Guide to Using NIM.ui.InfiniteScroller

v1.0

Table of Contents

The InfiniteScroller control provides a fluid, multi-page control for displaying tabular data across A-grade browsers. The Infinite Scroller widget handles datasets of any size, and provides a desktop-like experience by allowing the user to scroll to view all the data available in the data set.

The following steps outline how to make use of the InfiniteScroller control:

Step 1: Including required dependencies

The following dependencies need to be included in your page:

Javascript:

- itemcache.js
- cached.mail.controller.js
- YUI Selector
- YUI Element
- YUI DataSource
- YUI DataTable
- infinitescroller.js

CSS:

- YUI Skin Sam DataTable CSS
- infinitescroller.css

Step 2: Instantiating the InfiniteScroller UI control

The InfiniteScroller control accepts a single configuration object in order to initialize it. The following configuration properties are available:

Name	Type	Description
container	String | HTMLElement	(Required) The parent DOM container in which to render the InfiniteScroller
columns	Object	(Required) A list of column objects to be rendered in the InfiniteScroller (see Column Definitions below)

Name	Type	Description
dataSource	NIM.util.DataSource	(Required) A NIM.util.DataSource instance (see Data Source Definition below)
sortedBy	Object	An object defining the initial sort column for the InfiniteScroller. The object contains the following properties: • *key*: This is the column's key to set the initial sort order for the InfiniteScroller • *dir*: The direction to sort (available options: NIM.ui.InfiniteScroller.SORT_ASC and NIM.ui.InfiniteScroller.SORT_DESC)
MSG_LOADING	String	The string to display when messages are loading in the InfiniteScroller
MSG_EMPTY	String	The string to display when there are no records in the InfiniteScroller

prefetch	Object	An object to indicate the table should be preloaded with data. The object contains the following properties:
		• handler: The callback function to call in order to indicate which records should be loaded. The function should accept the following parameters:
		  • data: An object containing the following properties:
		  • start: The start offset/index of the viewable segment of rows to sort
		  • end: The end offset/index of the viewable segment of rows to sort
		• scope: The object to set as the execution scope of the handler function

Name	Type	Description
formatRow	HTMLFunction	A function to call to perform any custom formatting for the table row. The function should accept the following parameters: • elTr: The DOM node representing the row in the table • record: The Record object for the row in the table
dragdrop	NIM.ui.BaseDDProxy	A drag and drop proxy object to support drag and drop in the InfiniteScroller

***Example**:*

```
var myScrollGrid = new NIM.ui.InfiniteScroller( {
    container: "scrollContainer",
    columns: orderedColumns,
    dataSource: dataSource,
    sortedBy: {
        key: "date",
        dir: NIM.ui.InfiniteScroller.SORT_ASC
    },
    MSG_LOADING: "Loading messages...",
    MSG_EMPTY: "No messages in folder",
    prefetch: {
        handler: loadMyData,
        scope: this
    },
    formatRow: myCustomRowFormatter,
    dragdrop: myCustomMessageListProxy
} );
```

Column Definitions:

Columns are defined as objects containing the following properties:

Property	Type	Description
key	String	(Required) The unique name assigned to each Column. When a Column key maps to a DataSource field, cells of the Column will automatically populate with the the corresponding data. If a key is not defined in the Column definition, one will be auto-generated.
field	String	The DataSource field mapped to the Column. By default, the field value is assigned to be the Column's key. Implementers may specify a different field explicitly in the Column definition. This feature is useful when mapping multiple Columns to a shared field, since keys must remain unique, or when the field name contains characters invalid for DOM or CSS usage.
label	String	By default, the Column's label is populated with the Column's key. Supply a label to display a different header.
formatter	HTMLFunction	A function or a pointer to a function to handle HTML formatting of cell data.
minWidth	Number	Minimum pixel width. Please note that minWidth validation is executed after cells are rendered, which may cause a visual flicker of content.
resizeable	Boolean	True if Column is resizeable.
width	Number	Pixel width. (nb: It is recommended to set the column's width using CSS rather than hard coding the width, as it allows for easier customization without having to change the Javascript code. See "Styling Column Widths" below).

Columns are displayed in the order in which they are defined.

***Example**:*

```
var columns = {
    attachments: { key: "attachments", formatter: this.formatAttachment, label: "%",
sortable: true, resizeable: false, minWidth: 35 },

    sender: { key: "sender", formatter: this.formatSender, label: "Sender", sortable:
true, resizeable: true, minWidth: 100 },

    subject: { key: "subject", formatter: this.formatSubject, label: "Subject",
sortable: true, resizeable: true, minWidth: 150 },

    date: { key: "date", label: "Date", formatter: this.formatDate, sortable: true,
resizeable: true, minWidth: 100 },

    size: { key: "size", label: "Size", formatter: this.formatSize, sortable: true,
resizeable: true, minWidth: 35 },

    checked: { key: "checked", formatter: NIM.ui.InfiniteScroller.COLUMN_CHECKBOX,
selectAll: true, resizeable: false }
};
```

Styling Column Widths:

It is recommended to set the width of a column through CSS rather than JavaScript. This allows for easier customization of the column width (ie: as per a client's requirement). A CSS hook is provided for each column when it is created by the InfiniteScroller. A class name is assigned to each column using the following format:

- col-<column-key>

<column-key> indicates the column's key (specified when the columns were created). By exposing this CSS hook, setting the width of a column is easily accomplished through CSS:

Example:

```
.col-mycolumnkey{
    width: 100px;
}
```

Data Source Definition:

A DataSource provides a common configurable interface for which other components can fetch tabular data from. It is a required dependency of the InfiniteScroller control.

To create a DataSource object, simply instantiate it:

Example:

```
var myDataSource = new NIM.util.DataSource();
```

DataSource uses a responseSchema to determine what data gets parsed out for use by the calling widget. Defining a schema for your data allows DataSource to cache and return only the data consumed by UI controls. A responseSchema is an array of strings that define the fields of the data it is storing. Defining a schema can be done as follows:

Example:

```
var dataSource = new NIM.util.DataSource();
dataSource.responseSchema = {
    fields: [ "property1", "property2", "property3" ]
}
```

Please note that the order of the fields is irrelevant since the name of the field maps to result data, and data values that don't have a defined field are ignored.

Step 3: Subscribing to the InfiniteScroller's events

The InfiniteScroller control provides a robust Custom Event model to allow you to seamlessly integrate and expand upon its built-in fuctionality. The InfiniteScroller fires the following Custom Events:

Name	Description	Parameters
dataChangeEvent(oArgs)	Fired when there has been a change to the data being displayed in the control (ie: user has scrolled, and new records need to be loaded)	• *oArgs.start*: The start offset/index of the viewable segment of rows to sort • *oArgs.end*: The end offset/index of the viewable segment of rows to sort

Name	Description	Parameters
columnSortEvent(oArgs)	Fired when a column is sorted.	• *oArgs.column*: The Column's key • *oArgs.order*: The sort order (NIM.ui.InfiniteScroller.SORT_ASC or NIM.ui.InfiniteScroller.SORT_DESC) • *oArgs.start*: start: The start offset/ index of the viewable segment of rows to sort • *oArgs.end*: The end offset/index of the viewable segment of rows to sort
selectAllClickEvent(oArgs)	Fired when a column defined as a NIM.ui.InfiniteScroller.COLUMN_CHECKBOX with selectAll behaviour is clicked	• oArgs.checked: True if select all is checked
checkboxClickEvent(oArgs)	Fired when a CHECKBOX element is clicked	• oArgs.checked: True if the item was checked • oArgs.element: The DOM element reference of the checkbox element • oArgs.recordId: The record ID corresponding to the checked row
rowClickEvent(oArgs)	Fired when a row has a click	• oArgs.recordId: The record ID corresponding to the clicked row
rowMouseoverEvent(oArgs)	Fired when a row has a mouseover	• oArgs.recordId: The record ID corresponding to the clicked row
rowMouseoutEvent(oArgs)	Fired when a row has a mouseout	• oArgs.recordId: The record ID corresponding to the clicked row
rowDragStartEvent(oArgs)	Fired when a row drag has started.	• oArgs.recordId: The record ID corresponding to the clicked row
rowDragEndEvent(oArgs)	Fired when a row drag has ended.	• oArgs.recordId: The record ID corresponding to the clicked row

Subscribing to any of its events can be done as follows:

Example:

```
function onDataChange( oArgs ){
    // need to load records from start to end
    myController.load( oArgs.start, oArgs.end );
}

myScrollGrid.subscribe( "dataChangeEvent", onDataChange, this, true );
```

The subscribe method takes the following parameters:

Name	Type	Description
p_type	String	The name of the event
p_fn	HTMLFunction	The callback function to execute when the event fires
p_obj	Object	An object to be passed along when the event fires
p_override	Boolean	If true, the obj passed in becomes the execution scope of the listener

Step 4: Updating the InfiniteScroller's row data

The InfiniteScroller exposes two methods to render its data:

Name	Description	Parameters
setTotalRows	Sets the total number of rows in the table	*totalRows*: The total number of rows in the entire record set
updateData	Updates the table's RecordSet with the given rows and updates the UI with the new rows	*rows*: Array of rows to render