NORRIS Framework

FLAMETECH Inc.

Developer Manual

Document information

Version	1.0.0
Editors	Sartor Michele
Reviewers	Cardin Andrea
Accountable	Merlo Gianluca
Use	External
Distribution list	**FlameTech Inc.**
	Prof. Vardanega Tullio
	Prof. Cardin Riccardo
	CoffeeStrap

Description

Developer manual for Norris framework

Version	Modification	Author	Role	Date	Status
1.0.0	Document approved	Merlo Gianluca	Project Manager	2015/06/05	Approved
0.1.0	Document reviewed	Cardin Andrea	Reviewer	2015/06/04	Reviewed
0.0.5	Translated appendix A and B	Sartor Michele	Administrator	2015/06/04	Processing
0.0.4	Translated section 5	Sartor Michele	Administrator	2015/06/03	Processing
0.0.3	Translated section 3 and 4	Sartor Michele	Administrator	2015/06/02	Processing
0.0.2	Translated section 1 and 2	Sartor Michele	Administrator	2015/06/02	Processing
0.0.1	Beginning translation	Sartor Michele	Administrator	2015/06/02	Processing

Contents

1 Introduction

1.1 Document purpose

This document represents the developer manual for the Norris *framework*$_G$; it exposes all the features that the product offers and explains the correct ways to use the *framework*$_G$.

1.2 Product purpose

The product purpose is the production of a *Node.js*$_G$ *framework*$_G$ which is compatible with the standard use of *Express*$_G$ version 4.x *middlewares*$_G$.
This *framework*$_G$ will be used to rapidly create charts and update them in real time.

2 System requirements

Norris *framework*$_G$ is compatible with every operating system that supports *Node.js*$_G$ and *npm*.
Requirements:

- *Node.js*$_G$ version 0.12.2 or later;
- *npm*$_G$ version 2.7.4 or later.

The installation requires a command-line interface for executing *npm* and an internet connection. Additional dependencies will be automatically resolved when installing the *framework*$_G$.
The supported browsers are:

- **Chrome** version 38.0.X and later;
- **Firefox** version 32.X and later.

3 Installation

3.1 *Framework*$_G$ installation

Norris *framework*$_G$ can be installed with the following command line:

```
npm install norris-rtbi
```

In order to create a new project, it is also required to install *Express*$_G$ and *socket.io*$_G$ with the following command lines:

```
npm install express
npm install socket.io
```

Also, the front-end part of Norris *framework*$_G$ requires Bower to be installed globally on the system.
Therefore, if Bower is not installed already, the following command line is also required:

```
npm install -g bower
```

Additional dependencies will be automatically resolved when installing the *framework*$_G$.
For more information about creating a new project, see section 4.

4 Project configuration

4.1 Required libraries and instantiation

In order to use the features of Norris *framework*$_G$, two libraries are required for a correct execution. Specifically, it is required to have an instance of:

- ***Socket.io***$_G$: an exclusive *namespace*$_G$ that will be used to send *push*$_G$ notifications and update single charts;
- ***Express***$_G$: required to create a *mount*$_G$ point for the correct integration of the *framework*$_G$ with the application which uses its features and also to make the resources available to web requests.

The following example is a correct way to configure a project.

```
// Express instantiation
var app = require('express')();

// Socket.io instantiation on the express instance
var http = require('http').createServer(app);
var socket = require('socket.io').listen(http);

// Creating namespace for the Norris instance
var nsp = socket.of('/norris');

// Norris instantiation
var Norris = require('norris-rtbi')(nsp);
```

4.2 Mount

For a more in-depth explanation of the integration between Norris *framework*$_G$ and the developer application, the next example covers the correct use of a *mount*$_G$ point. The effects will be explained in the description that follows the example, while every proper *SDK*$_G$ feature will be described in the section 5.

```
// Creating Norris objects
var page = new Norris.Page("Page title", { "pageWidth": 1024, "columns": 2});
var barChart = new Norris.BarChart("Bar Chart title", "X Axis name", "Y Axis
    name", [1], [[5]]);
page.addGraph(barChart);

// Mounting the Norris object "page" on the express application
app.use('/norris', Norris.PageRouter(page));
```

The last instruction provides a *mount*$_G$ point for the *framework*$_G$, allowing it to create an internal *routing*$_G$ for its resources, making them available to be obtained via web requests. Specifically, *HTTP*$_G$ requests to the application URL with the addition of the *mount*$_G$ point name ("/norris" in the example) will result in the display of an *HTML*$_G$ page template containing the graphic representation of the created charts.
The internal *routing*$_G$ is also extended with the possibility of adding the "/raw" keyword after every resource; any request to this URL returns the raw JSON data of the respective resource. Being able to access the raw data directly allows them to be used

outside of the Norris *framework*$_G$ template, for example creating a customized template or use different graphic libraries for the chart visualization.

4.2.1 PageRouter(page)

Calling the method PageRouter() is required in order to make a certain page available with *Express*$_G$. The specific page will be the only parameter for this method.
The PageRouter method will create a *middleware*$_G$ object that has to be included in the application through the *Express*$_G$ method `app.use("URL", middlewareObject)`, as described in the previous example.
The page will not be available to be displayed if this method is not called with the specific page parameter.

4.3 Project example

The following code shows a simple and complete example of a Norris project.
After creating the page and adding two charts to it, the application will be started and the update logic will keep the charts updated.

```
// Express instantiation
var app = require('express')();
// Socket.io instantiation on the express instance
var http = require('http').createServer(app);
var socket = require('socket.io').listen(http);
// Creating namespace for the Norris instance
var nsp = socket.of('/Socket');
// Norris instantiation
var Norris = require('norris-rtbi')(nsp);

// Creating Norris objects
var Page = Norris.Page;
var LineChart = Norris.LineChart;
var BarChart = Norris.BarChart;
var PageRouter = Norris.PageRouter;
// Create new page
var page = new Page("First page", {columns:2} );

//creating Line Chart with some options
var axisLabel = [1,2,3];
var lineData = [[4,6,5],[7,6,2],[2,4,5],[3,2,1]];
var lineOpt = { series:["Jan", "Feb", "Mar", "Jun"]
              , legendPosition: "bottom"
              , labelsLimit: 10
              };
var myLine = new LineChart("Line Chart title", "X Axis", "Y Axis", axisLabel,
    lineData, lineOpt);

//creating Bar Chart with some options
var barData = [[7,8,9],[-1,-2,-3]];
var barOpt = { orientation: "horizontal", valueType: "dollars"};
var myBar = new BarChart("Bar Chart title", "X Axis", "Y Axis", [1,2,3],
    barData, barOpt);
```

```
// add graphs into page
page.addGraph(myLine);
page.addGraph(myBar);

// Mounting the Norris object "page" on the express application
app.use('/myPage', PageRouter(page));

// Update function
var label = 4;
setInterval(function() {
        var randVal = Math.floor(Math.random()*10);
        // insert new random value on new label
        myLine.updateStream(label, [randVal, randVal*2, randVal*0.5,
            randVal*1.5]);
        // updating serie "3" with random value on various label
        myLine.updateInPlace(label-1, 3, Math.floor(Math.random()*10));
        // update label 2 on serie 1 with random value
        myBar.updateInPlace(2,0, Math.floor(Math.random()*10)+1);
        label++;
}, 5000);

http.listen(process.env.PORT || 3000);
```

5 Software development kit

5.1 General description

Norris *framework*$_G$ exposes to the developer two main entities: pages and charts.
A page is a structure that works as a collection of different types of charts.
There are four types of chart: *Bar Chart*$_G$, *Line Chart*$_G$, *Map Chart*$_G$ and *Table*$_G$.
Every chart has to be included in a page in order to be displayed.

5.2 Page

A Page is a structure that identifies a web page and will contain the page properties and all the charts that were included in it. This page will be available on-line after it will be added to a *mount*$_G$ point of an *Express*$_G$ *middleware*$_G$ (see sec. 4.2).

5.2.1 Page(title, options)

This method creates a page with a set title, width and a maximum number of charts displayed in a row.
The title parameter is mandatory, while the options parameter is optional.

- **title**: string that represents the web page title;
- **options**: optional parameter that contains the options for the web page, this object must have the same structure of the example below.

```
{ "pageWidth": value
, "columns": value
}
```

  - **pageWidth**: integer that represents the pixel width of the web page, must be greater than 800;
  - **columns**: integer that represents the maximum number of charts that will be displayed in the same row of the web page, the value must range between 1 and 12, both included. If the page contains more charts than the value of this parameter, the charts will be displayed in more lines.

5.2.2 getPageInfo()

This method, called on a Page object, returns an object containing the page properties and the contained charts' information.

5.2.3 addGraph(graph)

This method, called on a Page object, includes a chart into a page. A specific chart may be included in one or more pages.

5.3 Line Chart

A *Line Chart*$_G$ is a chart type which shows the data set as a series of points linked with a line.
It is possible to have many data sets and set various parameters, as described next.

5.3.1 LineChart(title, xAxisName, yAxisName, labels, data, options)

This method creates a *Line Chart* type chart. Every parameter is mandatory, except for the options parameter.

- **title**: string that represents the chart title;
- **xAxisName**: string that represents the horizontal axis name;
- **yAxisName**: string that represents the vertical axis name;
- **labels**: array of strings that represents the labels of the values in the horizontal axis;
- **data**: array that represents the initial data sets, every element of this array has to be in the form of another array as it represents a single data set and must have the same length as the labels array;

$$[[a_0, a_1, a_2, ..., a_{label.length-1}], [b_0, b_1, b_2, ..., b_{label.length-1}], [c_0, c_1, c_2, ..., c_{label.length-1}]]$$

- **options**: optional parameter that contains the options for the chart, this object must have the same structure of the example below but doesn't have to be complete;

```
{ "series": []
, "grid": "value"
, "legend": "value"
, "legendPosition": "value"
, "colors": []
, "valueType": []
, "decimals": value
, "labelsLimit": value
}
```

  - **series**: array of strings that represents the name of each data set, these values will be displayed in the chart legend. If not left empty, the name for each data set must be specified. The default value for this option is automatically generated by Norris;
  - **grid**: boolean parameter that represents the choice of showing (true) or hiding (false) the chart grid. The default value for this parameter is true;
  - **legend**: boolean parameter that represents the choice of showing (true) or hiding (false) the chart legend. The default value for this parameter is true;
  - **legendPosition**: string that represents the position of the chart legend, the allowed values are "right", "left", "top" and "bottom". The default value for this parameter is "right";
  - **colors**: array of strings that represents the color each data set in the chart. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF"). If not left empty, the color for each data set must be specified. The default value for this option is automatically generated by Norris;

- **valueType**: string that represents the value type of the data in the chart, the allowed values are "euro", "dollars" and "pounds". The default value for this parameter doesn't apply any type;
- **decimals**: integer that represents the amount of decimal places for the data in the chart, the value must range between 0 and 6, both included. The default value for this parameter is 2;
- **labelsLimit**: integer that represents the maximum amount of labels and respective data that will be maintained for the chart. If more data is added past this point, the oldest data will be removed. The default value for this parameter is 300;

If this method is called with any invalid parameter, an error is thrown and will terminate the application if not caught.

5.3.2 getChartInfo()

This method, called on a *Line Chart*$_G$, returns an object containing the chart properties and its data.

5.3.3 updateInPlace(label, set, newValue)

This method, called on a *Line Chart*$_G$, updates a single value of the chart data. Every parameter is mandatory.

- **label**: string that represents the label of the value that is going to be updated;
- **set**: integer that represents the index of the data set that is going to be updated, value must range between 0 and the amount of data sets, with the latter not included;
- **newValue**: number that represents the new value for the specific data set in the specific label;

5.3.4 updateStream(newLabel, newValue)

This method, called on a *Line Chart*$_G$, updates the chart adding a new label and the respective new values for each data set. Every parameter is mandatory.

- **newLabel**: string that represents the label of the new values of each data set;
- **newValue**: array that contains the new values of each data set. It must contain one value for each data set;

5.4 Bar Chart

A *Bar Chart*$_G$ is a chart type which shows the data set as a series of horizontal or vertical bars.
It is possible to have many data sets and set various parameters, as described next.

5.4.1 BarChart(title, xAxisName, yAxisName, labels, data, options)

This method creates a *Bar Chart*$_G$ type chart. Every parameter is mandatory, except for the options parameter.

- **title**: string that represents the chart title;
- **xAxisName**: string that represents the horizontal axis name;
- **yAxisName**: string that represents the vertical axis name;
- **labels**: array of strings that represents the labels of the values in the horizontal axis;
- **data**: array that represents the initial data sets, every element of this array has to be in the form of another array as it represents a single data set and must have the same length of the labels array;

$$[[a_0, a_1, a_2, ..., a_{label.length-1}], [b_0, b_1, b_2, ..., b_{label.length-1}], [c_0, c_1, c_2, ..., c_{label.length-1}]]$$

- **options**: optional parameter that contains the options for the chart, this object must have the same structure of the example below but doesn't have to be complete;

```
{ "series": []
, "orientation": "value"
, "grid": "value"
, "legend": "value"
, "legendPosition": "value"
, "colors": []
, "valueType": []
, "decimals": value
}
```

  - **series**: array of strings that represents the name of each data set, these values will be displayed in the chart legend. If not left empty, the name for each data set must be specified. The default value for this option is automatically generated by Norris;
  - **orientation**: string that represents the orientation of the bars in the chart, the allowed values are "horizontal" and "vertical". The default value for this parameter is "vertical";
  - **grid**: boolean parameter that represents the choice of showing (true) or hiding (false) the chart grid. The default value for this parameter is true;
  - **legend**: boolean parameter that represents the choice of showing (true) or hiding (false) the chart legend. The default value for this parameter is true;
  - **legendPosition**: string that represents the position of the chart legend, the allowed values are "right", "left", "top" and "bottom". The default value for this parameter is "right". If the orientation parameter is set to "horizontal", legendPosition can't be set to "left";

- **colors**: array of strings that represents the color each data set in the chart. Only *HTML*$_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF"). If not left empty, the color for each data set must be specified. The default value for this option is automatically generated by Norris;
- **valueType**: string that represents the value type of the data in the chart, the allowed values are "euro", "dollars" and "pounds". The default value for this parameter doesn't apply any type;
- **decimals**: integer that represents the amount of decimal places for the data in the chart, the value must range between 0 and 6, both included. The default value for this parameter is 2;

If this method is called with any invalid parameter, an error is thrown and will terminate the application if not caught.

5.4.2 getChartInfo()

This method, called on a *Bar Chart*$_G$, returns an object containing the chart properties and its data.

5.4.3 updateInPlace(label, set, newValue)

This method, called on a *Bar Chart*$_G$, updates a single value of the chart data. Every parameter is mandatory.

- **label**: string that represents the label of the value that is going to be updated;
- **set**: integer that represents the index of the data set that is going to be updated, value must range between 0 and the amount of data sets, with the latter not included;
- **newValue**: number that represents the new value for the specific data set in the specific label;

5.5 Map Chart

A *Map Chart*$_G$ is a chart type which shows the information inside a map provided by Google Maps.
It is possible to highlight paths on the map, add *markers*$_G$ to it and set various parameters as described next.

5.5.1 MapChart(title, paths, points, centerLatitude, centerLongitude, options)

This method creates a *Map Chart* type chart. Every parameter is mandatory, except for the options parameter.

- **title**: string that represents the chart title;
- **paths**: array that represents the paths on the map, every element of this array has to be in the form of another array as it represents a single path composed by a series of points of interest. Each of those point is identified with an array of two elements, representing respectively the latitude and longitude coordinates.

The path will then be calculated by Google Maps API_G following the points of interest order.
In the following example, the "a" series represents the first path while the series "b" represents the second one.

$$[[[x_{a,0}, y_{a,0}], [x_{a,1}, y_{a,1}]], [[x_{b,0}, y_{b,0}], [x_{b,1}, y_{b,1}], [x_{b,2}, y_{b,2}]]]$$

It is possible to create a *Map Chart*$_G$ with no path providing an empty array to the constructor function, but it will not be possible to add any path afterwards;

- **points**: array that represent the set of markers to add to the map. Every point must have the following structure:

$$[\{latitude : x_0, longitude : y_0, id : "point-1"\}, \{latitude : x_1, longitude : y_1, id : "point-2"\}]$$

The "id" parameter must be unique compared to the other points id.
It is possible to create a *Map Chart*$_G$ with no markers providing an empty array to the constructor function and it will be possible to add markers afterwards with the movie type update;

- **centerLatitude**: number that represents the latitude of the map center, the value must range between -90 and 90;
- **centerLongitude**: number that represents the longitude of the map center, the value must range between -180 and 180:
- **options**: optional parameter that contains the options for the chart, this object must have the same structure as the example below but doesn't have to be complete;

```
{ "zoom": 0
, "legend": "value"
, "mapLegendPosition": "value"
, "colors": [ ]
, "pathName": [ ]
, "pathMode": [ ]
}
```

  - **zoom**: integer that represents the initial zoom level of the map, the value must range between 0 and 14, both included. The default value for this parameter is 4;
  - **legend**: boolean parameter that represents the choice of showing (true) or hiding (false) the chart legend. The default value for this parameter is true;
  - **mapLegendPosition**: string that represents the position of the map legend, the allowed values are "top-right", "top-left", "bottom-right" and "bottom-left". The default value for this parameter is "top-right";
  - **colors**: array of strings that represents the color of each path in the map. Only *HTML*$_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF"). If not left empty, the color for each data set must be specified. The default value for this option is automatically generated by Norris;

- **pathName**: array of strings that represents the name of each path, these values will be displayed in the map legend. If not left empty, the name for each path must be specified. The default value for this option is automatically generated by Norris;
- **pathMode**: string that represents the travel mode for the paths, the allowed values are "walking", "driving", "bicycling" and "transit". Specifically:
  - **walking**: travel walking;
  - **driving**: travel by car;
  - **bicycling**: travel by bicycle;
  - **transit**: travel by public transport.

  This allows to calculate the paths specifically for different needs. The default value for this parameter is "driving".

If this method is called with any invalid parameter, an error is thrown and will terminate the application if not caught.

5.5.2 getChartInfo()

This method, called on a *Map Chart*$_G$, returns an object containing the chart properties and its data.

5.5.3 updateInPlace(point, latitude, longitude)

This method, called on a *Map Chart*$_G$, updates the position of a single existing *marker*$_G$ on the map. Every parameter is mandatory.

- **point**: string that represents the id of an existing marker that is going to be updated;
- **latitude**: number that represents the new latitude of the *marker*$_G$, the value must range between -90 and 90;
- **longitude**: number that represents the new longitude of the *marker*$_G$, the value must range between -180 and 180.

5.5.4 updateMovie(newPositions)

This method, called on a *Map Chart*$_G$, updates the position of every *marker*$_G$ on the map at the same time. The newPositions parameter is mandatory.

- **newPosition**: array that represents the markers that the map will contain after the update, every element of this array represents a single *marker*$_G$ and must have the following structure:

$$[\{\}, \{\}, \{\}, ...]$$

  - **id**: unique string that identifies the *marker*$_G$ on the map;
  - **latitude**: number that represents the new latitude of the *marker*$_G$, the value must range between -90 and 90;

– **longitude**: number that represents the new longitude of the $marker_G$, the value must range between -180 and 180.

```
{ id: "name"
, latitude: value
, longitude: value
}
```

If one of the $markers_G$ already exists on the map, its position will be changed accordingly to respect the new coordinates; if it doesn't, a new $marker_G$ will be added to the map. Any previously existing $marker_G$ that does not appear in this array will be removed from the map.

5.6 Table

A $Table_G$ is a chart type which shows the information inside a table.
It is possible to set various parameters as described next.

5.6.1 Table(title, headers, data, options)

This method creates a *Table* type chart. Every parameter is mandatory, except for the options parameter.

- **title**: string that represents the chart title;
- **headers**: array of strings that represents the title of each column header of the table. The amount of headers will also determine the amount of columns in the table;
- **data**: array that represents the initial data sets, every element of this array has to be in the form of another array as it represents a single row and must have the same length of the headers array;

$$[[x_{0,0}, y_{0,1}], [x_{1,0}, y_{1,1}], [x_{2,0}, y_{2,1}]]$$

It is possible to create a $Table_G$ with no data providing an empty array to the constructor function and it will be possible to add rows afterwards with the stream type update;

- **options**: optional parameter that contains the options for the chart, this object must have the same structure as the example below but doesn't have to be complete;

```
{ "insertPosition": "value"
, "orderBy":{ "column": 0
            , "order": "value"
            }
, "displayedLines": int
, "border": "value"
, "colorColumn" : []
, "colorRow" : []
, "colorCell" : []
```

```
, "colorColumnFont" : []
, "colorRowFont" : []
, "colorFont" : []
, "format": []
, "rowsLimit": 300
};
```

- **insertPosition**: string that represents the position where the new row will be added to the table after a stream type update, the allowed values are "top" and "bottom". The default value for this parameter is "bottom";
- **orderBy**: object that represents the order of the table elements. This object must have the same structure of the example below and, if not left empty, every parameter must be defined with a valid value;

```
{ "column": value
, "order": "value"
}
```

  - **column**: integer that represents the column on which the order will be applied. The value must range between 0 and the amount of columns, with zero included;
  - **order**: string that represents the type of ordering to apply on the column, the allowed values are "ascending" and "descending".
- **displayedLines**: integer that represents the maximum amount of rows that will be displayed at the same time, the value must be greater than 0;
- **border**: boolean parameter that represents the choice of showing (true) or hiding (false) the table borders. The default value for this parameter is true;

  colorColumn: array that represents the background color of one or more column in the table. Each element of the array represents a single column in the form of a two cells array that contains the column index and the color string. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

  ```
  [ [columnIndex, "" ], ... ]
  ```

  If left completely or partially empty, the default value for this option is automatically set by Norris;
- **colorRow**: array that represents the background color of one or more rows in the table. Each element of the array represents a single row in the form of a two cells array that contains the row index and the color string. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

  ```
  [ [rowIndex, "" ], ... ]
  ```

  If left completely or partially empty, the default value for this option is automatically set by Norris;
- **colorCell**: array that represents the background color of one or more cells in the table. Each element of the array represents a single cell in the form of a three cells array that contains the row index, the column index and the color

string. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

```
[ [rowIndex, columnIndex, "" ], ... ]
```

If left completely or partially empty, the default value for this option is automatically set by Norris;

- **colorColumnFont**: array that represents the font color of one or more column in the table. Each element of the array represents a single column in the form of a two cells array that contains the column index and the color string. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

```
[ [columnIndex, "" ], ... ]
```

If left completely or partially empty, the default value for this option is automatically set by Norris;

- **colorRowFont**: array that represents the font color of one or more row in the table. Each element of the array represents a single row in the form of a two cells array that contains the row index and the color string. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

```
[ [rowIndex, "" ], ... ]
```

If left completely or partially empty, the default value for this option is automatically set by Norris;

- **colorFont**: array that represents the font color of one or more cells in the table. Each element of the array represents a single cell in the form of a three cells array that contains the row index, the column index and the color string. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

```
[ [rowIndex, columnIndex, "" ], ... ]
```

If left completely or partially empty, the default value for this option is automatically set by Norris;

- **format**: array that represents the data format of the table data. Each element of the array represents the data format of a single column and must have the same structure of the example below (only the column parameter is mandatory);

```
{ "column": value
, "valueType": "value"
, "decimals": value
}
```

  * **column**: integer that represents the column on which the data format will be applied. The value must range between 0 and the amount of columns, with zero included;
  * **valueType**: string that represents the value type of the data in the column, the allowed values are "euro", "dollars" and "pounds". The default value for this parameter doesn't apply any type;

    - **decimals**: integer that represents the amount of decimal places for the data in the column, the value must range between 0 and 6, both included. The default value for this parameter is 2.
  - **rowsLimit**: integer that represents the maximum amount of rows that will be maintained for the table. If more data is added past this point, the oldest data will be removed. The default value for this parameter is 300.

  If this method is called with any invalid parameter, an error is thrown and will terminate the application if not caught.

5.6.2 getChartInfo()

This method, called on a *Table*$_G$ chart, returns an object containing the chart properties and its data.

5.6.3 updateInPlace(row, column, newValue, options)

This method, called on a *Table*$_G$, updates a single value of the table data. Every parameter is mandatory, except for the options parameter.

- **row**: integer that represents the row on which the update will be applied. The value must range between 0 and the amount of rows, with zero included;
- **column**: integer that represents the column on which the update will be applied. The value must range between 0 and the amount of columns, with zero included;
- **newValue**: value that represents the new value of the cell that will be updated;
- **options**: optional parameter that contains the options for the chart, this object must have the same structure as the example below but doesn't have to be complete;

```
{ "colorFont": "value"
, "colorCell": "value"
}
```

  - **colorFont**: string that represents the new font color of the cell. Only *HTML*$_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF");
  - **colorCell**: string that represents the new background color of the cell. Only *HTML*$_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

5.6.4 updateStream(data, options)

This method, called on a *Table*$_G$, updates the chart adding a new row to the table. The data parameter is mandatory, while the option parameter is optional.

- **data**: array that represents the new row data, it may contain a different amount of element compared to the headers of the table. If the row contains more cells, the extra cells will be ignored. If the row contains less cells, the missing cells will remain empty;

- **options**: optional parameter that contains the options for the chart, this object must have the same structure as the example below but doesn't have to be complete;

```
{ "colorRow": "value"
, "colorRowFont": "value"
, "colorCell": []
, "colorFont": []
};
```

  - **colorRow**: string that represents the new background color of the row cells. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF");
  - **colorRowFont**: string that represents the new font color of the row cells. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF");
  - **colorCell**: array that represents the background color of one or more cells of the new row. Each element of the array represents a single column in the form of a two cells array that contains the column index and the color string. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

    ```
    [ [columnIndex, "" ], ... ]
    ```

    If left completely or partially empty, the default value for this option is automatically set by Norris;
  - **colorFont**: array that represents the font color of one or more cells in the table. Each element of the array represents a single column in the form of a two cells array that contains the column index and the color string. Only $HTML_G$ hexadecimal values are accepted (e.g. "#FFFFFF" or "FFFFFF").

    ```
    [ [columnIndex, "" ], ... ]
    ```

    If left completely or partially empty, the default value for this option is automatically set by Norris.

A Glossary

A

API (Application Programming Interface)
An API is a set of routines, protocols, and tools for building software applications.

B

Bar Chart
A bar chart is a chart that presents grouped data with rectangular bars with lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

C

Client
A client is a piece of computer hardware or software that accesses a service made available by a *server*$_G$. The *server*$_G$ is often (but not always) on another computer system, in which case the client accesses the service by way of a network.

E

Express
Express is a *Node.js*$_G$ web application *framework*$_G$, designed for building single-page, multi-page, and hybrid web applications

F

Framework
A framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software.

H

HTML (HyperText Markup Language)
HTML is the standard markup language used to create web pages.

HTTP (HyperText Transfer Protocol)
The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

J

JSON (JavaScript Object Notation)
JSON is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a *server*$_G$ and web application, as an alternative to *XML*$_G$.

L

Line Chart
is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments. It is a basic type of chart common in many fields. A line chart is often used to visualize a trend in data over intervals of time – a time series – thus the line is often drawn chronologically.

M

Map Chart
A map chart is a map in which some thematic mapping variable is substituted for land area or distance. The geometry or space of the map is distorted in order to convey the information of this alternate variable.

Marker
A marker is a symbol used in a *Map Chart*$_G$ to show a specific point on the map.

Middleware
Middleware is a computer software that provides services to software applications beyond those available from the operating system. It makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application.

Mount
In Unix-like operative systems, the mount command instructs the operating system that a file system is ready to use, and associates it with a particular point in the overall file system hierarchy (its mount point) and sets options relating to its access. Mounting makes file systems, files, directories, devices and special files available for use and available to the user.
In Norris, mount is intended as making a Page resource available to the final user through a specifi URL (its mount point).

N

Namespace
A namespace is a set of named symbols, usually variables. Names or identifiers are keys allowing access to symbol values. Namespaces provide a level of direction to specific identifiers, thus making it possible to distinguish between identical identifiers.
In computer programming, namespaces are typically employed for the purpose of grouping symbols and identifiers around a particular behavior. In Norris, the namespace is used to avoid conflicts between the developer's main instance of socket.io and the one used by Norris.

Node.js
Node.js is an open source, cross-platform runtime environment for *server*$_G$-side and networking applications. Node.js applications are written in JavaScript. Node.js provides an event-driven architecture and a non-blocking I/O *API*$_G$ that optimizes an application's throughput and scalability. These technologies are commonly used for real-time web applications.

npm
It is the default package manager for *Node.js*$_G$. It manages dependencies for an application and allows users to install Node.js applications that are available on the npm registry.

P

Push
Push, or *server*$_G$ push, describes a style of Internet-based communication where the request for a given transaction is initiated by the publisher or central *server*$_G$. It is contrasted with pull/get, where the request for the transmission of information is initiated by the receiver or *client*$_G$.

R

Routing
Routing refers to the definition of end points to an application and how it responds to client requests. A route is a combination of a URI, a HTTP request method (GET in Norris) and one or more handlers for the endpoint.

S

SDK
An SDK is a set of software development tools that allows the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar development platform.

Server
A server is a running instance of an application capable of accepting requests from the $client_G$ and giving responses accordingly. Servers can run on any computer including dedicated computers.

Socket.io
Socket.io is a JavaScript library for realtime web applications. It enables realtime, bi-directional communication between web $clients_G$ and $server_G$. It has two parts: a $client_G$-side library that runs in the browser, and a $server_G$-side library for $node.js_G$. Socket.io primarily uses the WebSocket protocol.

T

Table
A table is a means of arranging data in rows and columns.

X

XML (Extensible Markup Language)
XML is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable.

B Contacts

You can report any problem or bug occurrence found during the installation or the use of Norris *framework*$_G$ opening a GitHub issue at the following URL:

https://github.com/FlameTech/Norris-rtbi/issues

Logging in to the site is required and after displaying the following page, simply press the button "New issue" to create a report.

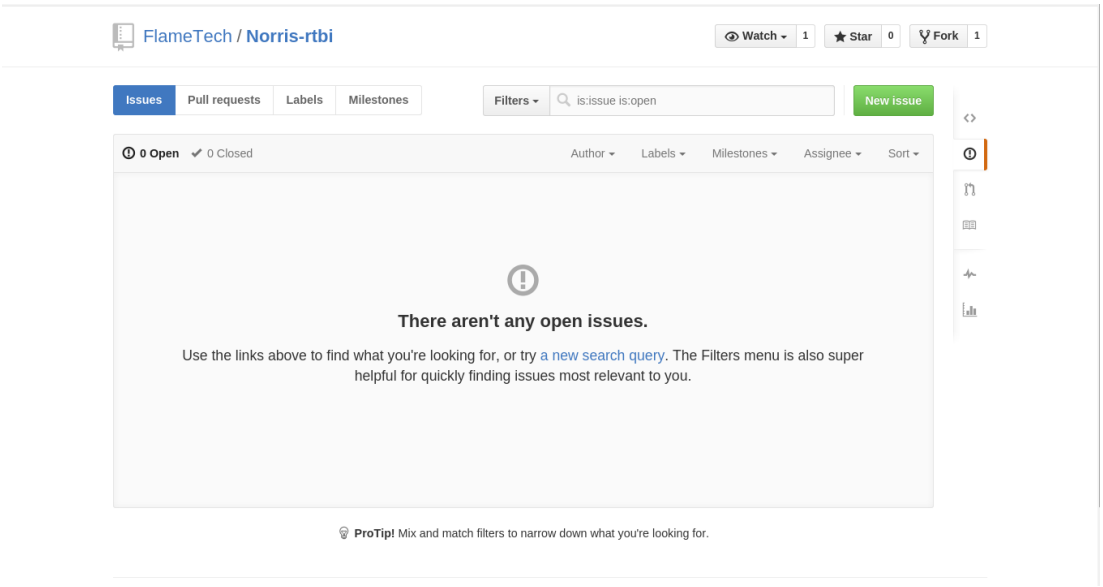

Figure 1: Issue section

Once the report is compiled, press the button “Submit new issue” to submit it to us.

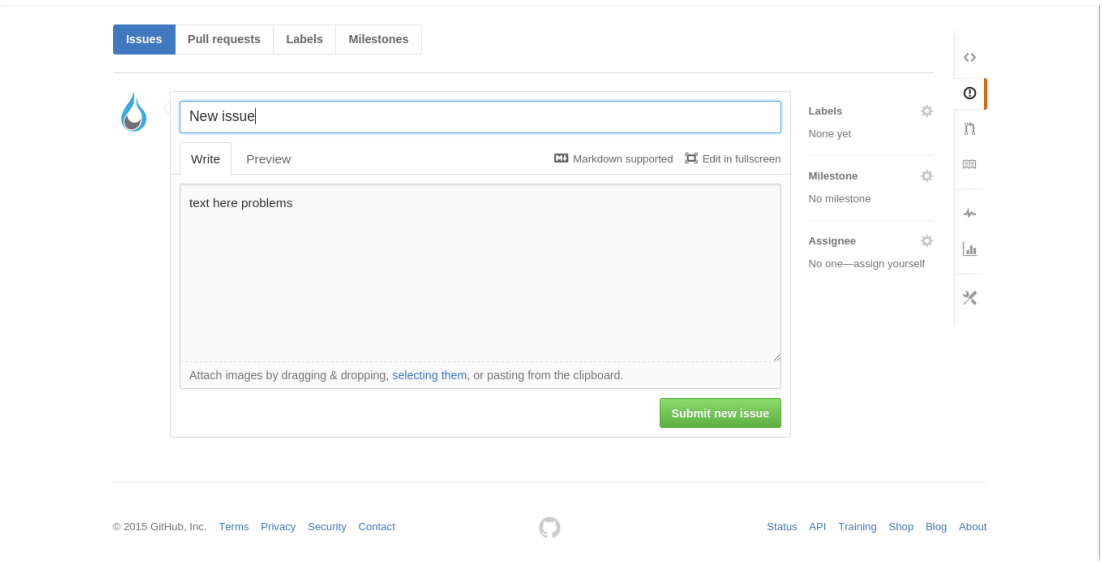

Figure 2: Issue creation