put a grid in your life

Plug in to the grid

This is it, the mythical drag-and-drop multi-column grid has arrived. Gridster is a jQuery plugin that allows building intuitive draggable layouts from elements spanning multiple columns. You can even dynamically add and remove elements from the grid. It is on par with sliced bread, or possibly better. MIT licensed. Suitable for children of all ages. Made by **Ducksboard**.

Download now

It's so sweet we like to call it drag-and-drool.

Demos

Usage

Setup

Include dependencies

Gridster is currently written as a jQuery plugin, so you need to include it in the head of the document. Download the latest release at **jQuery**.

HTML Structure

Class names and tags are customizable, gridster only cares about data attributes. Use a structure like this:

```
<div class="gridster">
    <ul>
        <li data-row="1" data-col="1" data-sizex="1" data-sizey="1"></li>
        <li data-row="2" data-col="1" data-sizex="1" data-sizey="1"></li>
        <li data-row="3" data-col="1" data-sizex="1" data-sizey="1"></li>

        <li data-row="1" data-col="2" data-sizex="2" data-sizey="1"></li>
        <li data-row="2" data-col="2" data-sizex="2" data-sizey="2"></li>

        <li data-row="1" data-col="4" data-sizex="1" data-sizey="1"></li>
        <li data-row="2" data-col="4" data-sizex="2" data-sizey="1"></li>
        <li data-row="3" data-col="4" data-sizex="1" data-sizey="1"></li>

        <li data-row="1" data-col="5" data-sizex="1" data-sizey="1"></li>
        <li data-row="3" data-col="5" data-sizex="1" data-sizey="1"></li>

        <li data-row="1" data-col="6" data-sizex="1" data-sizey="1"></li>
        <li data-row="2" data-col="6" data-sizex="1" data-sizey="2"></li>
    </ul>
</div>
```

gridster.html hosted with ❤ by GitHub view raw

Grid it up!

Gridster accepts one argument, a hash of with configuration options. See the **documentation** for details.

Using the API

To get hold of the API object, use the jQuery data method like so:

```
$(function(){ //DOM Ready

    var gridster = $(".gridster ul").gridster().data('gridster');

});
```

gridster.js hosted with ❤ by GitHub view raw

Add a new widget to the grid

```
gridster.add_widget('<li class="new">The HTML of the widget...</li>', 2, 1);
```

gridster.js hosted with ❤ by GitHub view raw

Remove a widget from the grid

```
gridster.remove_widget( $('.gridster li').eq(3) );
```

gridster.js hosted with ❤ by GitHub view raw

Get a serialized array with the elements positions

Creates a JavaScript array of objects with positions of all widgets, ready to be encoded as a JSON string.

```
gridster.serialize();
```

gridster.js hosted with ❤ by GitHub view raw

Documentation

Options

widget_class **widget_margins** **widget_base_dimensions** **min_cols** **max_cols** **min_rows** **max_size_x** **max_size_y** **extra_cols** **extra_rows** **autogenerate_stylesheet** **avoid_overlapped_widgets** **draggable.start** **draggable.drag** **draggable.stop** **resize.enabled** **resize.axes** **resize.handle_class** **resize.handle_append_to** **resize.max_size** **resize.start** **resize.resize** **resize.stop** **collision.on_overlap_start** **collision.on_overlap** **collision.on_overlap_stop**

Methods

These are the most commonly used methods. If you want more details, check out the **documentation generated from source**.

add_widget **resize_widget** **remove_widget** **remove_all_widgets** **serialize** **serialize_changed** **enable** **disable** **enable_resize** **disable_resize**

Options

A gridster configuration object.

widget_selector: *"li"*

Define which elements are the widgets. Can be a CSS Selector string or a jQuery collection of HTMLElements.

widget_margins: *[10, 10]*

Horizontal and vertical margins respectively for widgets.

widget_base_dimensions: *[140, 140]*

Base widget dimensions in pixels. The first index is the width, the second is the height.

extra_rows: *0*

Add more rows to the grid in addition to those that have been calculated.

extra_cols: *0*

Add more rows to the grid in addition to those that have been calculated.

max_cols: *null*

The maximum number of columns to create. Set to `null` to disable.

min_cols: *1*

The minimum number of columns to create.

min_rows: *15*

The minimum number of rows to create.

max_size_x: *false*

The maximum number of columns that a widget can span.

autogenerate_stylesheet: *true*

If true, all the CSS required to position all widgets in their respective columns and rows will be generated automatically and injected to the `<head>` of the document. You can set this to false and write your own CSS targeting rows and cols via data-attributes like so: `[data-col="1"] { left: 10px; }`.

avoid_overlapped_widgets: *true*

Don't allow widgets loaded from the DOM to overlap. This is helpful if you're loading widget positions form the database and they might be inconsistent.

serialize_params: *function($w, wgd) { return { col: wgd.col, row: wgd.row, size_x: wgd.size_x, size_y: wgd.size_y } }*

A function to return serialized data for each each widget, used when calling the **serialize method**. Two arguments are passed: `$w`: the jQuery wrapped HTMLElement, and `wgd`: the grid coords object with keys `col`, `row`, `size_x` and `size_y`.

draggable.start: *function(event, ui){}*

A callback for when dragging starts.

draggable.drag: *function(event, ui){}*

A callback for when the mouse is moved during the dragging.

draggable.stop: *function(event, ui){}*

A callback for when dragging stops.

resize.enabled: *false*

Set to true to enable drag-and-drop widget resizing. This setting doesn't affect to the `resize_widget` method.

resize.axes: *['both']*

Axes in which widgets can be resized. Can be *x*, *y* or *both*

resize.handle_class: *'gs-resize-handle'*

CSS class name used by resize handles.

resize.handle_append_to: *''*

Set a valid CSS selector to append resize handles to. If value evaluates to false it's appended to the widget.

resize.max_size: *[Infinity, Infinity]*

Limit widget dimensions when resizing. Array values should be integers: `[max_cols_occupied, max_rows_occupied]`

resize.start: *function(e, ui, $widget) {}*

A callback executed when resizing starts.

resize.resize: *function(e, ui, $widget) {}*

A callback executed during the resizing.

resize.stop: *function(e, ui, $widget) {}*

A callback executed when resizing stops.

collision.on_overlap_start: *function(collider_data) { }*

A callback for the first time when a widget enters a new grid cell.

collision.on_overlap: *function(collider_data) { }*

A callback for each time a widget moves inside a grid cell.

collision.on_overlap_stop: *function(collider_data) { }*

A callback for the first time when a widget leaves its old grid cell.

.add_widget(html, [size_x], [size_y], [col], [row])

Create a new widget with the given html and add it to the grid.

Parameters

html *String|HTMLElement*

The string of HTMLElement that represents the widget is going to be added.

size_x *Number*

The number of rows that the widget occupies. Defaults to *1*.

size_y *Number*

The number of columns that the widget occupies. Defaults to *1*.

col *Number*

The column the widget should start in.

row *Number*

The row the widget should start in.

Returns

Returns the jQuery wrapped HTMLElement representing the widget that's been created.

.resize_widget($widget, [size_x], [size_y], [reposition], [callback])

Change the size of a widget. Width is limited to the current grid width.

Parameters

$widget *HTMLElement*

The jQuery wrapped HTMLElement that represents the widget is going to be resized.

size_x *Number*

The number of rows that the widget is going to span. Defaults to current size_x.

size_y *Number*

The number of columns that the widget is going to span. Defaults to current size_y.

reposition *Boolean*

Set to false to not move the widget to the left if there is insufficient space on the right. By default `size_x` is limited to the space available from the column where the widget begins, until the last column to the right.

Returns

Returns the jQuery wrapped HTMLElement representing the widget that's been resized.

.remove_widget(el, [callback])

Remove a widget from the grid.

Parameters

el *HTMLElement*

The jQuery wrapped HTMLElement representing the widget that you want to remove.

callback *Function*

A callback for when the widget is removed.

Returns

Returns the instance of the Gridster class.

.serialize([$widgets])

Creates an array of objects representing the current position of all widgets in the grid.

Parameters

$widgets *HTMLElement*

The collection of jQuery wrapped HTMLElements you want to serialize. If no argument is passed all widgets will be serialized.

Returns

Returns an Array of Objects (ready to be encoded as a JSON string) with the data specified by the **serialize_params** option.

.serialize_changed( )

Creates an array of objects representing the current position of the widgets who have changed position.

Returns

Returns an Array of Objects (ready to be encoded as a JSON string) with the data specified in the **serialize_params** option.

.enable( )

Enables dragging.

Returns

Returns the instance of the Gridster class.

.disable( )

Disables dragging.

Returns

Returns the instance of the Gridster class.

Download

Remember that gridster depends on jQuery. Download the latest release at **jQuery**.

gridster.js

Development version

jquery.gridster.js

Production version (minified)

jquery.gridster.min.js

gridster.css

Development version

jquery.gridster.css

Production version (minified)

jquery.gridster.min.css

or clone the repo from github

Github project

gridster.js

Download .zip

gridster.js.zip

Browser support

Gridster supports Internet Explorer 9+, Firefox, Chrome, Safari and Opera.

put a grid in your life

A project by: