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 seperated 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 :

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.

size : *int* (required)
This is the size of the module. At the moment, the following values are supported:

1. full width
2. 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.

ajax : *Object*

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

```
{'ajax':[
        {
                'callType' : callType
                'url' : url
                'data' : data
                'type' : type
                'dataType' : dataType
                'async' : async
                'timeout' : timeout
                'onSuccess' : onSuccess
                'onFailure' : onFailure
                'onComplete' : onComplete
                'required' : required
        },
        { ...
]}
```

Where …

callType (*string* = "*ajax*") is either "ajax", "github" or "bitbucket". If "github" or "bitbucket", some parameters to access the Github or Bitbucket API's will be set automatically.

url (*string*) is the url for the Ajax request if *callType* = "*ajax*".

data (*Object* = {}) is the data to be sent with the request as a JSON Object.

type (*string* = "*GET*") is the type of request; GET or POST if *callType* = "*ajax*".

dataType (*string* = "*json*") is the dataType the Ajax call expects to be returned if *callType* = "*ajax*".

async (*boolean* = *true*) is the synchronicity of the call. False for synchronous.

timeout (*int* = *null*) is the timeout for the Ajax call. If exceeded, the call fails.

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 fail the complete request.

Multiple Ajax calls are supported. The values inside the brackets e.g. $(string = "json")$ are the default values.

$body : function(data = []) : Object$ (required)
In this function, an HTML element is created and filled through JavaScript. The created element is returned in the form of a standard JavaScript object. Inside the modules, it is possible to use jQuery and D3 functions. If the D3 library is used for the creation of the element to be returned, return the element with 'return {element}[0]', where {element} is the name of the element to be returned. This is because D3 handles elements as objects in a list, while the module loader uses jQuery's appendTo() function on the returned elements which uses objects as data types for the elements.
If the module requires external data, the $ajax$ 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, the elements returned from $body()$ and $failBody()$ will be added to the DOM inside a standard container, currently a div with class '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 an svg element, return a div with id 'moduleX' and class 'xs-col-3' with the svg element inside of it.

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.