js.datagrid

Versión 0.1.1

API Reference

Contenidos

[Dependencias 4](#_Toc227411314)

[lib.jquery-1.2.5.---.js 4](#_Toc227411315)

[lib.base.js 4](#_Toc227411316)

[lib.validation.js 4](#_Toc227411317)

[ui.datepicker.js 4](#_Toc227411318)

[ui.datepicker-es.js 4](#_Toc227411319)

[lib.caretaker.js 4](#_Toc227411320)

[lib.datagrid.commands.js 5](#_Toc227411321)

[Constructor 5](#_Toc227411322)

[Métodos 5](#_Toc227411323)

[init 5](#_Toc227411324)

[toggleColumns 5](#_Toc227411325)

[refresh 6](#_Toc227411326)

[getRows 6](#_Toc227411327)

[refreshRow 6](#_Toc227411328)

[appendRow 6](#_Toc227411329)

[insertRow 6](#_Toc227411330)

[removeRow 7](#_Toc227411331)

[editRow 7](#_Toc227411332)

[commitEdit 7](#_Toc227411333)

[cancelEdit 8](#_Toc227411334)

[setSort 8](#_Toc227411335)

[getSort 8](#_Toc227411336)

[Eventos 9](#_Toc227411337)

[created 9](#_Toc227411338)

[row\_created: {grid, rowIndex, dataItem} 9](#_Toc227411339)

[row\_rollover: {grid, rowIndex, colIndex, dataItem} 9](#_Toc227411340)

[row\_rollout: {grid, rowIndex, colIndex, dataItem} 9](#_Toc227411341)

[row\_click: {grid, rowIndex, coldIndex, dataItem} 9](#_Toc227411342)

[row\_edit: {grid, rowIndex, colIndex, dataItem} 9](#_Toc227411343)

[row\_update\_failed: {grid, rowIndex, oldValues, newValues, dataItem} 10](#_Toc227411344)

[row\_updating: { grid, rowIndex, oldValues, newValues, dataItem, cancel } 10](#_Toc227411345)

[row\_updated: {grid, rowIndex, oldValues, newValues, dataItem} 10](#_Toc227411346)

[row\_edit\_cancel: {grid, rowIndex, dataItem} 10](#_Toc227411347)

[row\_command: {grid, commandName, rowIndex, dataItem} 10](#_Toc227411348)

[sort\_request: {grid, colIndex, sortBy, sortDirection} 10](#_Toc227411349)

[subgrid\_created: {grid, subgrid} 10](#_Toc227411350)

[footer\_created: {grid, tfoot} 10](#_Toc227411351)

# Dependencias

## lib.jquery-1.2.5.---.js

**[Requerido]**

Librería de jQuery. La mínima versión requerida es 1.2.5. Esta librería viene en tres formas:

* **Full**  
  Versión legible de jQuery. Ideal para debuguear sobre entornos de desarrollo. No se recomiendo su uso en producción debido a su tamaño (100kb ~).
* **Minified**  
  En esta versión, el archivo es optimizado eliminando todos los comentarios, espacios en blanco, renombrando variables para acortar sus nombres y hasta modificando el código para hacerlo más compacto (55kb ~).
* **Packed**  
  Es una versión aun más reducida. Presenta exactamente la misma funcionalidad pero el código se encuentra comprimido y ofuscado. Ideal para usar en producción (30kb ~).

## lib.base.js

**[Requerido]**

Presenta los tipos base de los componentes [js.luminicbox](http://www.luminicbox.com/) además de métodos helpers.

## lib.validation.js

**[Requerido para el modo de edición]**

Provee la funcionalidad de validación.

## ui.datepicker.js

**[Requerido para soportar fechas y modo de edición]**

Provee los métodos helper para el parse de fechas y el componente para mostrar el calendario en modo edición.

## ui.datepicker-es.js

**[Requerido para soportar fechas y modo de edición]**

Provee la información de regionalización para ui.datepicker.js.

## lib.caretaker.js

**[Requerido para undo/redo]**

Contiene la clase Caretaker encargada de mantener el estado de los comandos para el undo/redo.

## lib.datagrid.commands.js

**[Requerido para undo/redo]**

Contiene los comandos de undo/redo que aplican a la grilla.

# Constructor

DataGrid(container, schema, model, buildHeader)

Crea una instancia de grilla sobre el container especificado.

Una vez que la grilla es creada y se subscribe a los eventos, se debe llamar al método init().

#### Parametros

* **container** (DOM Node) - **Requerido**   
  Representa el nodo padre donde se creará la tabla.
* **schema** (ARRAY) – **Requerido**  
  Define el esquema para las columnas de la tabla. Debe mapearse con propiedades en los elementos del modelo. Ver [documentación general](js.datagrid.general.docx) para definición de esquema.
* **model** (ARRAY) – **Requerido**  
  Modelo de datos.
* **buildHeader** (BOOL) – Opcional  
  Indica si se deben mostrar la cabecera de las columnas.

# Métodos

## init

Genera la grilla en el contenedor especificado. Debe llamarse la primera vez, después de llamar al constructor y subscribirse a los eventos.  
Eventos generados: [*created*](#_created:_{grid})

*Ejemplo:*

var grid = new DataGrid(document.getElementById("simpleGridContainer"), $schema, $data, true);

grid.init();

## toggleColumns

#### Parámetros

* **colIndexes** – (ARRAY) - **Requerido**

Índices de las columnas a mostrar u ocultar.

Colapsa o muestra las columnas especificadas. El parámetro es un array con los índices de las columnas que se quieren ocultar o mostrar.

*Ejemplo:*

grid.toggleColumns([5,6]);

## refresh

Refresca la grilla a partir de las modificaciones que haya sufrido el modelo.  
Eventos generados: [*created*](#_created:_{grid})

## getRows

#### Retorna ARRAY

Devuelve un array con las filas de la tabla generada (array de <tr>).

## refreshRow

#### Parámetros

* **rowIndex** – (NUMBER) **–** **Requerido**  
  Índice de la fila a refrescar

Refresca la fila especificada buscando cambios en el modelo.

*Ejemplo:*

// Refrescar la primera fila

grid.refreshRow(0);

## appendRow

#### Parámetros

* **dataItem** – (OBJECT) – **Requerido**  
  Elemento modelo para agregar a la grilla. Debe seguir el mismo esquema que los demás ítems del modelo.

Agrega una nueva fila al final de la tabla.  
Eventos generados: [*row\_created*](#_row_created:_{grid,_rowIndex,)

*NOTA: Este método no modifica el modelo interno de la grilla, por lo que llamar a* [*refresh()*](#_refresh) *provocaría que la nueva fila desaparezca*

*Ejemplo:*

grid.appendRow( { Name: "John Doe", Email: "john.doe@gmail.com", Age: 23 } );

## insertRow

#### Parámetros

* **dataItem** – (OBJECT) – **Requerido**  
  Elemento modelo para agregar a la grilla. Debe seguir el mismo esquema que los demás ítems del modelo.
* **rowIndex** – (NUMBER) – **Requerido**  
  Índice de la fila donde se desea agregar el nuevo elemento.

Agrega una nueva fila en determinada posición.Eventos generados: [*row\_created*](#_row_created:_{grid,_rowIndex,) *NOTA: Este método no modifica el modelo interno de la grilla, por lo que llamar a* [*refresh()*](#_refresh) *provocaría que la nueva fila desaparezca*

*Ejemplo:*

grid.insertRow( { Name: "John Doe", Email: "john.doe@gmail.com", Age: 23 }, 0 ); // agrega como primera fila

## removeRow

#### Parámetros

* **rowIndex** – (NUMBER) - RequeridoÍndice de la fila a remover.

Remueve una fila en determina posición.

*NOTA: Este método no modifica el modelo interno de la grilla, por lo que llamar a* [*refresh()*](#_refresh) *provocaría que la nueva fila reaparezca.*

## editRow

#### Parámetros

* **rowIndex** – (NUMBER) – **Requerido**Índice de fila que se desea editar
* **colIndex** – (NUMBER) – Opcional  
  Índice de la columna que debe tener foco inicial. Valor por defecto: 0.

Cambia una fila a modo edición.  
Eventos generados: [*row\_edit*](#_row_edit:_{grid,_rowIndex,)

*Ejemplo:*

grid.editRow(0); // primera fila, primera columna editable

## commitEdit

#### Returna BOOLEAN

Intenta guardar los cambios realizados en la fila que se encuentra en modo de edición.  
Si no hay validación asociada, se aplican los cambios, se cambia a modo lectura y devuelve True.

Si hay alguna validación asociada y esta no pasa, entonces se devuelve False, sin cancelar o aplicar los cambios.  
Eventos generados: [*row\_update\_failed*](#_row_update_failed:_{grid,_rowIndex,), [*row\_updating*](#_row_updating:_{_grid,), [*row\_updated*](#_row_updated:_{grid,_rowIndex,).

## cancelEdit

Cancela los cabios realizados en la fila que se encuentra en modo de edición.  
Eventos generados: [*row\_edit\_cancel*](#_row_edit_cancel:_{grid,_rowIndex,)

## setSort

#### Parámetros

* **colIndex** – (NUMBER) – **Requerido**   
  Índice de la columna supuestamente ordenada.
* **sortDirection** – (NUMBER) - **Requerido**   
  Dirección del ordenamiento
  + 1 = Ascendente
  + -1 = Descendente

Indica, a modo informativo, como se encuentran ordenados los datos. Agrega indicadores en la cabecera de la columna correspondiente.

*Ejemplo:*

// on sort request

grid.events.bind("sort\_request", function(e, args) {

var propType = $schema[args.colIndex].type;

var propName = $schema[args.colIndex].propertyName;

$data = sortData($data, propName, propType, args.sortDirection);

// update

grid.model = $data;

grid.refresh();

grid.setSort(args.colIndex, args.sortDirection);

});

## getSort

#### Retorna [Object] {colIndex, sortDirection}

Devuelve un objeto con la información sobre ordenamiento.

* **colIndex** – (NUMBER)  
  Índice de la columna supuestamente ordenada.
* **sortDirection** – (NUMBER)  
  Dirección del ordenamiento
  + 1 = Ascendente
  + -1 = Descendente

# Eventos

La grilla genera una serie de eventos ante las distintas acciones del usuario. Fácilmente se puede subscribir a estos eventos para extender la funcionalidad de la grilla o sobrescribir comportamientos propios.

**Subscripción a un evento:**

[instanciaGrilla].events.bind([eventName], [callbackFunction])

La *callbackFunction* debe aceptar dos parámetros, un string que contiene el nombre del evento (*e*) y un objeto javascript el cual contiene los argumentos del evento (*args*).

*Ejemplo:*

grid.events.bind("row\_click", function(e, args) {

var data = args.dataItem;

alert("Clicked on '" + data.Name + "'");

});

## created

#### Argumentos

* **grid**  
  Referencia al componente

Se dispara una vez rellenada la grilla. Después de haber llamado a init() o refresh().

## row\_created

#### Argumentos

* **grid**Referencia al componente
* **rowIndex**Índice de fila creada
* **dataItem**Modelo que corresponde a la fila

Se dispara cuando una nueva fila es agregada a la tabla.  
Este evento es ideal para generar modificaciones sobre las celdas.

*Ejemplo:*

grid.events.bind("row\_created", function(e, args) {

var tr = grid.getRows()[args.rowIndex];

var firstCell = tr.childNodes[0];

if(!args.dataItem.IsMember) {

// hide

firstCell.firstChild.style.visibility = 'hidden';

}

});

## row\_rollover

#### Argumentos

* **grid**
* **rowIndex**
* **colIndex**
* **dataItem**

Se dispara cuando el cursor pasa por encima de una celda o fila.

*Ejemplo*:

grid.events.bind("row\_rollover", function(e, args) {

var dataItem = args.dataItem;

var txt = dataItem.Name + ": " + dataItem.Email + ", Registration: " + dataItem.Registration.toString();

document.getElementById('statusBar').text = txt;

});

## row\_rollout

#### Argumentos

* **grid**
* **rowIndex**
* **colIndex**
* **dataItem**

Se dispara cuando el cursor se va de una celda o fila.

*Ejemplo*:

grid.events.bind("row\_rollout", function(e, args) {

document.getElementById('statusBar').text = "";

});

## row\_click

#### Argumentos

* **grid**
* **rowIndex**
* **colIndex**
* **dataItem**

Se dispara cuando se realiza click sobre una celda o fila. Aplica solamente si la columna no tiene habilitada la edición.

*Ejemplo*:

grid.events.bind("row\_click", function(e, args) {

var rowData = args.dataItem

alert("Clicked on '" + rowData.Name + "'");

});

## row\_edit

#### Argumentos

* **grid**
* **rowIndex**
* **colIndex**Índice de la celda clickeada
* **dataItem**Modelo que corresponde a esa fila

Se dispara cuando el usuario clickea sobre una celda y la columna tiene habilitada la edición. Automáticamente se cambia esa fila a modo edición.

## row\_update\_failed

#### Argumentos

* **grid**
* **rowIndex**
* **oldValues**  
  Diccionario con los valores originales
* **newValues**  
  Diccionario con los nuevos valores, incluso los valores que fallaron en la validación
* **dataItem**Modelo que corresponde a esa fila

Se dispara cuando el usuario intenta aplicar los cambios realizados y la validación falla.

## row\_updating

#### Argumentos

* **grid**
* **rowIndex**
* **oldValues**Diccionario con los valores originales
* **newValues**Diccionario con los nuevos valores
* **dataItem**
* **cancel**Boolean que sirve para cancelar el behavior por defecto de la grilla (en este caso, actualizar los valores en el modelo)

Se dispara después que la validación fue exitosa y antes de aplicar los cambios al model.  
Desde el event handler se puede cambiar el valor de *cancel* a *true* para cancelar que la grilla aplique los cambios. Este evento es ideal para introducir validaciones más complejas o sobrescribir el comportamiento por defecto de cómo se guardan los cambios (como cuando se usa el undo/redo).

// Caretaker

caretaker = new CommandCaretaker();

// Grid

// ...

// Grid Events

grid.events.bind("row\_updating", function(e, args) {

// cancel default grid behavior

args.cancel = true;

// DataGridRowEditCommand(grid, gridRow, dataItem, oldValues, newValues)

var cmd = new DataGridRowEditCommand(args.grid, args.dataItem, args.rowIndex, args.oldValues, args.newValues);

caretaker.pushCommand(cmd);

cmd.commit();

});

## row\_updated

#### Argumentos

* **grid**
* **rowIndex**
* **oldValues**
* **newValues**
* **dataItem**

Se dispara una vez que los cambios de edición se aplicaron al modelo.

## row\_edit\_cancel

#### Argumentos

* **grid**
* **rowIndex**
* **dataItem**

Se dispara cuando la edición de una fila es cancelada (por código usando [cancelEdit()](#_cancelEdit) o cuando el usuario presiona la tecla ESC).

## row\_command

#### Argumentos

* **grid**
* **commandName**Nombre del comando especificado en el esquema de columna
* **rowIndex**
* **dataItem**

Se dispara cuando se clickea sobre un botón o link. El parámetro *commandName* contiene el mismo *commandName* indicado en el esquema.

*Ejemplo*:

// on delete

grid.events.bind("row\_command", function(e, args) {

if (args.commandName == "delete") {

if (!confirm("Are you sure you want to delete '" + args.dataItem.Name + "'?")) return;

// DataGridRowDeleteCommand(grid, rowIndex)

var cmd = new DataGridRowDeleteCommand(args.grid, args.rowIndex);

caretaker.pushCommand(cmd);

cmd.commit();

}

});

## sort\_request

#### Argumentos

* **grid**
* **colIndex**  
  Índice de la columna seleccionada
* **sortBy**  
  Nombre de la propiedad seleccionada
* **sortDirection**  
  Dirección del ordenamiento
  + 1: Ascendenete
  + -1: Descendente

Se dispara al clickear sobre la cabecera de una columna que tiene el sorting activado.

*Ejemplo*:

// on sort request

grid.events.bind("sort\_request", function(e, args) {

// get sort info

var propType = $schema[args.colIndex].type;

var propName = $schema[args.colIndex].propertyName;

// sort data

$data = sortData($data, propName, propType, args.sortDirection);

// update

grid.model = $data;

grid.refresh();

grid.setSort(args.colIndex, args.sortDirection);

});

## subgrid\_created

#### Argumentos

* **grid**Referencia a la grilla padre
* **subgrid**Referencia a la grilla anidada

Se dispara cuando una grilla anidada es creada (cuando dispara el [*created*](#_created) event de la grilla hija).

Para subscribirse a eventos de las subgrillas la forma es la siguiente:

// row on click

grid.events.bind("subgrid\_created", function(e, args) {

var subgrid = args.subgrid;

subgrid.events.bind("row\_click", function(e, args) {

var rowData = args.dataItem

alert("Clicked on '" + rowData.Name + "'");

});

});

*Ejemplo de cómo integrar undo/redo en grillas anidadas*:

$(document).ready(function() {

// caretaker

caretaker = new CommandCaretaker();

// create grid

var grid = new DataGrid(document.getElementById("gridContainer"), $schema, $data, false);

// bind to subgrid-create and override row\_updating

grid.events.bind("subgrid\_created", function(e, args) {

// subgrid

var subgrid = args.subgrid;

// on updating

subgrid.events.bind("row\_updating", function(e, args) {

// cancel default grid behavior

args.cancel = true;

// DataGridRowEditCommand(grid, gridRow, dataItem, oldValues, newValues)

var cmd = new DataGridRowEditCommand(args.grid, args.dataItem, args.rowIndex, args.oldValues, args.newValues);

caretaker.pushCommand(cmd);

cmd.commit();

});

});

// init grid

grid.init();

});

## footer\_created

#### Argumentos

* **grid**
* **tfoot**Referencia al nodo <tfoot> de la tabla

Se dispara cuando el footer es creado o actualizado. Sólo aplica si alguna columna del esquema posee la propiedad *calculateTotal.*