Moon Mega Grid

is a multipurpose horizontal masonry grid building solution

- integrated support for canvas and svg filters
- perfect for mobile devices
- simple templating tools
- built-in lightbox with retina support
- 2 retina modes
- memory saving algorithm
- works either with JSON or HTML formats

http://gallery.scalapro.net

https://github.com/Kremlianski/moon-gallery

User Guide

http://gallery.scalapro.net/free-pack/api.html

Content

Events

Lightbox Options

captionHideAfter
name
ns
retina
retinaSuffix
simpleClick
swipe

Lightbox Events

timeForCaption
timeForCaptionHide

Swipe Options

auto
closeOnEnd
continuous
delay
enableClick
enableWheel
firstClass
indicator
lastClass
name
parser
playClass
slideClass
speed
stopClass
swipeClass
swipeWrapClass

Canvas Filters

General Information
grayscale
brightness
sepia
contrast
vibrance
saturate
colorize
noise
Combination of canvas filters
Combination of canvasfilter with SVG filter

How to Start

Upload **dest**, **free-pack**, **lib** folders to the server.

Include jQuery, Underscore, jQuery Scrollstop plugin in your <head> section of the page:

```
<script src="../lib/jquery-3.1.1.min.js"></script>
<script src="../lib/underscore-min.js"></script>
<script src="../lib/jquery.scrollstop.min.js"></script>
```

Include Moon Mega Grid css and js files:

```
<link rel="stylesheet" href="../moon-mega-grid/css/mmg.css">
<script src="../moon-mega-grid/js/moon-mega-grid.min.js"></script>
```

Include css and js files of the template which you prefer:

```
<link rel="stylesheet" href="../grand-pack/classica/light/css/light.min.css">
<script src="../grand-pack/classica/light/js/template.min.js"></script>
```

Create new MMG.Grid.Grid instance in your script.

First (main) way: using the factory function:

```
$(document).ready(function() {
  var MMG, grid;
  MMG = window.MMG; //a global variable
  var options = {
    retina: 2, //the Retina mode is switched on
    grid: '#container', // the root element
    url: 'assets/json/light.json', //the JSON file
    lightbox: {
      retina: true //the Retina mode is switched on for the lightbox
    }
  };
  grid = MMG.Gallery('Light', options); //the new Grid instance is created
});
```

Second way: using the constructor function

If you are not going to develop new template, you don't need to use this way.

```
$(document).ready(function() {
  var MMG, grid;
  MMG = window.MMG; //a global variable
  grid = new MMG.Grid.Grid({ //the new grid instance is created
    retina: 2, //the Retina mode is switched on
    grid: '#container', // the root element
    url: 'assets/json/light.json', //the JSON file
    templateName: 'Light', //the template name
    gridClass: 'light', //this class is added to the root element
    lightbox: {
      retina: true //the Retina mode is switched on for the lightbox
    }
  });
});
```

I recommend you to use the factory function rather than the constructor.

New grid

Constructor

```
var grid = new MMG.Grid.Grid(options);
```

When you develop new template from scratch, every option is meen something for you. See the list of options below.

But if you use ready template from Grand Pack, some options are the part of this template and you shouldn't change them. So it's better to use the factory function in this situation.

Factory function

```
var grid = MMG.Gallery('templateName', options, grayscale);
```

@param {String} templateName - a name of the template

@param {Object} options

@param {Boolean} grayscale - if true, the script will create the SVG grayscale filter element and append it to the body. Default true.

@return {MMG.Grid.Grid}

It's the simpliest way if you use a template from Grand Pack.

The function automatically specifies options which are needed for the chosen template, and you don't have to do it.

So if to compare with the constructor function, the number of options will be smaller.

Moreover, if the grayscale filter is needed, the factory function will create an appropriate SVG element and insert it to the body. To switch off this option add *false* as the third attribute to the function.

Options

canvas

type: 0, 1, 2

default: 0

the 'canvas' mode

0 - the 'canvas' mode is disabled

1 - the 'canvas' mode is enabled

2 - the 'canvas' mode is enabled for mobile devices

When the 'canvas' mode is enabled, the image is replaced with canvas element.

canvasFilters

type: Array

default: undefined

The array of specified canvas filters

see Canvas Filters

data

type: Object

default: undefined

The initial data. Must be specified in the script.

When specified, the 'data' mode is switched on and the grid data must be got from this data object

excludable

type: Boolean

default: undefined

If defined as **true** it lets some images be opened as links

See how to

excludeClass

type: String

default: 'mmg-external'

This string must be added to the 'classList' attribute of the json when we need to prevent from opening the lightbox when we click on an icon.

gridClass

type: String

default: undefined

The class that is added to the root element

example:

```
gridClass: 'morning'
```

jsonParser

type: Function

default: undefined

If defined it will be used to convert JSON object

Takes JSON object as a single parameter

Returns an array of two elements:

The first element is an array of item data objects (href, src, title ...)

The second one is an array of metadata

If you don't need metadata you can choose that the function returns only

an array of item data objects (href, src, title ...)

example:

```
jsonParser: function(data) {
  return _.map(data, function(item){
    return {
      href: item.imageHref,
      src: item.imageSrc,
      title: item.description
    };
  });
}
```

See examples

height

type: Integer

default: undefined

The maximum height of a row. When this option is not defined, the script will take a natural height of images as a base of calculation.

example:

```
height: 180
```

lightbox

type: false or Object

default: undefined

lightbox options

false - the built-in lightbox is disabled

undefined - the built-in lightbox is enabled with default options and without caption

Object - the built-in lightbox is enabled with specified options

see Lightbox Options

margin

type: Integer or '0'

default: 2

margin-right, margin-bottom css properties

example:

```
margin: 18
```

middle

type: Integer

default: 400

The maximum width for images which can be classified as 'middle'.

When the image is classified as 'middle', the class 'mmg-middle' is added to the outer element with the class 'mmg-img'

example:

```
middle: 300
```

mobileGridClass

type: String

default: undefined

The class that is added to the root element for mobile devices

example:

```
mobileGridClass: 'm-morning'
```

ns

type: String

default: 'mmg-'

The namespace prefix, that will be added to css-classes of the grid elements

example:

```
ns: 'alex-'
```

oldIEFilter

type: 'canvas', 'css', 'none'

default: undefined

It's for IE9. These browser have its own filters. Moreover IE9 supports canvas.

'none' or **undefined** - the 'twin' mode is disabled and no filtering at all

'css' - if the 'twin' mode is disabled the script will add a class 'mmg-filtered' to the image element, if the 'twin' mode is enabled the script will create an other image element and will add a class 'mmg-filtered' to it.

'canvas' - for IE9 canvas filters will be applied.

example:

```
oldIEFilter: 'css'
```

parser

type: Function

default: undefined

The function that is used as a parser.

When specified and the 'html' mode is switched on, the script will parse the content of the root element.

The 'html' mode is enabled unless 'url' or 'data' are specified.

example:

```
parser: myParser
```

retina

type: 0, 1, 2

default: 0

0 - the Retina mode is disabled

1 - When script loads image, it calculates its size as a half of a natural size.

2 - When the device has a Retina screen, the script append the Retina suffix '@2x' to the image name and calculates the image size as a half of a natural size.

example:

```
retina: 1
```

retinaSuffix

type: String
default: '@2x'

Overrides the default retina suffix ('@2x')

example:

```
retinaSuffix: '-r1'
```

small

type: Integer

default: 180

The maximum width for images which can be classified as 'small'.

When the image is classified as 'small', the class 'mmg-small' is added to the outer element with the class 'mmg-img'

example:

```
small: 150
```

svgFiltersId

type: String

default: undefined

The ID of the appropriate SVG-filter

example:

SVG Filters

templateName

type: String

default: 'Simple'

The name of the template which is used

example:

```
templateName: 'Paris'
```

timeout

type: Integer

default: 5000

The maximum loading time in ms

example:

```
timeout: 3000
```

twin

type: Boolean

default: false

The 'twin' mode. If some filters are used and the twin mode is enabled, the image itself isn't removed. Thus both the result of filtering (SVG or canvas) and the image itself are staying in one parent element.

example:

```
twin: true
```

url

type: String

default: undefined

An Url of JSON file that is used for data loading by AJAX.

When specified, the 'json' mode is switched on.

example:

```
url: 'assets/json/grid.json'
```

vars

type: Object

default: undefined

An object, that contains custom variables.

Pares of keys and values.

example:

```
vars: {
  delay: 0,
  minus: true
}
```

Methods

getExternalLightbox(lb, options, cbs)

@param {String} lb — may be: 'colorbox', 'photoswipe'

@param {Object} options — native lightbox options

@param {Object} cbs — callback functions

@return lightbox object

Factory function for creating colorBox or photoSwipe

ColorBox example

```
grid.getExternalLightbox('colorbox',
{
  maxWidth: '100%',
  maxHeight: '100%'
},
{
  getTitle: function(item){
    return item.lb.title;
  }
});
```

You can define folowing callbacks:

getTitle — returns the caption title

getHref — returns the src of the image to be shown
default: item.href

PhotoSwipe example

```
grid.getExternalLightbox('photoswipe', {}, {
```

```
  getWidth: function(item) {return item.lb.width;},
  getHeight: function(item) {return item.lb.height;},
  getTitle: function(item) {return item.lb.title;}

// if you needn't msrc attribute:
// getMsrc: function(item) {return void(0);}

});
```

You can define folowing callbacks:

getTitle — returns the caption title

getHref — returns the src of the image to be shown
default: item.href

getMsrc — returns the src of the appropriate icon
default: item.src

getHeight — returns the height of the image. Required

getWidth — returns the width of the image. Required

more about external lightboxes

getLastLoadedMeta()

@public

@return object

This public method returns metadata object that is stored in the lastLoadedMeta private parameter.

This metadata object can be fetched (not necessarily) when JSON object is parsed using the jsonParser callback function.

loadByAjax(url, urlData, type)

@param {String} url

@param {Object} urlData — The object of data or {}

@param {String} type — 'json'(default) or 'html'

Loads additional items by AJAX

example

```
grid.loadByAjax('json/load2.json');
```

example

```
grid.loadByAjax('html/load.html', {}, 'html');
```

Events

dataLoaded

```
$('#container').on('dataLoaded', function(event, data){
  console.log(data.all);
});
```

A 'dataLoaded' event is fired when the object, that contain information of all items has been formed. This situation happens when the script gets data from ajax or from markup and the structure of the grid is being built or being rebuilt.

This event is fired by the root element of the grid.

The plain object of params is passed to the callback function as the second attribute. It consists of single element 'all', which value is a object containing all items of the grid.

afterLoad

```
$('#container').on('afterLoad', function(event, data){
  console.log(data.all);
});
```

The 'afterLoad' event is fired when the script got data from ajax or from markup and the new structure of the grid was built or rebuilt.

This event is fired by the root element of the grid.

The plain object of params is passed to the callback function as the second attribute. It consists of single element 'all', which value is a object containing all items of the grid.

The difference from the 'dataLoaded' event:

The 'dataLoaded' event is fired as soon as the data object is formed. So we can use this object right after it was formed.

The 'afterLoad' event is fired when new rows were created. So we can use new markup.

Lightbox Options

captionHideAfter

type: Integer
default: undefined

The caption will be hidden after captionHideAfter milliseconds.

The option is not applicable in the swipe mode.

name

type: String
default: undefined

The lightbox template name. If is undefined, no lightbox template is used and no caption will be shown as a result.

You can use the 'default' template name, or you have to create your own lightbox template.

The option is not applicable in the swipe mode.

ns

type: String

default: 'mmg-'

The namespace prefix, that will be added to css-classes of the lightbox elements.

retina

type: Boolean
default: false

When the device has a Retina screen, the script append the Retina suffix '@2x' to the image name and calculates the image size as a half of a natural size.

retinaSuffix

type: String
default: '@2x'

Overrides the default retina suffix ('@2x')

simpleClick

type: Boolean
default: true

The user interface of the lightbox is very simple. And I think it's good. Thus if you need to close the lightbox panel, you may click somewhere inside the window. I named it the **'simple click'** mode.

When the 'simple click' mode is switched off and you click on the lightbox panel the lightbox will not be closed.

The option is not applicable in the swipe mode.

swipe

type: Boolean or Object
default: false

A user can choose the swipe mode.

When the value is defined as true the default options will be applied.

See Swipe options for more details

example

```
lightbox: {
  swipe: true
}
```

Lightbox Events

timeForCaption

This event is fired when it's time for the caption to appear.

It doesn't work in the swipe mode.

example

```
meta.root.on('timeForCaption', function(event, caption) {
  $('.mmg-lb-socials a, .mmg-lb-socials, .mmg-lb-captions', caption)
    .stop(true)
    .delay(200)
    .animate({
      opacity: 1
    });
});
```

timeForCaptionHide

This event is fired when it's time for the caption to disappear.

It doesn't work in the swipe mode.

example

```
meta.root.on('timeForCaptionHide', function(event, caption) {
  $('.mmg-lb-socials a, .mmg-lb-socials, .mmg-lb-captions', caption)
    .stop(true)
    .animate({
      opacity: 0
    });
});
```

Swipe Options

auto

type: Boolean
default: false

If true, the slideshow will start automatically

example

```
lightbox: {
  swipe: {
    auto: true
  }
}
```

closeOnEnd

type: Boolean
default: false

if true and if continuous option is false and when the slideshow is in action, the lightbox will be closed when the last slide is shown.

example

```
lightbox: {
  swipe: {
    continuous: false,
    closeOnEnd: true
  }
}
```

continuous

type: Boolean
default: true

It creates an infinite feel with no endpoints

example

```
lightbox: {
  swipe: {
    continuous: false
  }
}
```

delay

type: Inreger
default: 4000

Duration of showing a slide in slideshow (ms)

example

```
lightbox: {
  swipe: {
    delay: 2000
  }
}
```

enableClick

type: Boolean
default: true

switches on/off click event support

example

```
lightbox: {
  swipe: {
    enableClick: false
  }
}
```

enableWheel

type: Boolean
default: true

switches on/off mouse wheel support

example

```
lightbox: {
  swipe: {
    enableWheel: false
  }
}
```

firstClass

type: String
default: 'swipe-first-slide'

The class that is added to the swipe container if the first slide is shown and the continuous option is false

example

```
lightbox: {
  swipe: {
    continuous: false,
    firstClass: 'my-first-class'
  }
}
```

indicator

type: Boolean
default: true (if swipe presetting is 'simple' or 'minimal', the indicator option is false)

should indicator be shown

example

```
lightbox: {
  swipe: {
    indicator: false
  }
}
```

lastClass

type: String
default: 'swipe-last-slide'

The class that is added to the swipe container if the last slide is shown and the continuous option is false

example

```
lightbox: {
  swipe: {
    continuous: false,
    firstClass: 'my-last-class'
  }
}
```

name

type: String ('classica', 'minimal', 'simple', 'vertical') or function
default: 'classica'

The name of presetting

example

```
lightbox: {
  swipe: {
    name: 'vertical'
  }
}
```

parser

type: Function
default:

```
function(data) {
  return _.map(data, function(item) {
    var ref;
    return {
      href: item.href,
      title: ((ref = item.lb) != null ? ref.title : void 0) || item.title
    };
  });
}
```

The parser function

playClass

type: String
default: 'swipe-play'

The class that is added to the play/stop button if the slideshow starts

example

```
lightbox: {
  swipe: {
    playClass: 'my-play-class'
  }
}
```

slideClass

type: String
default: 'slide'

The class of the slide element

example

```
lightbox: {
  swipe: {
    slideClass: 'my-slide'
  }
}
```

speed

type: Inreger
default: 400

speed of prev and next transitions in milliseconds

example

```
lightbox: {
  swipe: {
    speed: 600
  }
}
```

stopClass

type: String
default: 'swipe-stop'

The class that is added to the play/stop button if the slideshow has been stoped

example

```
lightbox: {
  swipe: {
    stopClass: 'my-stop-class'
  }
}
```

swipeClass

type: String
default: 'swipe'

The class of the swipe element

example

```
lightbox: {
  swipe: {
    slideClass: 'my-swipe'
  }
}
```

swipeWrapClass

type: String
default: 'swipe-wrap'

The class of the slides wrapper element

example

```
lightbox: {
  swipe: {
    slideClass: 'my-wrapper'
  }
}
```

Canvas Filters

General Information

You can apply one or several canvas filters from the list.

Some filters have additional options, some haven't.

You must specify the list of filters in form of array of arrays.

The first element of every array is a name of the filter. The second element is an array of options (if any).

grayscale

There are no additional options.

example:

```
canvasFilters: [['grayscale']]
```

brightness

option: adjust (from -100 to 100)

example:

```
canvasFilters: [['brightness', [20]]]
```

sepia

There are no additional options.

example:

```
canvasFilters: [['sepia']]
```

contrast

option: adjust (from -100 to 100)

example:

```
canvasFilters: [['contrast', [20]]]
```

vibrance

option: adjust (from -100 to 100)

example:

```
canvasFilters: [['vibrance', [20]]]
```

saturate

option: adjust (from -100 to 100)

example:

```
canvasFilters: [['saturate', [20]]]
```

colorize

options:

red (from 0 to 255)

green (from 0 to 255)

blue (from 0 to 255)

adjust (from 0 to 100)

example:

```
canvasFilters: [['colorize', [25, 150, 50, 25]]]
```

noise

option: adjust (from 0 to 100)

example:

```
canvasFilters: [['noise', [25]]]
```

Combination of canvas filters

example:

```
canvasFilters: [['colorize', [25, 150, 50, 25]], ['brightness', [-15]],
['contrast', [-15]], ['vibrance', [50]]]
```

Combination of canvas filter with SVG filter

SVG filters are very good. But unfortunately old versions of Android don't support SVG.

You can use combination of SVG filter and canvas filter to solve this problem.

example:

```
canvasFilters: [['grayscale']], // the canvas filter array
svgFiltersId: 'grayscale', //the ID of the SVG filter
```