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. The following values are supported:
1 - small (165x165)
2 - medium (350x350)
3 - large (720x350)

The exact sizes might change in the future. Also, more sizes might be added. All changes will be followed by an update to this file.

ajax : *string*
If the module needs external data, this may be acquired through the use of an Ajax call. In this case, define the Ajax call url in this field. This field is only required when an Ajax call is needed.

More ajax functions such as data, type and datatype will be implemented soon.

noAjax : *boolean*
If the module does not need an Ajax call and is self-contained, this field needs to be set to true. This field is only required when no Ajax call is needed.

body : *function* (*data* = []) : *Object* (required)
This is where the magic happens. 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.