

RequireJS in Octopeer Analytics

For modularisation and ease of programming and debugging, RequireJS is used. This library makes it possible to load in scripts as modules and internally handles module dependencies. The primary use case of the library is for creating, loading and inserting visualisations such as graphs, charts, etc. We desired to create a standard for creating visualisation modules. In this document we will describe the process of creating and loading in modules in our application.

The modules are JavaScript(js) files with a special RequireJS syntax. These js files are placed in the modules folder under src. Due to security reasons, the js files can't be read by the application directly and need to be added to a list of which modules to load. This list is maintained in moduleList.js under src/modules. After creating a module and adding it to the modules folder, the string 'module/{filename}', where {filename} is the name of the js file without the '.js' extension, is added to the list between the '[' and the ']'. Module names in this list are separated by ',' (comma's) and the position of the strings determine the order in which the modules will be loaded. Now let's see how to create a module.

As has been stated earlier, the module files are js files placed in the modules directory under src. The syntax for these files is as follows :

```
define(function () {  
    return {  
        field1 : value1,  
        field2 : value2,  
        field3 : ...  
    };  
});
```

For a module to be loaded correctly, the following fields have to be defined :

Definition

name : *string* (required)

The name of the module. This value needs to be unique for every module for the program to function correctly as this name is used as the id of the encapsulating div for this module. We recommend the value of this field to be the same as the name of the file.

title : *string* (required)

The title of the card the visualisation resides in. This is what the user sees. If the title is too long to fit inside the card, it will be truncated.

size : *int* (required)

This is the size of the module. At the moment, the following values are supported:

1. ½ width

parentSelector : *string*

The category to add the module to. This is the css selector of the location the module is added to, as used in jQuery, e.g. 'body > div#moduleContainer'. Make sure that the selector is specific enough to point to a single element. Currently, this value defaults to div#modules.

Content

data : *Object*

For Ajax calls for retrieving external data, define this field with a JSON object with the following format :

```
[
  {
    'serviceCall' : serviceCall
    'onSuccess' : onSuccess
    'onFailure' : onFailure
    'onComplete' : onComplete
    'required' : required
  },
  { ...
}
```

Where ...

serviceCall (*function* ()) is a function that returns a promise for a data call with a JSON response value. These can be newly created or the following services and their function can be used :

- octopeerService
 - getPullRequestsAmount()
 - getUserAmount()
 - ...
- githubService
 - ...
- bitbucketService
 - ...

An example value for this field could be *function* () { *return octopeerService.getUserAmount*() } .

onSuccess (*function* (*data* = {})) is the function to be executed after the Ajax call is completed successfully.

onFailure (*function* (*data* = {})) is the function to be executed after the Ajax call fails.

onComplete (*function* (*data* = {})) is the function to be executed after the Ajax call is completed.

required (*boolean* = *true*) is whether or not the failure of this call should put the module in a failed state.

Multiple Ajax calls are supported. The values inside the brackets, e.g. (*boolean = true*), are the default values.

body : *function* (*data* = []) : *Object* (required)

In this function, an HTML group (<g> </g>) element is created and filled through JavaScript. The created element is returned in the form of a D3 encapsulated element object. This element can be created with *d3.select(document.createElementNS(d3.ns.prefix.svg, "g"))*;

Inside the modules, it is possible to use jQuery and D3 functions.

If the module requires external data, the *data* field has been defined and all Ajax calls have been completed successfully, the obtained data is passed to this function as an argument to use the data while creating the element. As multiple Ajax calls can be created from one module, this argument is in the form of an array in which the (JSON) responses are stored. The order of the objects in this array corresponds with the order of the defined Ajax calls.

failBody : *function* (*status* = []) : *Object*

If external data is required for the module to function, the standard body might not be wanted to be added to the DOM if the call to the external data has failed. In this case, this function can be defined. On an external call fail, this Object will be added to the DOM instead of the standard *body()*. The error, error state and error message of each Ajax call is passed on to this function as an argument in the form of an array. The order of the objects in this array corresponds with the order of the defined Ajax calls.

customContainer : *boolean*

If not defined or if false, the elements returned from *body()* and *failBody()* will be added to the DOM inside a standard container, currently an svg inside a div with class 'card module'. If the standard elements stylings of this container is not desired for this module, set this value to true and return the elements from *body()* and *failBody()* with their containers, e.g. instead of returning a <g></g> element, return a div with id 'moduleX' and class 'xs-col-3' with the svg element inside of it.

Be aware that if this functionality is used, only the *name*, *parentSelector*, *data*, *body* and *failBody* fields can be used.

Axes

xAxis : *boolean*

yAxis : *boolean*

yRightAxis : *boolean*

Whether the axis needs to be drawn. *xAxis* and *yAxis* default to true, while *yRightAxis* defaults to false.

xAxisLabel : *string*

yAxisLabel : *string*

yRightAxisLabel : *string*

The label of the axis. If defined, this value will be drawn next to or under the axis as the axis name.

xAxisLine : *boolean*

yAxisLine : *boolean*

yRightAxisLine : *boolean*

Whether a line must be drawn for the axis. When false, only ticks and tick labels will be visible. Defaults to true.

xAxisTicks : *boolean*

yAxisTicks : *boolean*

yRightAxisTicks : *boolean*

If true, tick lines are drawn in the background of the graph.

xAxisLabelRotation : *int*

yAxisLabelRotation : *int*

yRightAxisLabelRotation : *int*

The rotation of the tick labels of an axis. If not defined, defaults to 0 and the labels are drawn horizontally.

xAxisScale : *function* () : "fit" or *d3.scale*

yAxisScale : *function* () : "fit" or *d3.scale*

yRightAxisScale : *function* () : "fit" or *d3.scale*

The values for the tick labels. Needs to be a function that either returns "fit" or a *d3.scale* object. When "fit" is returned by the function, the axis is initially labeled from 0 to 100 and after the data for the graph (defined in the *data* field) is received, the axis is scaled according to the data with the use of the *AxisFitFunction* below.

xAxisFitFunction : *function* (*data* = []) : [*int*, *int*]

yAxisFitFunction : *function* (*data* = []) : [*int*, *int*]

yRightAxisFitFunction : *function* (*data* = []) : [*int*, *int*]

If the *AxisScale* functions return "fit", this function is called with the data call response to rescale the axis. The function must return an array of 2 ints, [from :int , to :int].

Legends

legend : *Object*

If legends are desired to be added to the created graphs, they can be defined inside the modules with the following syntax:

```
[
  {
    'type' : type
    'style' : style
    'text' : text
  },
```

```
] { ...
```

Where...

type (string) is a string defining the type of the legend item to be drawn. Can be one of the following:

line - Draws a line

dot - Draws a circle with radius 3

linedot - Draws a line with a dot in the middle

rect - Draws a rectangle

style (string) is a string in css syntax. This is the style the legend item is given. With this, the fill and stroke colors can be defined, as well as the stroke-width and stroke dashing.

text (string) is a string defining the text to be written next to the legend item.

The fields and values in this file might be incomplete or might change in the future. All changes to the code will be followed by an update to this file.