js.datagrid

Versión 0.1.1

Documentación General, Escenarios de Uso y Ejemplos

Contenidos

[Características 4](#_Toc227398942)

[Licencia 4](#_Toc227398943)

[Dependencias 4](#_Toc227398944)

[Modelos y Esquemas de modelo 5](#_Toc227398945)

[Esquema de columnas 6](#_Toc227398946)

[Métodos y eventos 10](#_Toc227398947)

[Métodos 10](#_Toc227398948)

[Eventos 11](#_Toc227398949)

[Escenarios de uso 12](#_Toc227398950)

[Grilla sencilla 12](#_Toc227398951)

[Grilla compleja 13](#_Toc227398952)

[Columnas colapsables 17](#_Toc227398953)

[Edición 17](#_Toc227398954)

[Validación en edición 17](#_Toc227398955)

[Soporte de Undo/Redo sobre edición 18](#_Toc227398956)

[Integrar Undo/Redo 18](#_Toc227398957)

[Eliminación de filas 19](#_Toc227398958)

[Grillas anidadas 19](#_Toc227398959)

[Ejemplos 22](#_Toc227398960)

[Grilla sencilla 22](#_Toc227398961)

[Grilla compleja 22](#_Toc227398962)

[Ordenamiento client-side 22](#_Toc227398963)

[Edición 22](#_Toc227398964)

[Edición con validación y undo/redo 22](#_Toc227398965)

[Grillas anidadas 22](#_Toc227398966)

[Edición en grillas anidadas 22](#_Toc227398967)

[Troubleshooting 22](#_Toc227398968)

[Columnas desiguales en grillas anidadas 22](#_Toc227398969)

[Columnas desacomodadas en modo edición 24](#_Toc227398970)

# Características

Js.datagrid es un componente client-side para la generación de tablas y grillas editables.

Este componente usa un modelo de datos definido como un Array de JavaScript, y un esquema que le indica que columnas crear en la grilla y como estas se deben mapear al modelo.

**Características principales**

* Generación automática de tablas
* Varios tipos de datos soportados
* Soporte de edición *in-line*
* Soporte de Undo/Redo
* Grillas anidadas

**NOTA**: El componente no viene con ninguna integración con servicios AJAX. La tarea de “enlazar” el componente con los datos de un servicio remoto queda en el desarrollador.

## Licencia

Este componente se distribuye bajo [licencia MIT](http://es.wikipedia.org/wiki/MIT_License).

Esta licencia permite reutilizar el Software así licenciado tanto para ser software libre como para ser software no libre, permitiendo no liberar los cambios realizados al programa original.

js.luminicbox - lib.datagrid.js v0.1.1

Copyright (c) 2006-2009 - Pablo Costantini (luminicbox.com)

Permission is hereby granted, free of charge, to any person

obtaining a copy of this software and associated documentation

files (the "Software"), to deal in the Software without

restriction, including without limitation the rights to use,

copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the

Software is furnished to do so, subject to the following

conditions:

The above copyright notice and this permission notice shall be

included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES

OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND

NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT

HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,

WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING

FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR

OTHER DEALINGS IN THE SOFTWARE.

## Dependencias

El componente requiere de [jQuery](http://jquery.com/) 1.2.5 (o superior) y de la librería base *lib.base.js*, la cual incluye funciones comunes para todos los componentes.

**Scripts:** lib.jquery-1.2.5.js (o lib.jquery-1.2.5-min.js), lib.base.js, lib.datagrid.js **CSS:** datagrid.css

#### Soporte para fechas (read-only y edición)

**Scripts:** ui.datepicker.js, ui.datepicker-es.js

#### Edición

**Scripts:** lib.validation.js **CSS:** ui.datepicker.css

#### Undo/Redo

**Scripts:** lib.caretaker.js, lib.datagrid.commands.js

# Modelos y Esquemas de modelo

La grilla usa dos objetos para generar dinámicamente la tabla. Por un lado está el modelo, un array con los datos que se desean mostrar y un esquema que define como *armar* la tabla.

El modelo en si es sencillo, es un array de objetos (entidades). Fácilmente se puede usar la respuesta de un servicio JSON que devuelve un array de entidades.

Los siguientes tipos de datos son soportados:

* String
* Number (entero o float)
* Date
* Otro objecto Javascript [Object] (usando el *formatString* para imprimir un valor o generando una grilla anidada) – Ver [grillas anidadas](#_Grillas_anidadas_1).

**Ver** [**Ejemplo de uso**](#_Grilla_sencilla)**.**

**Ejemplo de modelo:**

var $model = [

{

Name: "Camacho, Amos H.",

Email: "adipiscing@laoreetlibero.org",

Gender: 1,

IsMember: true,

Registration: new Date(2008, 10, 29),

Points: 2199

},

{

Name: "Potts, Olga A.",

Email: "libero.et.tristique@CurabiturdictumPhasellus.com",

Gender: 2,

IsMember: true,

Registration: new Date(2009, 07, 26),

Points: 7000

},

{

Name: "Duke, Upton P.",

Email: "id.magna@suscipitnonummyFusce.org",

Gender: 1,

IsMember: false,

Registration: new Date(2009, 10, 14),

Points: 4949

}

];

El esquema define que celdas mostrar y como se deben mostrar/imprimir sus valores. Se compone con un array donde cada ítem define una celda.

**Ejemplo de Esquema:**

var $schema = [

{

headerText: "Name",

propertyName: "Name",

type: "string",

sortable: true

},

{

headerText: "Email",

propertyName: "Email",

type: "string",

className: "email",

sortable: true

},

{

headerText: "Gender",

propertyName: "Gender",

type: "string"

},

{

headerText: "Membership",

propertyName: "IsMember",

type: "boolean",

className: "bool"

}

];

## Esquema de columnas

El esquema de columna contiene las siguientes propiedades:

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Required | Expected type | Default Value (if none specified) |
| [headerText](#_headerText) | No | String | “” |
| [propertyName](#_propertyName) | Yes | String | - |
| [type](#_type) | No | String | “string” |
| [editable](#_editable) | No | Boolean | False |
| [validation](#_validation) | No | Enum or Function | - |
| [className](#_className) | No | String | - |
| [sortable](#_sortable) | No | Boolean | False |
| [formatString](#_formatString) | No | String or Function | - |
| [options](#_options) | No | Array | - |
| [commandName](#_commandName) | No | String | - |
| [calculateTotal](#_calculateTotal) | No | Boolean | False |

#### headerText

Indica el texto que se debe mostrar en la cabecera de la columna.  
Sólo aplica si la grilla fue instanciada con el parámetro *buildHeader* en *True.*

#### propertyName

Requerido – Nombre de la propiedad que se debe usar sobre la entidad para obtener el valor. Ej:

*// one record*

entidad = { **Name**: "John Doe", Age: 30 }

*// first column schema*

schema = { headerText: "Fullname", **propertyName: "Name"** }

#### type

Indica el tipo de datos que se espera del valor. Si la grilla soporta edición, se aplica también una validación según el tipo.

Ver [Ejemplos de uso](#_Tipos_de_datos).

Los valores posibles son:

* **string**Se imprime el valor tal cual se recibe. En caso de recibir algo distinto a un string, se imprime el resultado de .toString() de ese valor.
* **number**Imprime el valor en modo numérico. En modo edición valida que el valor ingresado sea un entero o un decimal.  
  *NOTA*: El “*number”* *type* es requerido es caso de usar *calculateTotal: true.*
* **boolean**Imprime un checkbox indicando el estado (*true* = checked, *false* = unchecked). En modo edición el checkbox se encuentra habilitado.
* **date**Imprime el valor con formato de fecha. En modo edición valida que el valor ingresado corresponda a una fecha.  
  *NOTA*: Require incluir la dependencia *ui.datepicker.js* (también *ui.datepicker-es.js* para formatear en modo DD/MM/YYYY).
* **schema**Define que en lugar de imprimir el valor se debe mostrar otra grilla en su lugar. Ver [Ejemplo de uso](#_Grillas_anidadas).
* **linkbutton o button**Genera un botón en lugar de imprimir un valor.  
  Se usa principalmente para generar botones, por ejemplo botones de “eliminar” o de “ver detalles”. Ver [*commandName*](#_commandName).

#### editable

Indica si la celda debe soportar edición. En caso de ser asi, clickear sobre la celda de la columna provoca que la fila cambie a modo edición.  
Aquellas columnas que tengan un *type* específico como *date* o *number* serán validados antes de aplicar los cambios. Si la validación falla, las celdas con error aparecerán remarcadas.  
Para validar valores ver la propiedad [*validation*](#_validation).

Ver [ejemplo de uso](#_Edición).

#### validation

Indica el tipo de validación a aplicar para esa columna cuando se aplican los cambios de edición.  
La validación a usar se puede indicar de dos formas, usando la built-in validation o pasando por referencia una función.

Especificando uno o más de los siguientes valores:

* DataGrid.Validation.REQUIRED
* DataGrid.Validation.NUMERIC
* DataGrid.Validation.PERCENTAGE
* DataGrid.Validation.ALPHA
* DataGrid.Validation.ALPHANUMERIC
* DataGrid.Validation.DATE
* DataGrid.Validation.EMAIL
* DataGrid.Validation.URL

Para especificar mútiples valores se debe usar el operador bitwise ‘|’. Ejemplos:

DataGrid.Validation.REQUIRED | DataGrid.Validation.URL  
Valida que sea requerido y contenga una URL válida.  
  
DataGrid.Validation.PERCENTAGE  
No valida que sea requerido, pero en caso de contener algún valor, que este será numérico y entre 0 y 100.

Ver [Ejemplos de uso](#_Validación_en_edición).

*NOTA*: Para validaciones más complejas, se puede usar el evento *row\_updating* para interceptar los cambios realizados y decidir si aplican o no. Ver *row\_updating* en [eventos](#_Lista_de_eventos).

#### className

Agrega a las celdas de esa columna el atributo “class” con el valor especificado. Este atributo se aplica a las celdas de la cabecera, cuerpo y pie de la tabla.  
Ejemplo:

*// CSS - align all numeric values to the right*

.**numeric** { text-align: right; }

*// Age column schema*

var colSchema = {

headerText: "Age",

propertyName: "Age",

type: "Number",

**className: "numeric"**

}

#### sortable

Indica si esa columna puede ser ordenada según sus valores. Si se indica que es *sortable,* un click sobre la cabecera de esa columna generará un evento de JavaScript.

*NOTA*: La grilla no re-ordena los datos a partir de esta propiedad. Sólo genera el evento de ‘pedido’ de ordenamiento. Queda en el desarrollador ordenar los datos, ya sea client-side o server-side.

Ver [Ejemplo de uso](#_Ordenamiento).

#### formatString

Indica cómo se debe formatear el valor a imprimir. Acepta un string o una función de javascript.  
Si *formatString* es "$ {v}" y el *valor* es 123. Devuelve "$ 123".  
Si *formatString*  es una función y el valor es 2332, entonces se llama a la función con el valor 123 y se imprime el nuevo valor de retorno.

Ver [Ejemplos de uso](#_Formateo_de_campos).

#### options

En caso que el valor representa una enumeración, se puede especificar una lista que representa la enumeración. De esta forma se logra imprimir el ‘friendly-name’ de la enumeración en vez de su valor.

En modo edición, el campo se reemplaza por una lista desplegable con las opciones de la enumeración.

El valor de *options* debe ser un array de objetos con la siguiente estructura { text, value }. Ejemplo:

{

text: "Título a mostrar",

value: 2

}

Ver [Ejemplo de uso](#_Opciones_predefinidas).

#### commandName

(Sólo aplica cuando el *type* es *linkbutton* o *button)*Indica el nombre del comando a usar cuando se clickea sobre el botón y se dispara el evento [*row\_command*](#_Eventos)*.*

#### calculateTotal

(Sólo aplica si el *type* de la columna es *numeric)*Indica si al pie de la tabla debe mostrar la suma de todos los valores para esa columna.

Ver [Ejemplo de uso](#_Valores_totales).

# Métodos y eventos

Las instancias de DataGrid poseen los siguientes métodos y eventos:

(Para más detalle ver [Referencia de la API](js.datagrid.reference.docx))

## Métodos

* **CONSTRUCTOR(container, schema, model, buildHeader)**  
  Crea una instancia del componente.
  + container: DIV contenedor donde se generará la grilla
  + schema: Esquema de grilla (array con esquemas de columna)
  + model: Modelo de datos (array de entidades)
  + buildHeader: Indica si se muestra la cabecera de tabla con nombres de columna
* **init()**Genera la grilla en el contenedor especificado. Debe llamarse la primera vez que se desea enlazar los datos.  
  Eventos generados: *created*
* **toggleColumns(colIndexes[])**Colapsa/muestra las columnas especificadas. El parámetro es un array con los índices de las columnas que se quieren ocultar o mostrar. Ver [Ejemplo de uso](#_Columnas_colapsables).
* **refresh()**Refresca la grilla a partir del modelo modificado.  
  Eventos generados: *created*
* **getRows():tr[]**Devuelve un array con las filas de la tabla generada (array de <tr>).
* **refreshRow(rowIndex)**Refresca la fila especificada buscando cambios en el modelo.
* **appendRow(dataItem)**Agrega una nueva fila al final de la tabla.  
  Eventos generados: *row\_created  
  NOTA: No modifica el modelo.*
* **insertRow(dataItem, rowIndex)**Agrega una nueva fila en determinada posición.Eventos generados: *row\_created**NOTA: No modifica el modelo.*
* **removeRow(rowIndex)**Remueve una fila en determina posición.*NOTA: No modifica el modelo.*
* **editRow(rowIndex, [colIndex])**Cambia una fila a modo edición.  
  Eventos generados: *row\_edit*
* **commitEdit():bool**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\_updating*, *row\_updated*.

* **cancelEdit()**Cancela los cabios realizados en la fila que se encuentra en modo de edición.  
  Eventos generados: *row\_edit\_cancel*
* **setSort(colIndex, direction)**Indica, a modo informativo, como se encuentran ordenados los datos. Agrega indicadores en la cabecera de la columna correspondiente.
* **getSort():{colIndex, sortDirection}**Devuelve un objeto con la información sobre ordenamiento.

## 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 + "'");

});

#### Lista de eventos

* **created: {grid}**Se dispara una vez rellenada la grilla. Después de haber llamado a init() o refresh().
* **row\_created: {grid, rowIndex, dataItem}**Se dispara cuando una nueva fila es agregada a la tabla.  
  Este evento es ideal para generar modificaciones sobre las celdas.
* **row\_rollover: {grid, rowIndex, colIndex, dataItem}**Se dispara cuando el cursor pasa por encima de una celda o fila.
* **row\_rollout: {grid, rowIndex, colIndex, dataItem}**Se dispara cuando el cursor se va de una celda o fila.
* **row\_click: {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.
* **row\_edit: {grid, rowIndex, colIndex, dataItem}**Se dispara cuando el usuario clickea sobre una celda y la columna soporta edición. Automáticamente se cambia esa fila a modo edición.
* **row\_update\_failed: {grid, rowIndex, oldValues, newValues, dataItem}**Se dispara cuando el usuario intenta aplicar los cambios realizados y la [validación](#_validation) falla.
* **row\_updating: { grid, rowIndex, oldValues, newValues, dataItem, cancel }**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).
* **row\_updated: {grid, rowIndex, oldValues, newValues, dataItem}**Se dispara una vez que los cambios de edición se aplicaron al modelo.
* **row\_edit\_cancel: {grid, rowIndex, dataItem}**Se dispara cuando la edición de una fila es cancelada.
* **row\_command: {grid, commandName, rowIndex, dataItem}**Se dispara cuando se clickea sobre un [botón o link](#_type). El parámetro *commandName* contiene el mismo [*commandName*](#_commandName) indicado en el esquema.
* **sort\_request: {grid, colIndex, sortBy, sortDirection}**Se dispara al clickear sobre la cabecera de una columna que tiene el [sorting activado](#_sortable).
* **subgrid\_created: {grid, subgrid}**Se dispara cuando una grilla anidada es creada (cuando dispara el *created* event de la grilla hija).
* **footer\_created: {grid, tfoot}**Se dispara cuando el footer es creado o actualizado. Sólo aplica si alguna columna del esquema posee la propiedad [*calculateTotal*](#_calculateTotal)*.*

# Escenarios de uso

## Grilla sencilla

Una grilla sencilla, de sólo lectura, sin edición y donde no hay interacción con el usuario se puede resolver en pocas líneas de código.

1. Agregar referencias de Scripts y CSS

<link type="text/css" href="css/datagrid.css" rel="stylesheet" />

<script type="text/javascript" src="js/lib.jquery-1.2.5.js"></script>

<script type="text/javascript" src="js/lib.base.js"></script>  
<script type="text/javascript" src="js/lib.datagrid.js"></script>

1. Contenedor HTML para la tabla  
   <div id="simpleGridContainer"></div>
2. Definir esquema para la grilla

var $schema = [

{

headerText: "Name",

propertyName: "Name"

},

{

headerText: "Email",

propertyName: "Email"

},

{

headerText: "Some Number",

propertyName: "NumericProp",

type: "number"

}];

1. Definir (u obtener de un servicio) el modelo que alimentará la grilla

var $data = [

{ Name: "Camacho, Amos H.", Email: "adipiscing@laoreetlibero.org", DateProp: new Date(2008, 10, 29), NumericProp: 2199 },

{ Name: "Potts, Olga A.", Email: "libero.et.tristique@CurabiturdictumPhasellus.com", DateProp: new Date(2009, 07, 26), NumericProp: 7000 },

{ Name: "Duke, Upton P.", Email: "id.magna@suscipitnonummyFusce.org", DateProp: new Date(2009, 10, 14), NumericProp: 4949 },

// …

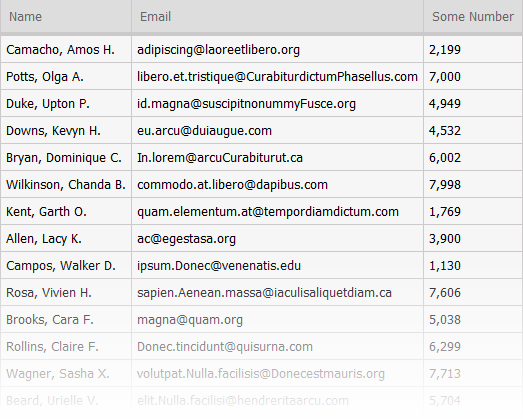
];

1. Instanciar e iniciar grilla

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

grid.init();

#### Resultado



## Grilla compleja

Para generar un grilla más compleja donde se deben mostrar distintos tipos de datos (fechas, números decimales o precios), se necesita personalizar la presentación (estilos) o agregar funcionalidades como ordenar o colapsar columnas.

#### Tipos de datos

Por definición ver [colschema - *type*](#_type).

{

headerText: "Membership",

propertyName: "IsMember",

type: "boolean",

className: "bool"

},

{

headerText: "A Date",

propertyName: "DateProp",

type: "date"

},

{

headerText: "Simple Format",

propertyName: "NumericProp",

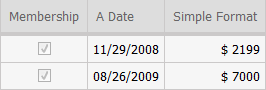
type: "number",

className: "num",

formatString: "$ {v}",

calculateTotal: true

}



#### Formateo de campos

Se indica por medio de la propiedad *formatString* en el esquema de columna.  
Por definición ver [colschema - *formatString*](#_formatString).

{

headerText: "Simple Format",

propertyName: "NumericProp",

type: "number",

className: "num",

formatString: "$ {v}"

},

{

headerText: "Complex Format",

propertyName: "NumericProp",

type: "numeric",

className: "num",

formatString: function(v) {

v += '';

x = v.split('.');

x1 = x[0];

x2 = x.length > 1 ? '.' + x[1] : '';

var rgx = /(\d+)(\d{3})/;

while (rgx.test(x1)) {

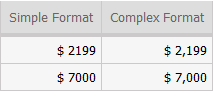
x1 = x1.replace(rgx, '$1' + ',' + '$2');

}

return "$ " + x1 + x2;

}

}



#### Opciones predefinidas

Por definición ver [colschema - *options*](#_options)

**Enumeración**

var $genderOptions = [

{ text: "Male", value: 1 },

{ text: "Female", value: 2 }

];

**Esquema de columna**

{

headerText: "Gender",

propertyName: "Gender",

type: "number",

options: $genderOptions

}

**Modelo**

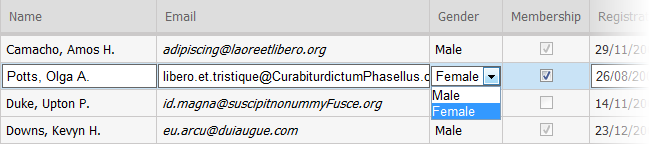
var $data = [

{ Name: "Camacho, Amos H.", Email: "adipiscing@laoreetlibero.org", Gender: 1 },

{ Name: "Potts, Olga A.", Email: "l.tristique@CurabiturdictumPhasellus.com", Gender: 2 }

//...

];



#### Valores totales

Se indica por medio de la propiedad *calculateTotal* en el esquema de columna.  
Por definición ver [colschema - *calculateTotals*](#_calculateTotal)

{

headerText: "Simple Format",

propertyName: "NumericProp",

type: "number",

className: "num",

calculateTotal: true

}

#### 

#### Ordenamiento

La grilla posee una serie de métodos y eventos que ayudan al ordenamiento de valores, pero la grilla por sí sola no ordena ningún valor. Esta tarea queda a cargo del desarrollador, implementando algún tipo de client-side o server-side sorting. Por definición ver [colschema - *sortable*](#_sortable)

**Esquema de columna**

{

headerText: "Name",

propertyName: "Name",

type: "string",

sortable: true

}

**Client-Side Sorting**

var grid;

$(document).ready(function() {

// create & first time sort

$data = sortData($data, "Name", "string", 1);

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

grid.init();

grid.setSort(0, 1);

// 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);

});

});

function sortData(data, propName, propType, sortDirection) {

var sortFunction;

if (propType == "date" || propType == "numeric") {

if (sortDirection == 1) {

sortFunction = function(a, b) { return a[propName] - b[propName]; }

} else {

sortFunction = function(a, b) { return b[propName] - a[propName]; }

}

} else {

if (sortDirection == 1) {

sortFunction = function(a, b) { return a[propName] > b[propName]; }

} else {

sortFunction = function(a, b) { return b[propName] > a[propName]; }

}

}

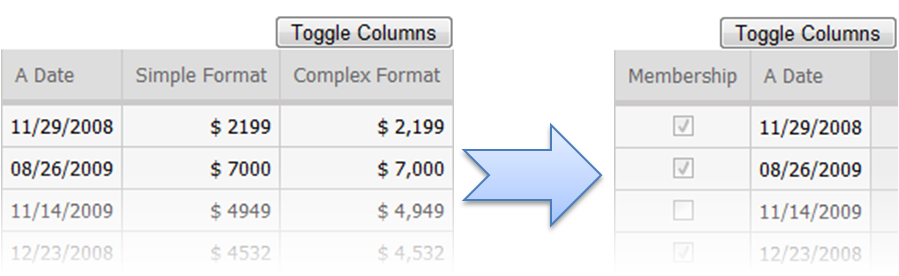
return data.sort(sortFunction);

}

## Columnas colapsables

Por definición ver *toggleColumns()* en “[Métodos](#_Métodos)”.

<input type="button" value="Toggle Columns" onclick="grid.toggleColumns([5,6])" />



## Edición

Para habilitar la edición en una grilla es necesario indicar que columnas son editables en el esquema.

{

headerText: "Name",

propertyName: "Name",

type: "string",

editable: true

},

{

headerText: "Email",

propertyName: "Email",

type: "string",

className: "email",

editable: true

},

{

headerText: "Gender",

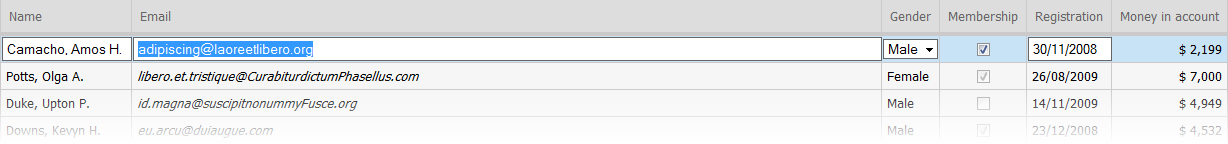
propertyName: "Gender",

type: "number",

options: $genderOptions,

editable: true

}



## Validación en edición

Por definición ver [colschema – *validation*](#_validation)*.*

{

headerText: "Name",

propertyName: "Name",

type: "string",

editable: true,

validation: DataGrid.Validation.REQUIRED

},

{

headerText: "Email",

propertyName: "Email",

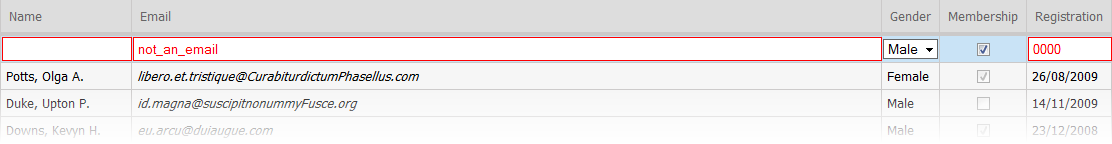
type: "string",

className: "email",

editable: true,

validation: DataGrid.Validation.REQUIRED | DataGrid.Validation.EMAIL

}



## Soporte de Undo/Redo sobre edición

El soporte de Undo y Redo sobre la grilla se logra cancelando el comportamiento de la grilla encargado de aplicar los cambios de vuela al modelo. Esto se logra con el evento *row\_updating.*

Luego es necesario reemplazar este comportamiento usando los comandos de *lib.datagrid.commands.js*. Los niveles de “deshacer y rehacer” se logran con el uso de un [Caretaker](http://en.wikipedia.org/wiki/Memento_pattern) (incluído en *lib.caretaker.js*)

Ver [ejemplo final](#_Edición_con_validación).

### Integrar Undo/Redo

**Dependencias**

<!-- validation and datepicker -->

<script type="text/javascript" src="js/lib.validation.js"></script>

<script type="text/javascript" src="js/ui.datepicker.js"></script>

<script type="text/javascript" src="js/ui.datepicker-es.js"></script>

<!-- undo/redo fro datagrid -->

<script type="text/javascript" src="js/lib.caretaker.js"></script>

<script type="text/javascript" src="js/lib.datagrid.commands.js"></script>

**HTML**

<input id="btnUndo" type="button" value="Undo" onclick="caretaker.undo()" />

<input id="btnRedo" type="button" value="Redo" onclick="caretaker.redo()" />

**JavaScript para el Caretaker**

caretaker = new CommandCaretaker();

caretaker.onChange = function(undoCount, redoCount) {

$('#btnUndo')[0].disabled = !(undoCount > 0);

$('#btnRedo')[0].disabled = !(redoCount > 0);

};

caretaker.onChange(); // update buttons status

**Integrando Caretaker, Comandos y Eventos de grilla**

// on updating

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();

});

### Eliminación de filas

**Esquema de columna para botón de ‘Delete’**

{

headerText: "",

text: "Delete",

type: "button",

commandName: "delete"

}

**Integración de JavaScript**

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);

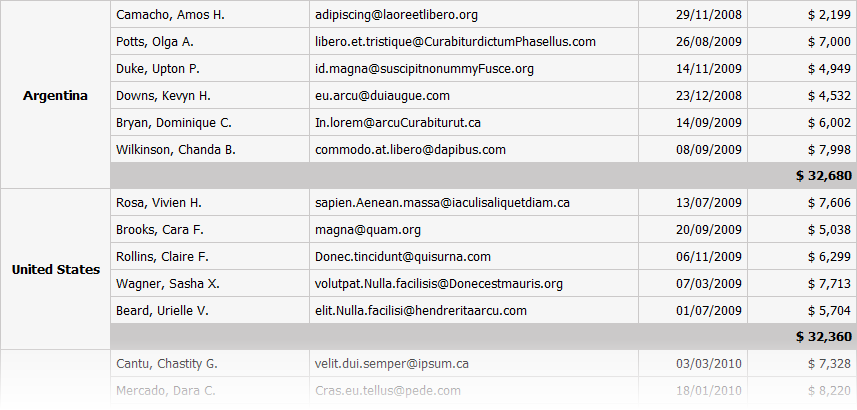
caretaker.pushCommand(cmd);

cmd.commit();

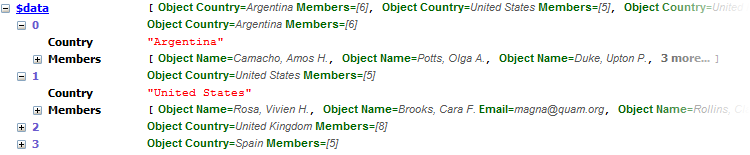
}

});

## Grillas anidadas



Primero se define un modelo que agrupe los datos (donde una propiedad sea otro array de objetos):



Luego se define un esquema de grilla que contenga un sub-esquema para la propiedad que contiene la colección que agrupada elementos (en este caso *Members):*

var $schema = [

{

headerText: "Country",

propertyName: "Country",

type: "string"

},

{

headerText: "Members",

propertyName: "Members",

type: "schema",

schema:[

{

headerText: "Name",

propertyName: "Name",

type: "string"

},

{

headerText: "Email",

propertyName: "Email",

type: "string"

},

{

headerText: "Registration",

propertyName: "DateProp",

type: "date"

}

]

}];

La grilla principal generará eventos por cada sub-grilla que se cree. De esta forma uno se puede subscribir a los eventos generados por las misma sub-grillas:

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

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

var rowData = args.dataItem

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

});

});

#### Soporte de edición en grillas anidadas

La edición en sub-grillas se configura de la misma forma que en cualquier grilla. Sólo se debe agregar ‘*editable: true*’ a los esquemas de columnas que deben ser editables. La [validación](#_Validación_en_edición) se aplica de la misma forma.

var $schema = [

{

headerText: "Country",

propertyName: "Country",

type: "string"

},

{

headerText: "Members",

propertyName: "Members",

type: "schema",

schema:[

{

headerText: "Name",

propertyName: "Name",

type: "string",

editable: true,

validation: DataGrid.Validation.REQUIRED

},

{

headerText: "Email",

propertyName: "Email",

type: "string",

editable: true,

validation: DataGrid.Validation.REQUIRED | DataGrid.Validation.EMAIL

},

{

headerText: "Registration",

propertyName: "DateProp",

type: "date",

editable: true,

validation: DataGrid.Validation.REQUIRED

}

]

}];

#### Soporte de undo/redo

Se aplica de [la misma forma](#_Soporte_de_Undo/Redo) que en un grilla normal, simplemente se overridea el comportamiento por defecto de la grilla, los cambios se encapsulan en un comando y un *caretaker* seencarga de mantener los estados. La única diferencia está en cómo se overridea el comportamiendo en la subgrillas.

// 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();

Ver [ejemplo final](#_Edición_en_grillas).

# Ejemplos

## Grilla sencilla

Ver **sample.grid.simple.htm**

## Grilla compleja

Ver **sample.grid.complex.htm**

## Ordenamiento client-side

Ver **sample.grid.sorting.htm**

## Edición

Ver **sample.grid.edit.simple.htm**

## Edición con validación y undo/redo

Ver **sample.grid.edit.complex.htm**

## Grillas anidadas

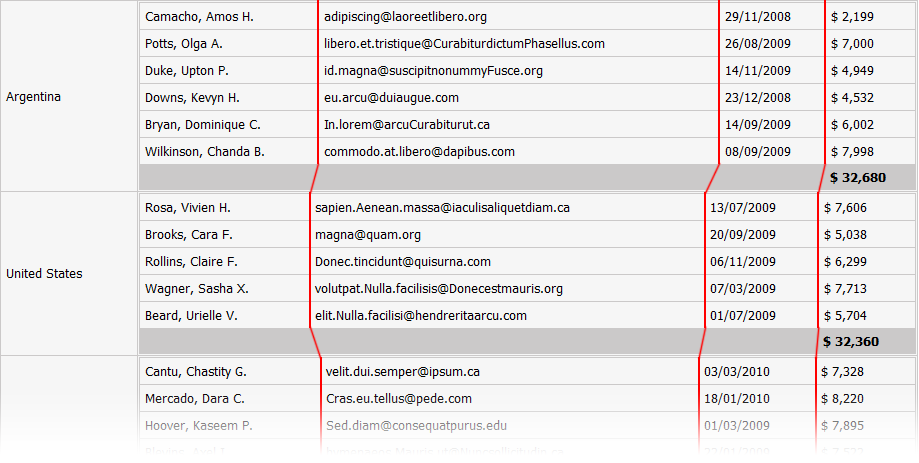
Ver **sample.grid.nested.htm**

## Edición en grillas anidadas

Ver **simple.grid.nested.edit.htm**

# Troubleshooting

## Columnas desiguales en grillas anidadas



Por defecto los navegadores ajustan el tamaño de las celdas de acuerdo el contenido que estas contienen.

La forma de evitar que las columnas se “desacomoden” es especificándoles un ancho, ya sea en píxeles o porcentajes.

***NOTA:*** El componente de grilla intenta separar estructura (datos) de presentación (estilos), por lo que el esquema de tabla no soporta especificar un ‘width’ directamente. Se debe especificar una clases de CSS y desde ella especificar el ancho.

**CSS**

<style type="text/css">

.country-name { width: 180px; font-weight: bold; text-align: center; } /\* ancho máximo \*/

.country-members { width: 100%; padding: 0; }

.member-name { width: 100%; white-space: nowrap; } /\* ancho máximo \*/

.member-email { width: 320px; white-space: nowrap; }

.member-registration { width: 100px; white-space: nowrap; text-align: right; }

.member-account { width: 100px; white-space: nowrap; text-align: right; }

</style>

**Esquema de columnas**

var $schema = [

{

headerText: "Country",

propertyName: "Country",

type: "string",

className: "country-name"

},

{

headerText: "Members",

propertyName: "Members",

type: "schema",

className: "country-members",

schema: [

{

headerText: "Name",

propertyName: "Name",

type: "string",

className: "member-name",

editable: true,

validation: DataGrid.Validation.REQUIRED

},

{

headerText: "Email",

propertyName: "Email",

type: "string",

className: "member-email",

editable: true,

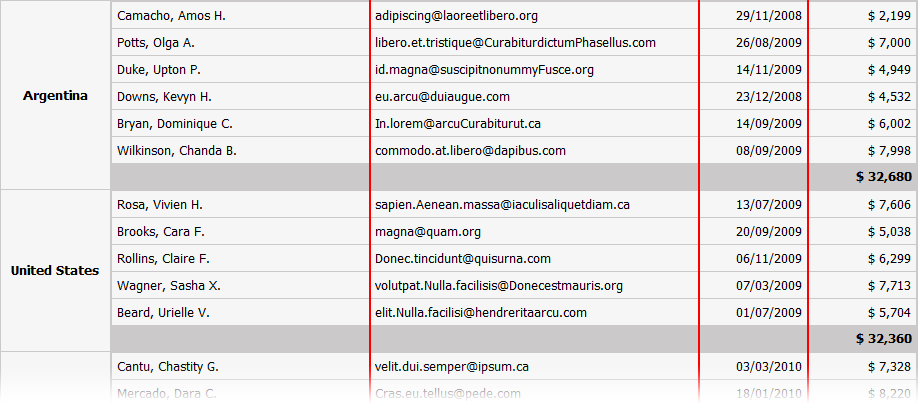
validation: DataGrid.Validation.REQUIRED | DataGrid.Validation.EMAIL

}

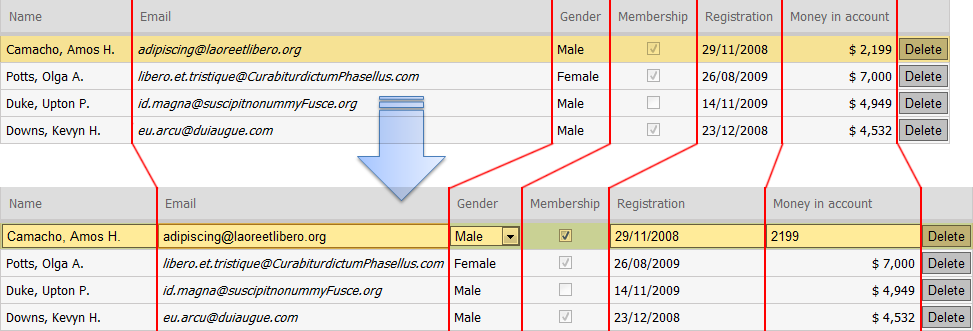
// ... resto de columnas ...

]

}];



## Columnas desacomodadas en modo edición



La misma solución para las [columnas desiguales en grillas anidadas](#_Columnas_desiguales_en) puede aplicar para estos casos.

