Marionette.RegionManager

Region managers provide a consistent way to manage a number of Marionette.Region objects within an application. The RegionManager is intended to be used by other objects, to facilitate the addition, storage, retrieval, and removal of regions from that object. For examples of how it can be used, see the Marionette.Application and Marionette.LayoutView objects.

Documentation Index

Basic Use

RegionManagers can be instantiated directly, and can have regions added and removed via several methods:

```
var rm = new Marionette.RegionManager();

var region = rm.addRegion("foo", "#bar");

var regions = rm.addRegions({
  baz: "#baz",
  quux: "ul.quux"
});

regions.get('baz').show(myView, options);

rm.removeRegion("foo");
```

▸ Constructing

The RegionManager take an optional `region` option in their constructor. the regions are passed directly into `addRegions` for the region manager instance.

```
var manager = new Marionette.RegionManager({
  regions: {
    "aRegion": "#bar"
  }
});

manager.get('aRegion').show(new MyView, options);
```

▸ RegionManager.addRegion

Regions can be added individually using the `addRegion` method. This method takes two parameters: the region name and the region definition.

```
var rm = new Marionette.RegionManager();

var region = rm.addRegion("foo", "#bar");
```

In this example, a region named "foo" will be added to the RegionManager instance. It is defined as a jQuery selector that will search for the `#bar` element in the DOM.

There are a lot of other ways to define a region, including object literals with various options, and instances of Region objects. For more information on this, see the Region documentation.

▸ RegionManager.addRegions

Regions can also be added en-masse through the use of the `addRegions` method. This method takes an object literal or a function that returns an object literal. The object literal must contain region names as keys and region definitions as values. The return value is an object literal with all the created regions.

```
var rm = new Marionette.RegionManager();

// With an object literal
var regions = rm.addRegions({
  main: '#main-content',
  navigation: {
    selector: '#navigation',
```

```
    regionClass: MyNavRegion
  }
});

// With a function
var otherRegions = rm.addRegions(function(regionDefinition) {
  return {
    footer: '#footer'
  };
});

regions.get('main');        //=> 'main' region instance
regions.get('navigation');  //=> 'navigation' region instance
otherRegions.get('footer'); //=> 'footer' region instance
```

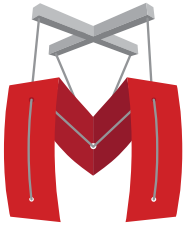
If you supply a function to `addRegions`, it will be called with the `RegionManager` instance context and all the arguments passed to `addRegions`.

```
var rm = new Marionette.RegionManager();

var regionDefaults = {
  regionClass: MyRegionClass
};

rm.addRegions(function(regionDefinition, defaults) {
```

```
  console.log(this);             // `rm` instance of `RegionManager`
  console.log(regionDefinition); // the region definition function
  console.log(defaults);         // `{ regionClass: MyRegionClass }`

  // ...return the region definiton object literal
}, regionDefaults);
```

▸ addRegions default options

When adding multiple regions it may be useful to provide a set of defaults that get applied to all of the regions being added. This can be done through the use of a `defaults` parameter. Specify this parameter as an object literal with `key: value` pairs that will be applied to every region added.



v2.4.4


```
var rm = new Marionette.RegionManager();

var defaults = {
  regionClass: MyRegionClass
};

var regions = {
  foo: "#bar",
  baz: "#quux"
```

```
};

rm.addRegions(regions, defaults);
```

In this example, all regions will be added as instances of `MyRegionClass`.

▸ RegionManager.get

A region instance can be retrieved from the RegionManager instance using the `get` method and passing in the name of the region.

```
var rm = new Marionette.RegionManager();
rm.addRegion("foo", "#bar");

var region = rm.get("foo");
```

▸ RegionManager.getRegions

Get all the regions from the region manager. Returns an object literal with named regions as attributes.

```
var rm = new Marionette.RegionManager();
rm.addRegion("foo", "#foo");
rm.addRegion("bar", "#bar");

var regions = rm.getRegions();

regions.foo; //=> foo region
regions.bar; //=> bar region
```

▸ RegionManager.removeRegion

A region can be removed by calling the `removeRegion` method and passing in the name of the region.

```
var rm = new Marionette.RegionManager();
rm.addRegion("foo", "#bar");

rm.removeRegion("foo");
```

A region will have its `empty` method called before it is removed from the RegionManager instance and `stopListening` is called.

▸ RegionManager.removeRegions

You can quickly remove all regions from the RegionManager instance by calling the `removeRegions` method.

```
var rm = new Marionette.RegionManager();
rm.addRegions({
  foo: "#foo",
  bar: "#bar",
  baz: "#baz"
});

rm.removeRegions();
```

This will empty all regions, and remove them.

▸ RegionManager.emptyRegions

You can quickly empty all regions from the RegionManager instance by calling the `emptyRegions` method.

```
var rm = new Marionette.RegionManager();
rm.addRegions({
  foo: "#foo",
  bar: "#bar",
  baz: "#baz"
```

```
});

rm.emptyRegions();
```

This will empty the regions without removing them from the RegionManager instance.

▸ RegionManager.destroy

A RegionManager instance can be destroyed entirely by calling the `destroy` method. This will both destroy and remove all regions from the RegionManager instance.

```
var rm = new Marionette.RegionManager();
rm.addRegions({
  foo: "#foo",
  bar: "#bar",
  baz: "#baz"
});

rm.destroy();
```

▸ RegionManager Events

A RegionManager will trigger various events as it

is being used.

▸ before:add:region event

The `RegionManager` will trigger a "before:add:region" event before a region is added to the manager. This allows you to perform some actions on the region before it is added.

```
var rm = new Marionette.RegionManager();

rm.on("before:add:region", function(name, region){
  // do something with the region instance
});

rm.addRegion("foo", "#bar");
```

▸ add:region event

The RegionManager will trigger a "add:region" event when a region is added to the manager. This allows you to use the region instance immediately, or attach the region to an object that needs a reference to it:

```
var rm = new Marionette.RegionManager();
```

```
rm.on("add:region", function(name, region){
  // add the region instance to an object
  myObject[name] = region;
});

rm.addRegion("foo", "#bar");
```

▸ before:remove:region event

The `RegionManager` will trigger a "before:remove:region" event before a region is removed from the manager. This allows you to perform any cleanup operations before the region is removed.

```
var rm = new Marionette.RegionManager();

rm.on("before:remove:region", function(name, region){
  // do something with the region instance here
});

rm.addRegion("foo", "#bar");

rm.removeRegion("foo");
```

▸ remove:region event

The RegionManager will trigger a "remove:region" event when a region is removed from the manager. This allows you to use the region instance one last time, or remove the region from an object that has a reference to it:

```
var rm = new Marionette.RegionManager();

rm.on("remove:region", function(name, region){
  // add the region instance to an object
  delete myObject[name];
});

rm.addRegion("foo", "#bar");

rm.removeRegion("foo");
```

▸ RegionManager Iterators

The RegionManager has several methods for iteration attached to it, from underscore.js. This works in the same way as the Backbone.Collection methods that have been imported. For example, you can easily iterate over

the entire collection of region instances by calling the `each` method:

```
var rm = new Marionette.RegionManager();

rm.each(function(region){
  // do stuff w/ the region instance here
});
```

The list of underscore methods include:

forEach

each

map

find

detect

filter

select

reject

every

all

some

any

include

contains

invoke

toArray

first

initial

rest

last

without

isEmpty

pluck