'+
    ,
                    <column dataField="freshmanPercent"
headerText="Percent" footerOperationPrecision="0" textAlign="right"
headerAlign="right" footerAlign="right" paddingRight="5"/>'+
    ,
                </columns>'+
    ,
            </columnGroup>'+
    ,
            <columnGroup headerText="Sophomore">'+
    ,
                <columns>'+
    ,
                    <column dataField="sophomoreCount"
headerText="Count" footerOperation="sum" footerOperationPrecision="0"
textAlign="right" headerAlign="right" footerAlign="right" paddingRight="5"/>'+
    ,
                    <column dataField="sophomorePercent"
headerText="Percent" footerOperationPrecision="0" textAlign="right"
headerAlign="right" footerAlign="right" paddingRight="5"/>'+
    ,
                </columns>'+
    ,
            </columnGroup>'+
    ,
            <columnGroup headerText="Junior">'+
    ,
                <columns>'+
    ,
                    <column dataField="juniorCount"
headerText="Count" footerOperation="sum" footerOperationPrecision="0"
textAlign="right" headerAlign="right" footerAlign="right" paddingRight="5"/
>'+
    ,
                    <column dataField="juniorPercent"
headerText="Percent" footerOperationPrecision="0" textAlign="right"
headerAlign="right" footerAlign="right" paddingRight="5"/>'+
    ,
                </columns>'+

```

```

'                </columnGroup>' +
'                <columnGroup headerText="Senior">' +
'                    <columns>' +
'                        <column dataField="seniorCount"
headerText="Count" footerOperation="sum" footerOperationPrecision="0"
textAlign="right" headerAlign="right" footerAlign="right" paddingRight="5"/>' +
'                            <column dataField="seniorPercent"
headerText="Percent" footerOperationPrecision="0" textAlign="right"
headerAlign="right" footerAlign="right" paddingRight="5"/>' +
'                                </columns>' +
'                            </columnGroup>' +
'                                <columnGroup headerText="Total">' +
'                                    <columns>' +
'                                        <column dataField="totalCount"
headerText="Count" footerOperation="sum" footerOperationPrecision="0"
textAlign="right" headerAlign="right" footerAlign="right" paddingRight="5"/>' +
'                                            <column dataField="totalPercent"
headerText="Percent" footerOperationPrecision="0" textAlign="right"
headerAlign="right" footerAlign="right" paddingRight="5"/>' +
'                                                </columns>' +
'                                            </columnGroup>' +
'                                                </columns>' +
'                                            </level>' +
' </grid>';

```

```
myCompanyNameSpace.rowSpanColSpan_getRowSpan=function (cell){
```

```

    if(!myCompanyNameSpace.rowSpanColSpan_rbnRowSpanselected) return 1;
    if(cell.getLevel().getNestDepth()==1 //top level
        && cell.getLevel().isItemOpen(cell.rowInfo.getData())//item is open
        && cell.getColumn()
        && cell.getColumn().getDataField()=="question" //its the first column
        && cell.getRowInfo().getIsDataRow() //its the data row, not the header
        or the footer row
        && !cell.getRowInfo().getIsFillRow()//since the filler is also
considered a data row
    )
        return cell.getRowInfo().getData().answers.length+1;

    return 1;
};

```

```
myCompanyNameSpace.rowSpanColSpan_rbnColSpanselected=false;
myCompanyNameSpace.rowSpanColSpan_getColSpan=function (cell){
```

```

    if(!myCompanyNameSpace.rowSpanColSpan_rbnColSpanselected) return 1;
    if(cell.getLevel().getNestDepth()==1 //top level
        && cell.getColumn()
        && cell.getColumn().getDataField()=="question" //its the first column
        && cell.getRowInfo().getIsDataRow() //its the data row, not the header
        or the footer row
        && !cell.getRowInfo().getIsFillRow()//since the filler is also
considered a data row
    )
        return cell.getGrid().getColumns().length;
    return 1;
};

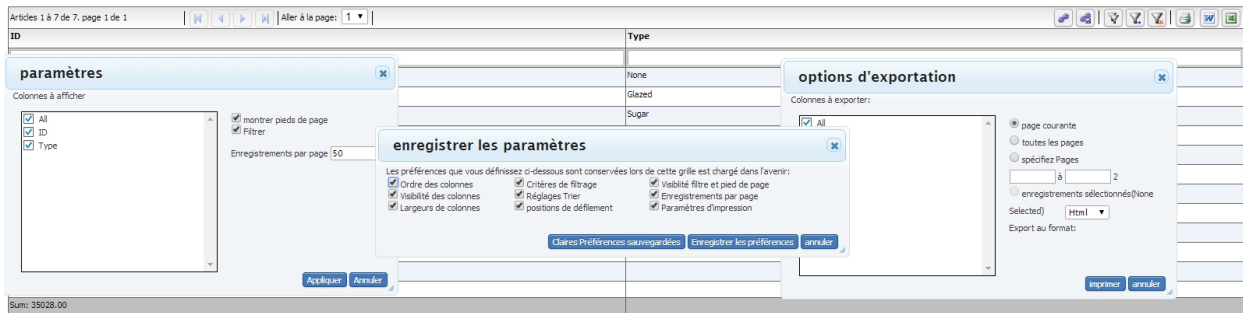
```

For a running version of this example, please refer :  
[http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=Row%20Span%20Col%20Span](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=Row%20Span%20Col%20Span)

## Localization

In this section, lets take a quick stab at implementing localization for the grid. This should be quite straightforward, in fact, there is just one class that holds all the strings used across the application. We have intentionally kept this as a stand alone class so you can integrate it with your localization code with ease. Lets take a look at a quick example:

You will notices that not only the paging section, but the popups, buttons, titles, every single string used in the grid is localizable. The code needed to do this follows the screen shot.



You can see a running example here : [http://www.htmltreegrid.com/demo/prod\\_ext\\_treegrid.html?example=Localization](http://www.htmltreegrid.com/demo/prod_ext_treegrid.html?example=Localization)

```
<!doctype html>
<html lang="en" >
<head>
  <meta charset="utf-8">
  <title>Simple</title>
  <!--These are jquery and plugins that we use from jquery-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-1.8.2.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery-ui-1.9.1.custom.min.js"></
script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.maskedinput-1.3.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.watermarkinput.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.ui.menu.js"></script>
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/external/js/adapters/jquery/jquery.toaster.js"></script>
  <!--End-->
  <!--These are specific to htmltreegrid-->
  <script type="text/javascript" src="http://www.htmltreegrid.com/
demo/minified-compiled-jquery.js"></script>
```

```

<script type="text/javascript" src="http://www.htmltreegrid.com/
demo/examples/js/Configuration.js"></script>
<script type="text/javascript" src="http://www.htmltreegrid.com/
demo/themes.js"></script>
<!--End-->
<!--css imports-->
<link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
flexicious/css/flexicious.css"
    type="text/css" />
<link rel="stylesheet" href="http://www.htmltreegrid.com/demo/
external/css/adapters/jquery/jquery-ui-1.9.1.custom.min.css"
    type="text/css" />
<!--End-->
<script type="text/javascript">

    $(document).ready(function(){
        flexiciousNmsp.Filter.ALL_ITEM = "tous";
        flexiciousNmsp.Constants.MCS_LBL_TITLE_TEXT = "Trier la
colonne multi";
        flexiciousNmsp.Constants.MCS_LBL_HEADER_TEXT = "S'il vous
plaît spécifier l'ordre de tri et de la direction des colonnes que
vous souhaitez trier par:";
        flexiciousNmsp.Constants.MCS_LBL_SORT_BY_TEXT = "Trier
par:";
        flexiciousNmsp.Constants.MCS_LBL_THEN_BY_TEXT = "Then
par:";
        flexiciousNmsp.Constants.MCS_RBN_ASCENDING_LABEL =
"ascendant";
        flexiciousNmsp.Constants.MCS_RBN_DESCENDING_LABEL =
"descendant";
        flexiciousNmsp.Constants.MCS_BTN_CLEAR_ALL_LABEL =
"effacer tout";
        flexiciousNmsp.Constants.MCS_BTN_APPLY_LABEL =
"appliquer";
        flexiciousNmsp.Constants.MCS_BTN_CANCEL_LABEL =
"annuler";

        flexiciousNmsp.Constants.SETTINGS_COLUMNS_TO_SHOW =
"Colonnes à afficher";
        flexiciousNmsp.Constants.SAVE_SETTINGS_TITLE = "Les
préférences que vous définissez ci-dessous sont conservées lors de
cette grille est chargé dans l'avenir:";
        flexiciousNmsp.Constants.SAVE_SETTINGS_PREFERENCE_NAME =
"Nom de l'option:";
        flexiciousNmsp.Constants.SAVE_SETTINGS_ORDER_OF_COLUMNS
= "Ordre des colonnes";
        flexiciousNmsp.Constants.SAVE_SETTINGS_VISIBILITY_OF_COLU
MNS = "Visibilité des colonnes";
        flexiciousNmsp.Constants.SAVE_SETTINGS_WIDTHS_OF_COLUMNS
= "Largeurs de colonnes";
        flexiciousNmsp.Constants.SAVE_SETTINGS_FILTER_CRITERIA =

```

```

"Critères de filtrage";
    flexiciousNmsp.Constants.SAVE_SETTINGS_SORT_SETTINGS =
"Réglages Trier";
    flexiciousNmsp.Constants.SAVE_SETTINGS_SCROLL_POSITIONS
= "positions de défilement";
    flexiciousNmsp.Constants.SAVE_SETTINGS_FILTER_AND_FOOTER_
VISIBILITY = "Visibilité filtre et pied de page";
    flexiciousNmsp.Constants.SAVE_SETTINGS_RECORDS_PER_PAGE
= "Enregistrements par page";
    flexiciousNmsp.Constants.SAVE_SETTINGS_PRINT_SETTINGS =
"Paramètres d'impression";
    flexiciousNmsp.Constants.SAVE_SETTINGS_REMOVE_ALL_SAVED_P
REFERENCES = "Supprimer toutes les préférences sauvegardées";
    flexiciousNmsp.Constants.SAVE_SETTINGS_CLEAR_SAVED_PREFER
ENCES = "Claire Préférences sauvegardées";
    flexiciousNmsp.Constants.SAVE_SETTINGS_SAVE_PREFERENCES
= "Enregistrer les préférences";
    flexiciousNmsp.Constants.SAVE_SETTINGS_CANCEL =
"annuler";

    flexiciousNmsp.Constants.SETTINGS_COLUMNS_TO_SHOW =
"Colonnes à afficher";
    flexiciousNmsp.Constants.SETTINGS_SHOW_FOOTERS = "montrer
pieds de page";
    flexiciousNmsp.Constants.SETTINGS_SHOW_FILTER =
"Filtrer";
    flexiciousNmsp.Constants.SETTINGS_RECORDS_PER_PAGE =
"Enregistrements par page";
    flexiciousNmsp.Constants.SETTINGS_APPLY = "Appliquer";
    flexiciousNmsp.Constants.SETTINGS_CANCEL = "Annuler";

    flexiciousNmsp.Constants.OPEN_SETTINGS_DEFAULT = "Par
défaut?";
    flexiciousNmsp.Constants.OPEN_SETTINGS_PREFERENCE_NAME =
"Nom de l'option";
    flexiciousNmsp.Constants.OPEN_SETTINGS_DELETE =
"effacer";
    flexiciousNmsp.Constants.OPEN_SETTINGS_APPLY =
"appliquer";
    flexiciousNmsp.Constants.OPEN_SETTINGS_REMOVE_ALL_SAVED_P
REFERENCES = "Supprimer toutes les préférences sauvegardées";
    flexiciousNmsp.Constants.OPEN_SETTINGS_SAVE_CHANGES =
"Enregistrer les modifications";
    flexiciousNmsp.Constants.OPEN_SETTINGS_CLOSE = "fermer";

    flexiciousNmsp.Constants.PGR_BTN_WORD_TOOLTIP = "Exporter
vers Word";
    flexiciousNmsp.Constants.PGR_BTN_EXCEL_TOOLTIP =
"Exporter vers Excel";

```

```

        flexiciousNmsp.Constants.PGR_BTN_PDF_TOOLTIP = "Imprimer
au format PDF";
        flexiciousNmsp.Constants.PGR_BTN_PRINT_TOOLTIP =
"imprimer";
        flexiciousNmsp.Constants.PGR_BTN_CLEAR_FILTER_TOOLTIP =
"Effacer le filtre";
        flexiciousNmsp.Constants.PGR_BTN_RUN_FILTER_TOOLTIP =
"Exécuter Filtre";
        flexiciousNmsp.Constants.PGR_BTN_FILTER_TOOLTIP =
"Afficher / Masquer filtre";
        flexiciousNmsp.Constants.PGR_BTN_FOOTER_TOOLTIP =
"Afficher / Masquer Footer";
        flexiciousNmsp.Constants.PGR_BTN_SAVE_PREFS_TOOLTIP =
"Enregistrer les préférences";
        flexiciousNmsp.Constants.PGR_BTN_PREFERENCES_TOOLTIP =
"préférences";
        flexiciousNmsp.Constants.PGR_BTN_COLLAPSE_ALL_TOOLTIP =
"Réduire tout";
        flexiciousNmsp.Constants.PGR_BTN_EXP_ALL_TOOLTIP =
"Développer tout";
        flexiciousNmsp.Constants.PGR_BTN_EXP_ONE_UP_TOOLTIP =
"Développer un Level Up";
        flexiciousNmsp.Constants.PGR_BTN_EXP_ONE_DOWN_TOOLTIP =
"Développer un niveau plus bas";
        flexiciousNmsp.Constants.PGR_BTN_MCS_TOOLTIP = "Tri sur
plusieurs colonnes";

        flexiciousNmsp.Constants.PGR_BTN_FIRST_PAGE_TOOLTIP =
"Première page";
        flexiciousNmsp.Constants.PGR_BTN_PREV_PAGE_TOOLTIP =
"page précédente";
        flexiciousNmsp.Constants.PGR_BTN_NEXT_PAGE_TOOLTIP =
"page suivante";
        flexiciousNmsp.Constants.PGR_BTN_LAST_PAGE_TOOLTIP =
"Dernière page";
        flexiciousNmsp.Constants.PGR_LBL_GOTO_PAGE_TEXT = "Aller
à la page:";

        flexiciousNmsp.Constants.PGR_ITEMS = "Articles";
        flexiciousNmsp.Constants.PGR_TO = "à";
        flexiciousNmsp.Constants.PGR_OF = "de";
        flexiciousNmsp.Constants.PGR_PAGE = "page";

        flexiciousNmsp.Constants.SELECTED_RECORDS =
"enregistrements sélectionnés";
        flexiciousNmsp.Constants.NONE_SELECTED = "Aucune
sélection";

```

```

flexiciousNmsp.Constants.EXP_LBL_TITLE_TEXT = "options
d'exportation";
flexiciousNmsp.Constants.EXP_RBN_CURRENT_PAGE_LABEL =
"page courante";
flexiciousNmsp.Constants.EXP_RBN_ALL_PAGES_LABEL =
"toutes les pages";
flexiciousNmsp.Constants.EXP_RBN_SELECT_PGS_LABEL =
"spécifiez Pages";
flexiciousNmsp.Constants.EXP_LBL_EXPORT_FORMAT_TEXT =
"Export au format:";
flexiciousNmsp.Constants.EXP_LBL_COLS_TO_EXPORT_TEXT =
"Colonnes à exporter:";
flexiciousNmsp.Constants.EXP_BTN_EXPORT_LABEL =
"exporter";
flexiciousNmsp.Constants.EXP_BTN_CANCEL_LABEL =
"annuler";

```

```

flexiciousNmsp.Constants.PPRV_LBL_TITLE_TEXT = "options
d'impression";
flexiciousNmsp.Constants.PRT_LBL_TITLE_TEXT = "options
d'impression";
flexiciousNmsp.Constants.PRT_LBL_PRT_OPTIONS_TEXT =
"options d'impression:";
flexiciousNmsp.Constants.PRT_RBN_CURRENT_PAGE_LABEL =
"page courante";
flexiciousNmsp.Constants.PRT_RBN_ALL_PAGES_LABEL =
"toutes les pages";
flexiciousNmsp.Constants.PRT_RBN_SELECT_PGS_LABEL =
"spécifiez Pages";
flexiciousNmsp.Constants.PRT_CB_PRVW_PRINT_LABEL =
"Aperçu avant impression";
flexiciousNmsp.Constants.PRT_LBL_COLS_TO_PRINT_TEXT =
"Colonnes à imprimer:";
flexiciousNmsp.Constants.PRT_BTN_PRINT_LABEL =
"imprimer";
flexiciousNmsp.Constants.PRT_BTN_CANCEL_LABEL =
"annuler";

```

```

flexiciousNmsp.Constants.PPRV_LBL_PG_SIZE_TEXT = "Taille
de la page:";
flexiciousNmsp.Constants.PPRV_LBL_LAYOUT_TEXT = "Mise en
page:";
flexiciousNmsp.Constants.PPRV_LBL_COLS_TEXT =
"colonnes:";
flexiciousNmsp.Constants.PPRV_CB_PAGE_HDR_LABEL = "tête
de page";
flexiciousNmsp.Constants.PPRV_CB_PAGE_FTR_LABEL = "Pied

```

```

de page";
flexiciousNmsp.Constants.PPRV_CB_RPT_FTR_LABEL = "tête de
l'état";
flexiciousNmsp.Constants.PPRV_CB_RPT_HDR_LABEL = "Rapport
pied de page";
flexiciousNmsp.Constants.PPRV_BTN_PRT_LABEL = "imprimer";
flexiciousNmsp.Constants.PPRV_BTN_CANCEL_LABEL =
"annuler";
flexiciousNmsp.Constants.PPRV_LBL_SETTINGS_1_TEXT =
"Remarque: Modification de la taille de page ou mise en page ne
mettra à jour l'aperçu, pas la réelle impression.";
flexiciousNmsp.Constants.PPRV_LBL_SETTINGS_2_TEXT = "S'il
vous plaît régler la Taille de la page / Mise en page sur Paramètres
de l'imprimante via la boîte de dialogue Imprimer qui sera affiché
lorsque vous imprimez.";
flexiciousNmsp.Constants.PPRV_BTN_PRT_1_LABEL =
"imprimer";
flexiciousNmsp.Constants.PPRV_BTN_CANCEL_1_LABEL =
"annuler";

flexiciousNmsp.Constants.SETTINGS_POPUP_TITLE =
"paramètres";
flexiciousNmsp.Constants.SAVE_SETTINGS_POPUP_TITLE =
"enregistrer les paramètres";
flexiciousNmsp.Constants.OPEN_SETTINGS_POPUP_TITLE =
"Gérer les paramètres";
flexiciousNmsp.Constants.EXPORT_OPTIONS_TITLE = "options
d'exportation";
flexiciousNmsp.Constants.PRINT_OPTIONS_TITLE = "options
d'impression";

var grid = new
flexiciousNmsp.FlexDataGrid(document.getElementById("gridContainer"),
{
    configuration: '<grid id="grid"
enablePrint="true" enablePreferencePersistence="true"
enableExport="true" forcePagerRow="true" pageSize="50"
enableFilters="true" enableFooters="true" enablePaging="true">' +
                    '                <level>' +
                    '                <columns>' +
                    '                <column
dataField="id" headerText="ID" filterControl="TextInput"
filterOperation="Contains" footerLabel="Sum: " footerOperation="sum"
footerOperationPrecision="2"/>' +
                    '                <column
dataField="type" headerText="Type" filterControl="TextInput"
filterOperation="Contains" />' +
                    '                </columns>' +
                    '                </level>' +

```



```

        ' ' +
        ' </grid>',
    dataProvider: [
        { "id": "5001", "type": "None",
"active" : true },
        { "id": "5002", "type": "Glazed"
, "active" : true},
        { "id": "5005", "type": "Sugar"
, "active" : true},
        { "id": "5007", "type":
"Powdered Sugar" , "active" : false},
        { "id": "5006", "type":
"Chocolate with Sprinkles" , "active" : true},
        { "id": "5003", "type":
"Chocolate" , "active" : false},
        { "id": "5004", "type": "Maple"
, "active" : true}
    ]
    });

});
</script>
</head>
<body>
    <div id="gridContainer" style="height: 400px; width: 100%;">
    </div>
</body>
</html>

```

## Next Steps & Further Reading

---

This guide was designed to get your feet wet in terms of the list of features available with the product. However, what we have covered here is merely the tip of the iceberg. There are so many different features, settings and configuration options, that going over each example by example would take forever. Once you get familiar with the concepts we have covered so far, you should be able grab the demo console, as well as the Stand Alone trial that you get when you request a trial, and be able to consume it with relative ease.

### Next Steps

- ➔ Download a Trial at <http://www.htmltreegrid.com/Home/Trial>, This will get you the entire source for the demo console <http://www.htmltreegrid.com/Home/Demo> as well as a StandAlone example that covers a number of features offered by the product.
- ➔ Take a look at <http://www.htmltreegrid.com/Home/Demo>. It covers a large number of scenarios that you are likely to encounter as you build LOB apps. We have built those examples from real use cases that customers have presented to us. Each example comes with source (click on the source tab). This tab will give you the configuration used for each grid. Start with an example that meets your needs, pickup its configuration, and give it a data provider.

If you have any questions, feel free to reach out to us at [support@flexicious.com](mailto:support@flexicious.com), and we will be glad to assist!