Norris

Node Real-time Intelligence

Norris - Developer Manual

Document Informations

Document Name	Norris - Developer Manual
Version	4.0
State	*Formale*
Usage	*Esterno*
Creation Date	2015/06/12
Last Change	2015/06/16
Writing	Samuele Zanella
	Enrico Savoca
Approval	Maria Giovanna Chinellato
Verification	Davide Trivellato
	Francesco Rossetto
Distribution List	Developers

Summary

This document contains the user manual to the **Norris** framework.

Changelog

Version	Date	Author	Role	Description
v3.00	2015/06/16	Maria Giovanna Chinellato	Project Manager	Document Approval
v2.11	2015/06/15	Francesco Rossetto	Verifier	Document verification
v2.10	2015/06/14	Davide Trivellato	Verifier	Document verification
v2.9	2015/06/14	Samuele Zanella	Programmer	Created the Web integration section
v2.8	2015/06/13	Samuele Zanella	Programmer	Changed structure slightly
v2.7	2015/06/12	Enrico Savoca	Programmer	Described the flow classes
v2.6	2015/06/11	Samuele Zanella	Programmer	Described the Norris, Page, and the charts classes
v2.5	2015/06/09	Enrico Savoca	Programmer	Created The Norris Framework section
v2.4	2015/06/09	Samuele Zanella	Programmer	Edited Getting Started section
v2.3	2015/06/08	Samuele Zanella	Programmer	Created Getting Started section
v2.2	2015/06/08	Enrico Savoca	Programmer	Created introduction section
v2.1	2015/06/07	Samuele Zanella	Programmer	Document creation

Tabella 1: Document Versioning.

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduction

1.1 Welcome to Norris

Norris is a framework$_{|g|}$ that allows you to create four types of graphs, namely Bar Charts$_{|g|}$, Line Charts$_{|g|}$, Map Charts$_{|g|}$ and Tables$_{|g|}$, and pages, which serve the purpose of containing several of the aforementioned graphs. Each graph can contain any number of data flows, and allows for several options including filters for data, legends, various appearance options and so on.

1.2 Document purpose

The document goal is to explain developers how to install and run an instance of the framework$_{|g|}$; by using this manual the developer will immediately have a detailed yet broad overview of the entire framework, which will allow its usage to get within his grasp.

1.3 Glossary

In pursuance of avoiding words' misunderstanding and allowing a clear comprehension of the manual, it's possible to find the explanation to some ambiguous or specific words at the end of the document, in a Glossary. Words that are reported in the glossary are marked with the following symbol: $_{|g|}$.

1.4 Minimal requirements

In order to use the Norris framework$_{|g|}$, the following libraries are required:

- Express.js$_{|g|}$
- Socket.io$_{|g|}$

Once the required libraries are included, the user can proceed installing the Norris module by using the following instruction:

```
1    npm install norris-nrti
```

2 Getting started

2.1 Framework installation

An updated version of the Norris framework$_{|g|}$ can be found on the npm platform, and it can hence be obtained by using the instruction:

```
npm install norris-nrti
```

2.2 What is Norris

As previously introduced, Norris is a framework$_{|g|}$ with the aim of creating and updating graphs, while continuously notifying every client connected to them. Said graphs are organised in different pages, which they are bound to and where they are displayed in a given order and layout.
The developer has full control over every single property of his pages and graphs, which he can update at any moment; the framework$_{|g|}$ also offers a handful of methods to update, add or remove records of the graph data, which vary depending on the graph type. Pages and graphs, once created, are fully available for clients to connect to; once connected, they will be constantly notified of every update to the object's properties and data.

2.3 First steps

The very first operation to do, when creating a Norris instance, is including the required libraries and creating the Express$_{|g|}$ app and the Socket$_{|g|}$ object

```
var express = require('express');
var app = express();
var http = require('http');
var server = http.createServer(app);
var io = require('socket.io')(server);
```

The next step is creating a Norris object:

```
var Norris = require('norris-nrti.js');
var norris = new Norris(app,io,'/norris','http://norris-
   example.com');
```

The details of the constructor usage will be covered later.
From now on, the developer can create several pages and graphs, and setup for each of them the desired updating procedures.
After all that regards Norris is set up, the developer can configure his own routings on the Express app: here's an example:

```
app.use(function(req, res, next) {
  res.status(404).send('404 Not Found');
});
```

Bear in mind, though, that several namespaces on the Express app will be already taken by the Norris instance; in particular, the namespaces used for the page list, the pages and the graphs won't be available for further usage.

Finally, the developer has to start the server, for the clients to be able to connect to it; the instruction might be similar to this:

```
1     var port = process.env.PORT || 3000;
2     server.listen(port);
```

3 The Norris framework

This section will explain the usage of each of the Norris framework classes. Both the general purpose and the detailed method overview will be explained in detail.

3.1 The Norris framework classes

The Norris framework consists in the following classes:

- Norris
- Page
- BarChart
- LineChart
- MapChart
- Table
- BarChartFlow
- LineChartFlow
- MapChartFlow
- TableFlow

3.2 Norris

The Norrisclass represents the actual instance of the Norris framework$_{|g|}$. It contains the list of pages and all the objects that are needed for network communication.

3.2.1 Method overview

3.2.1.1 constructor This is the constructor of the Norris class.
Parameters:

- *Express* app: the Express app that Norris uses;
- *SocketIo* io: the Socket.io object that Norris uses;
- *String* namespace: the namespace that will be used for the page list in this Norris instance;
- *String* baseURL: the base URL where the Norris instance will be located.

Example usage:

```
var norris = new Norris(app,io,'/norris','http://
   norris-example.com');
```

3.2.1.2 createPage This method allows the user to create a Page inside the Norris instance; the properties of the page are passed as a parameter to the method.
Return value: *Page*: a reference to the created page.
Parameters:

- *Object* params: the page parameters.
  The params object has to respect the following structure:

```
{
  ID: 'page1',
  name: 'Page 1',
  description: 'This is page 1',
  graphsPerRow: 3,
  graphsPerCol: 5
}
```

  Only the ID field is mandatory; every other field can be omitted. Field description:

  - ID
    - Type: String;
    - Description: text that univocally identifies the page in the Norris instance;
    - Valid values: any non-empty string.
  - name
    - Type: String;
    - Description: name of the page;

    * Default value: empty string;
    * Valid values: any non-empty string.
  - description
    * Type: String;
    * Description: description of the page;
    * Default value: empty string;
    * Valid values: any non-empty string.
  - graphsPerRow
    * Type: integer;
    * Description: number of graphs for each row of the page;
    * Default value: 1;
    * Valid values: any positive integer or -1 (for an unlimited amount of graphs).
  - graphsPerCol
    * Type: integer;
    * Description: number of graphs for each column of the page;
    * Default value: -1;
    * Valid values: any positive integer or -1 (for an unlimited amount of graphs).

Example usage:

```
1       var page1=norris.createPage({
2           ID:'page1',
3           name: 'Page 1',
4           description: 'This is a fancy little page',
5           graphsPerRow: 2,
6           graphsPerCol: 10
7       });
```

3.3 Page

The Page class represents a single page in the Norris framework$_{|g|}$. It's defined by several properties, and it contains the list of its graphs.

3.3.1 Method overview

3.3.1.1 createBarChart This method allows the user to create a Bar Chart inside the page; the properties of the graph are passed as a parameter to the method.
Return value: *BarChart*: a reference to the created Bar Chart.
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
  ID: 'bar1',
  title: 'bar1',
  width: 800,
  height: 600,
  enableLegend: true,
  barOrientation: 'V',
  backgroundColor: '#FFFFFF',
  legendOnPoint: true,
  grid: true,
  groupingControl: true,
  headers:[
    'col1',
    'col2',
    'col3'
  ],
  legend: {
    position: 'NE',
    fontColor: '#000000',
    backgroundColor: '#FFFFFF'
  },
  xAxis: {
    name: 'x',
    color: '#000000',
    minIndex: 5,
    maxIndex: 10,
    ticks: 10
  },
  yAxis: {
    name: 'y',
    color: '#000000',
    minIndex: 5,
    maxIndex: 10,
    ticks: 10
  }
```

```
36          }
```

Only the ID field is mandatory; every other field can be omitted. Field description:

- ID
  - Type: String;
  - Description: text that univocally identifies the graph in the page;
  - Valid values: any non-empty string.
- title
  - Type: String;
  - Description: title of the graph;
  - Default value: empty string;
  - Valid values: any non-empty string.
- width
  - Type: integer;
  - Description: width in pixel of the graph in the page;
  - Default value: 500;
  - Valid values: any positive integer.
- height
  - Type: integer;
  - Description: height in pixel of the graph in the page;
  - Default value: 400;
  - Valid values: any positive integer.
- enableLegend
  - Type: boolean;
  - Description: defines whether the legend is displayed or not in the graph;
  - Default value: false;
  - Valid values: true or false.
- barOrientation
  - Type: String;
  - Description: defines the bar orientation of the graphs;
  - Default value: 'V';
  - Valid values: 'V' (vertical) or 'H' (horizontal).
- backgroundColor
  - Type: String;
  - Description: defines the background color of the graph;
  - Default value: '#FFFFFF';
  - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.

- legendOnPoint
  - Type: boolean;
  - Description: defines whether the legend on point functionality is enabled or not;
  - Default value: false;
  - Valid values: true or false.
- grid
  - Type: boolean;
  - Description: defines whether the grid is displayed or not in the graph;
  - Default value: false;
  - Valid values: true or false.
- groupingControl
  - Type: boolean;
  - Description: defines whether the front end user is allowed to switch between grouped and stacked view;
  - Default value: false;
  - Valid values: true or false.
- headers
  - Type: Array of String;
  - Description: contains the list of the header labels;
  - Default value: [];
  - Valid values: any array of String objects.
- legend
  - Type: object;
  - Description: contains the legend configuration; refer to the legend section (3.12.1) for the structure description;
- xAxis
  - Type: object;
  - Description: contains the x axis configuration; refer to the axis section (3.12.2) for the structure description;
- yAxis
  - Type: object;
  - Description: contains the y axis configuration; refer to the axis section (3.12.2) for the structure description;

Example usage:

```
var barChart=page1.createBarChart({
    ID: 'bar1',
    title: 'BAR',
    height: 600,
    width: 1000,
    enableLegend: true,
```

```
        legend: {
            position: 'NE',
        },
        xAxis:{
            name: 'time',
            color: '#000000',
            minIndex: 0,
            maxIndex: 7,
            scale:'logarithmic'
        },
        yAxis:{
            name: 'pressure',
            color: '#000000'
        },
        headers: [
            'h1',
            'h2',
            'h3',
            'h4',
            'h5'
        ],
        backGroundColor: '#FFFFFF',
        viewFinder: true,
        grid: true,
        legendOnPoint: true,
        barOrientation: 'V',
        sortable: true,
        groupingControl: true
    });
```

3.3.1.2 createLineChart This method allows the user to create a Line Chart inside the page; the properties of the graph are passed as a parameter to the method.
Return value: *LineChart*: a reference to the created Line Chart.
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
  ID: 'line1',
  title: 'line1',
  width: 800,
  height: 600,
  enableLegend: true,
  backgroundColor: '#FFFFFF',
  legendOnPoint: true,
  horizontalGrid: true,
  verticalGrid: true,
  viewFinder: true,
  interpolation: 'monotone',
  legend: {
    position: 'NE',
    fontColor: '#000000',
    backgroundColor: '#FFFFFF'
  },
  xAxis: {
    name: 'x',
    color: '#000000',
    minIndex: 5,
    maxIndex: 10,
    ticks: 10
  },
  yAxis: {
    name: 'y',
    color: '#000000',
    minIndex: 5,
    maxIndex: 10,
    ticks: 10
  }
}
```

  Only the ID field is mandatory; every other field can be omitted. Field description:

  - ID
    - Type: String;
    - Description: text that univocally identifies the graph in the page;
    - Valid values: any non-empty string.
  - title

    - Type: String;
    - Description: title of the graph;
    - Default value: empty string;
    - Valid values: any non-empty string.
- width
    - Type: integer;
    - Description: width in pixel of the graph in the page;
    - Default value: 500;
    - Valid values: any positive integer.
- height
    - Type: integer;
    - Description: height in pixel of the graph in the page;
    - Default value: 400;
    - Valid values: any positive integer.
- enableLegend
    - Type: boolean;
    - Description: defines whether the legend is displayed or not in the graph;
    - Default value: false;
    - Valid values: true or false.
- backgroundColor
    - Type: String;
    - Description: defines the background color of the graph;
    - Default value: '#FFFFFF';
    - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.
- legendOnPoint
    - Type: boolean;
    - Description: defines whether the legend on point functionality is enabled or not;
    - Default value: false;
    - Valid values: true or false.
- horizontalGrid
    - Type: boolean;
    - Description: defines whether the horizontal grid is displayed or not in the graph;
    - Default value: false;
    - Valid values: true or false.
- verticalGrid
    - Type: boolean;

    - Description: defines whether the vertical grid is displayed or not in the graph;
    - Default value: false;
    - Valid values: true or false.
- viewFinder
    - Type: boolean;
    - Description: defines whether the view finder is enabled or not;
    - Default value: false;
    - Valid values: true or false.
- interpolation
    - Type: String;
    - Description: defines the type of interpolation to use on the graph;
    - Default value: 'linear';
    - Valid values: 'linear', 'step', 'basis', 'cardinal', 'monotone'.
- legend
    - Type: object;
    - Description: contains the legend configuration; refer to the legend section (3.12.1) for the structure description;
- xAxis
    - Type: object;
    - Description: contains the x axis configuration; refer to the axis section (3.12.2) for the structure description;
- yAxis
    - Type: object;
    - Description: contains the y axis configuration; refer to the axis section (3.12.2) for the structure description;

Example usage:

```
1       var lineChart=page1.createLineChart({
2           ID: 'line1',
3           title: 'LINE',
4           height: 600,
5           width: 1000,
6           enableLegend: true,
7           legend: {
8               position: 'NE',
9           },
10          xAxis:{
11              name: 'time',
12              color: '#000000',
13              scale: 'linear'
14          },
15          yAxis:{
```

```
            name: 'temperature',
            color: '#000000',
            scale: 'linear'
        },
        backGroundColor: '#FFFFFF',
        viewFinder: true,
        horizontalGrid: true,
        verticalGrid: true,
        legendOnPoint: true,
        interpolation: 'monotone'
    });
```

3.3.1.3 createMapChart This method allows the user to create a Map Chart inside the page; the properties of the graph are passed as a parameter to the method.
Return value: *LineChart*: a reference to the created Map Chart.
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
        {
          ID: 'map1',
          title: 'map1',
          width: 800,
          height: 600,
          enableLegend: true,
          legendOnPoint: true,
          latitude: 0,
          longitude: 0,
          mapType:'roadmap',
          mapWidth: 4000,
          mapHeight: 2500,
          legend: {
            position: 'NE',
            fontColor: '#000000',
            backgroundColor: '#FFFFFF'
          }
        }
```

  Only the ID field is mandatory; every other field can be omitted. Field description:

  - ID
    - Type: String;
    - Description: text that univocally identifies the graph in the page;
    - Valid values: any non-empty string.
  - title
    - Type: String;

    - Description: title of the graph;
    - Default value: empty string;
    - Valid values: any non-empty string.
- width
    - Type: integer;
    - Description: width in pixel of the graph in the page;
    - Default value: 500;
    - Valid values: any positive integer.
- height
    - Type: integer;
    - Description: height in pixel of the graph in the page;
    - Default value: 400;
    - Valid values: any positive integer.
- enableLegend
    - Type: boolean;
    - Description: defines whether the legend is displayed or not in the graph;
    - Default value: false;
    - Valid values: true or false.
- legendOnPoint
    - Type: boolean;
    - Description: defines whether the legend on point functionality is enabled or not;
    - Default value: false;
    - Valid values: true or false.
- latitude
    - Type: number;
    - Description: defines the map center point latitude;
    - Default value: 0;
    - Valid values: any valid latitude value.
- longitude
    - Type: number;
    - Description: defines the map center point longitude;
    - Default value: 0;
    - Valid values: any valid longitude value.
- mapType
    - Type: String;
    - Description: defines the type of map displayed to the clients;
    - Default value: 'roadmap;
    - Valid values: 'roadmap', 'satellite', 'terrain', 'hybrid'.
- mapWidth

    - Type: integer;
    - Description: defines the width in meters of the portion of map displayed;
    - Default value: 3000;
    - Valid values: any positive integer value.
  - mapHeight
    - Type: integer;
    - Description: defines the height in meters of the portion of map displayed;
    - Default value: 2000;
    - Valid values: any positive integer value.
  - legend
    - Type: object;
    - Description: contains the legend configuration; refer to the legend section (3.12.1) for the structure description;

Example usage:

```
var mapChart=page1.createMapChart({
    ID: 'map1',
    title: 'APS',
    height: 600,
    width: 1000,
    enableLegend: true,
    legend: {
        position: 'NW',
        fontColor: '#00AA00',
        backgroundColor: '#FFAAFF'
    },
    latitude: 45.4113311,
    longitude: 11.8876318,
    mapType: 'roadmap',
    mapWidth: 2000,
    mapHeight: 2000,
    legendOnPoint: true
});
```

3.3.1.4 createTable This method allows the user to create a Table inside the page; the properties of the graph are passed as a parameter to the method.
Return value: *LineChart*: a reference to the created Table.
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
  ID: 'table1',
  title: 'TABLE',
  width: 800,
  height: 600,
  sort: {
    column: 'col1',
    ordering: 'DESC',
  }
  maxItemsPage = 15,
  headers = [
    'col1',
    'col2',
    'col3',
    'col4'
  ],
  addRowOn = 'top',
  appearance: {
        horizontalGrid: {
            color: '#00AA00', //#xxxxxx,
            width: 1 // > 0
        },
        verticalGrid: {
            color: '#00AA00', //#xxxxxx,
            width: 1 // > 0
        },
        rowEven: {
            textColor: ['#00AB00', '#AA0000'],
            backgroundColor: ['#FAAFFF', '#
               FFFAFF']
        },
        rowOdd: {
            textColor: ['#BB0000', '#BB0000'],
            backgroundColor: ['#AAFFFF', '#
               FAAAFF']
        },
        headers: {
            textColor: ['#00CC00', '#00CC00'],
            backgroundColor: ['#FFAAFF', '#
               FABCDF']
        }
  }
```

```
40          }
```

Only the ID field is mandatory; every other field can be omitted. Field description:

- ID
  - Type: String;
  - Description: text that univocally identifies the graph in the page;
  - Valid values: any non-empty string.
- title
  - Type: String;
  - Description: title of the graph;
  - Default value: empty string;
  - Valid values: any non-empty string.
- width
  - Type: integer;
  - Description: width in pixel of the graph in the page;
  - Default value: 500;
  - Valid values: any positive integer.
- height
  - Type: integer;
  - Description: height in pixel of the graph in the page;
  - Default value: 400;
  - Valid values: any positive integer.
- sort
  - Type: object;
  - Description: defines the sorting rules; refer to the sort section (3.12.3) for the structure description;
- maxItemsPage
  - Type: integer;
  - Description: number of elements per each page of the Table;
  - Default value: 10;
  - Valid values: any positive integer.
- headers
  - Type: Array of String;
  - Description: contains the list of the header labels;
  - Default value: [];
  - Valid values: any array of String objects.
- addRowOn
  - Type: String;
  - Description: determines where the row are added to the Table;

    - Default value: 'bottom';
    - Valid values: 'top' or 'bottom'.
- appearance
    - Type: object;
    - Description: defines the appearance rules; refer to the appearance section (3.12.4) for the structure description.

Example usage:

```
var table=page1.createTable({
    ID: 'table1',
    title: 'Table',
    height: 600,
    width: 1000,
    sortable: true,
    maxItemsPage: 20,
    addRowOn: 'top',
    headers: ['col1', 'col2', 'col3'],
    sort: {
        column: 'col2',
        ordering: 'DESC'
    },
    appearance: {
        border: {
            color: '#00AA00',
            width: 1 // > 0
        },
        rowEven: {
            textColor: ['#00AB00', '#AA0000'],
            backgroundColor: ['#FAAFFF', '#FFFAFF']
        },
        rowOdd: {
            textColor: ['#BB0000', '#BB0000'],
            backgroundColor: ['#AAFFFF', '#FAAAFF']
        },
        headers: {
            textColor: ['#00CC00', '#00CC00'],
            backgroundColor: ['#FFAAFF', '#FABCDF']
        }
    },
});
```

3.3.1.5 updateProperties This method allows the user to update the page properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the page parameters.
  The params object has to respect the following structure:

```
{
  name: 'Page 1',
  description: 'This is page 1',
  graphsPerRow: 3,
  graphsPerCol: 5
}
```

  No field is mandatory. Field description:

  - name
    - Type: String;
    - Description: name of the page;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - description
    - Type: String;
    - Description: description of the page;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - graphsPerRow
    - Type: integer;
    - Description: number of graphs for each row of the page;
    - Default value: 1;
    - Valid values: any positive integer or -1 (for an unlimited amount of graphs).
  - graphsPerCol
    - Type: integer;
    - Description: number of graphs for each column of the page;
    - Default value: -1;
    - Valid values: any positive integer or -1 (for an unlimited amount of graphs).

Example usage:

```
page1.updateProperties({
    name: 'Page 1',
    description: 'This is a fancy little page',
    graphsPerRow: 2,
    graphsPerCol: 10
});
```

3.3.1.6 getProperties This method returns an object containing all of the page properties.
Return value: *Object*: the object that contains the page properties
Its structure is as follows:

```
{
  ID: 'page1',
  name: 'Page 1',
  description: 'This is page 1',
  graphsPerRow: 3,
  graphsPerCol: 5
}
```

Example usage:

```
page1.getProperties();
```

3.4 BarChart

The BarChart class represents a Bar Chart in the Norris framework$_{|g|}$. It's defined by several properties, and it contains the list of its flows.

3.4.1 Method overview

3.4.1.1 getFlowByID This method returns the flow that's identified by the given ID.
Return value: *BarChartFlow*: the desired flow
Parameters:

- *String* ID: the ID of the flow to return.

Example usage:

```
1       var flow1 = graph1.getFlowByID('flow1');
```

3.4.1.2 createBarChartFlow This method allows the user to create a flow inside a Bar Chart; the properties and records of the flow are passed as a parameter to the method.
The flow already contains all of its records when created, as later on it's not possible to add or remove any.
Each record in a flow represents a single bar, represented by an index, the position on the x axis, and a value, its height. Records from different flows having the same index will be rendered as bars grouped (or stacked) together. **Return value**: *BarChartFlow*: a reference to the created Bar Chart flow.
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
1         {
2           ID: 'flow1',
3           name: 'Flow 1',
4           indexKey: 'index',
5           valueKey: 'val',
6           flowColor: '#A1B2C3',
7           records: [
8             {index:1, val:2},
9             {index:2, val:7},
10            {index:3, val:3}
11          ]
12        }
```

  Only the ID field is mandatory; every other field can be omitted. Field description:

  - ID
    - Type: String;
    - Description: text that univocally identifies the flow in the graph;

    - Valid values: any non-empty string.
  - name
    - Type: String;
    - Description: name of the flow;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - indexKey
    - Type: String;
    - Description: defines which key represents the bar index in each record;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - valueKey
    - Type: String;
    - Description: defines which key represents the bar value in each record;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - flowColor
    - Type: String;
    - Description: color of the flow;
    - Default value: random;
    - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.

- *Object* records: the records for the flow.
  This parameter consists in an object containing several name: value pairs; the name of each pair can be chosen by the developer, and each of them can be either used as the bar index, the bar value or for filtering purposes.

Example usage:

```
1       barChartFlow1=barChart.createBarChartFlow(
2         {
3                   ID:'flow1',
4                   name: 'Time-pressure',
5                   indexKey: 'time',
6                   valueKey: 'pressure',
7                   flowColor: '#33AAFF'
8               },[
9                   {time: 1, pressure: 3},
10                  {time: 2, pressure: 10},
11                  {time: 3, pressure: 1},
12                  {time: 4, pressure: 5},
13                  {time: 5, pressure: 7}
14              ]
15          );
```

3.4.1.3 deleteFlow

This method deletes the flow that's identified by the given ID.
Return value: *boolean*: the deletion outcome
Parameters:

- *String* ID: the ID of the flow to delete.

Example usage:

```
graph1.deleteFlow('flow1');
```

3.4.1.4 deleteAllFlows

This method deletes all the graph flows.
Return value: *void*
Example usage:

```
graph1.deleteAllFlows();
```

3.4.1.5 updateRecord

This method updates a record in a flow.
Return value: *void*
Parameters:

- *String* flowID: the ID of the flow where the record is located;
- *int* index: the record index;
- *Object* record: the record that replaces the old one.

Example usage:

```
graph1.updateRecord('flow1', 2, {time: 3, pressure:
   10});
```

3.4.1.6 updateProperties

This method allows the user to update the graph properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
  title: 'bar1',
  width: 800,
  height: 600,
  enableLegend: true,
  barOrientation: 'V',
  backgroundColor: '#FFFFFF',
  legendOnPoint: true,
  grid: true,
  groupingControl: true,
  headers:[
```

```
            'col1',
            'col2',
            'col3'
          ],
          legend: {
            position: 'NE',
            fontColor: '#000000',
            backgroundColor: '#FFFFFF'
          },
          xAxis: {
            name: 'x',
            color: '#000000',
            minIndex: 5,
            maxIndex: 10,
            ticks: 10
          },
          yAxis: {
            name: 'y',
            color: '#000000',
            minIndex: 5,
            maxIndex: 10,
            ticks: 10
          }
        }
```

No field is mandatory. Field description:

- title
  - Type: String;
  - Description: title of the graph;
  - Default value: empty string;
  - Valid values: any non-empty string.
- width
  - Type: integer;
  - Description: width in pixel of the graph in the page;
  - Default value: 500;
  - Valid values: any positive integer.
- height
  - Type: integer;
  - Description: height in pixel of the graph in the page;
  - Default value: 400;
  - Valid values: any positive integer.
- enableLegend
  - Type: boolean;
  - Description: defines whether the legend is displayed or not in the graph;

    - Default value: false;
    - Valid values: true or false.
- barOrientation
    - Type: String;
    - Description: defines the bar orientation of the graphs;
    - Default value: 'V';
    - Valid values: 'V' (vertical) or 'H' (horizontal).
- backgroundColor
    - Type: String;
    - Description: defines the background color of the graph;
    - Default value: '#FFFFFF';
    - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.
- legendOnPoint
    - Type: boolean;
    - Description: defines whether the legend on point functionality is enabled or not;
    - Default value: false;
    - Valid values: true or false.
- grid
    - Type: boolean;
    - Description: defines whether the grid is displayed or not in the graph;
    - Default value: true;
    - Valid values: true or false.
- groupingControl
    - Type: boolean;
    - Description: defines whether the front end user is allowed to switch between grouped and stacked view;
    - Default value: false;
    - Valid values: true or false.
- headers
    - Type: Array of String;
    - Description: contains the list of the header labels;
    - Default value: [];
    - Valid values: any array of String objects.
- legend
    - Type: object;
    - Description: contains the legend configuration; refer to the legend section (3.12.1) for the structure description;

- xAxis
  - Type: object;
  - Description: contains the x axis configuration; refer to the axis section (3.12.2) for the structure description;
- yAxis
  - Type: object;
  - Description: contains the y axis configuration; refer to the axis section (3.12.2) for the structure description;

Example usage:

```
    barchart1.updateProperties({
        title: 'BAR',
        height: 600,
        width: 1000,
        enableLegend: true,
        legend: {
            position: 'NE',
        },
        xAxis:{
            name: 'time',
            color: '#000000',
            minIndex: 0,
            maxIndex: 7,
            scale:'logarithmic'
        },
        yAxis:{
            name: 'pressure',
            color: '#000000'
        },
        headers: [
            'h1',
            'h2',
            'h3',
            'h4',
            'h5'
        ],
        backGroundColor: '#FFFFFF',
        viewFinder: true,
        grid: true,
        legendOnPoint: true,
        barOrientation: 'V',
        sortable: true,
        groupingControl: true
    });
```

3.4.1.7 getProperties This method returns an object containing all of the graph properties.

Return value: *Object*: the object that contains the graph properties
Its structure is as follows:

```
{
  ID: 'bar1',
  title: 'bar1',
  width: 800,
  height: 600,
  enableLegend: true,
  backgroundColor: '#FFFFFF',
  legendOnPoint: true,
  horizontalGrid: true,
  verticalGrid: true,
  viewFinder: true,
  interpolation: 'monotone',
  legend: {
    position: 'NE',
    fontColor: '#000000',
    backgroundColor: '#FFFFFF'
  },
  xAxis: {
    name: 'x',
    color: '#000000',
    minIndex: 5,
    maxIndex: 10,
    ticks: 10
  },
  yAxis: {
    name: 'y',
    color: '#000000',
    minIndex: 5,
    maxIndex: 10,
    ticks: 10
  }
}
```

Example usage:

```
barchart1.getProperties();
```

3.5 LineChart

The LineChart class represents a Line Chart in the Norris framework$_{|g|}$. It's defined by several properties, and it contains the list of its flows.

3.5.1 Method overview

3.5.1.1 getFlowByID This method returns the flow that's identified by the given ID.
Return value: *LineChartFlow*: the desired flow
Parameters:

- *String* ID: the ID of the flow to return.

Example usage:

```
var flow1 = graph1.getFlowByID('flow1');
```

3.5.1.2 createLineChartFlow This method allows the user to create a flow inside a Line Chart; the properties of the flow are passed as a parameter to the method. Each flow represents a line in the graph, and each record represents a single point.
Return value: *LineChartFlow*: a reference to the created Line Chart flow.
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
  ID: 'flow1',
  name: 'Flow 1',
  xKey: 'x',
  yKey: 'y',
  flowColor: '#A1B2C3',
  marker: 'triangle'
}
```

  Only the ID field is mandatory; every other field can be omitted. Field description:

  - ID
    - Type: String;
    - Description: text that univocally identifies the flow in the graph;
    - Valid values: any non-empty string.
  - name
    - Type: String;
    - Description: name of the flow;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - xKey

    - Type: String;
    - Description: defines which key represents the x value in each record;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - yKey
    - Type: String;
    - Description: defines which key represents the y value in each record;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - flowColor
    - Type: String;
    - Description: color of the flow;
    - Default value: random;
    - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.
  - marker
    - Type: String;
    - Description: defines which shape the marker of each point should have;
    - Default value: 'none';
    - Valid values: 'none', 'square', 'triangle', 'circle', 'diamond'.

Example usage:

```
lineChartFlow=lineChart.createLineChartFlow({
        ID:'flow1',
        name: 'time-temperature',
        xKey: 'time',
  yKey: 'temperature',
        flowColor: '#B9D3EE',
        marker: 'triangle'
    });
```

3.5.1.3 deleteFlow

This method deletes the flow that's identified by the given ID.
Return value: *boolean*: the deletion outcome
Parameters:

- *String* ID: the ID of the flow to delete.

Example usage:

```
graph1.deleteFlow('flow1');
```

3.5.1.4 deleteAllFlows

This method deletes all the graph flows.
Return value: *void*
Example usage:

```
graph1.deleteAllFlows();
```

3.5.1.5 updateRecord

This method updates a record in a flow.
Return value: *void*
Parameters:

- *String* flowID: the ID of the flow where the record is located;
- *int* ID: the record ID;
- *Object* record: the record that replaces the old one.

Example usage:

```
graph1.updateRecord('flow1', 'flow1123456', {time: 3,
   temperature: 10});
```

3.5.1.6 addRecord

This method adds a record to a flow.
Return value: *String*: the ID generated for the new record
Parameters:

- *String* flowID: the ID of the flow where the record ought to be added;
- *Object* record: the record to add.

Example usage:

```
graph1.addRecord('flow1', {time: 3, temperature: 10});
```

3.5.1.7 updateProperties This method allows the user to update the graph properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
  ID: 'line1',
  title: 'line1',
  width: 800,
  height: 600,
  enableLegend: true,
  backgroundColor: '#FFFFFF',
  legendOnPoint: true,
  horizontalGrid: true,
  verticalGrid: true,
  viewFinder: true,
  interpolation: 'monotone',
  legend: {
    position: 'NE',
    fontColor: '#000000',
    backgroundColor: '#FFFFFF'
  },
  xAxis: {
    name: 'x',
    color: '#000000',
    minIndex: 5,
    maxIndex: 10,
    ticks: 10
  },
  yAxis: {
    name: 'y',
    color: '#000000',
    minIndex: 5,
    maxIndex: 10,
    ticks: 10
  }
}
```

  No field is mandatory. Field description:

  - title
    - Type: String;
    - Description: title of the graph;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - width

    - Type: integer;
    - Description: width in pixel of the graph in the page;
    - Default value: 500;
    - Valid values: any positive integer.
  - height
    - Type: integer;
    - Description: height in pixel of the graph in the page;
    - Default value: 400;
    - Valid values: any positive integer.
  - enableLegend
    - Type: boolean;
    - Description: defines whether the legend is displayed or not in the graph;
    - Default value: false;
    - Valid values: true or false.
  - backgroundColor
    - Type: String;
    - Description: defines the background color of the graph;
    - Default value: '#FFFFFF';
    - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.
  - legendOnPoint
    - Type: boolean;
    - Description: defines whether the legend on point functionality is enabled or not;
    - Default value: false;
    - Valid values: true or false.
  - horizontalGrid
    - Type: boolean;
    - Description: defines whether the horizontal grid is displayed or not in the graph;
    - Default value: true;
    - Valid values: true or false.
  - verticalGrid
    - Type: boolean;
    - Description: defines whether the vertical grid is displayed or not in the graph;
    - Default value: true;
    - Valid values: true or false.
  - viewFinder

    - Type: boolean;
    - Description: defines whether the view finder is enabled or not; Suggestion: only use the view finder with a sufficient amount of data;
    - Default value: false;
    - Valid values: true or false.
  - interpolation
    - Type: String;
    - Description: defines the type of interpolation to use on the graph;
    - Default value: 'linear';
    - Valid values: 'linear', 'step', 'basis', 'cardinal', 'monotone'.
  - legend
    - Type: object;
    - Description: contains the legend configuration; refer to the legend section (3.12.1) for the structure description;
  - xAxis
    - Type: object;
    - Description: contains the x axis configuration; refer to the axis section (3.12.2) for the structure description;
  - yAxis
    - Type: object;
    - Description: contains the y axis configuration; refer to the axis section (3.12.2) for the structure description;

Example usage:

```
linechart1.updateProperties(
{
  title: 'line1',
  width: 800,
  height: 600,
  enableLegend: true,
  backgroundColor: '#FFFFFF',
  legendOnPoint: true,
  horizontalGrid: true,
  verticalGrid: true,
  viewFinder: true,
  interpolation: 'monotone',
  legend: {
    position: 'NE',
    fontColor: '#000000',
    backgroundColor: '#FFFFFF'
  },
  xAxis: {
    name: 'x',
    color: '#000000',
```

```
        minIndex: 5,
        maxIndex: 10,
        ticks: 10
      },
      yAxis: {
        name: 'y',
        color: '#000000',
        minIndex: 5,
        maxIndex: 10,
        ticks: 10
      }
    });
```

3.5.1.8 getProperties This method returns an object containing all of the graph properties.
Return value: *Object*: the object that contains the graph properties
Its structure is as follows:

```
    {
      ID: 'line1',
      title: 'line1',
      width: 800,
      height: 600,
      enableLegend: true,
      backgroundColor: '#FFFFFF',
      legendOnPoint: true,
      horizontalGrid: true,
      verticalGrid: true,
      viewFinder: true,
      interpolation: 'monotone',
      legend: {
        position: 'NE',
        fontColor: '#000000',
        backgroundColor: '#FFFFFF'
      },
      xAxis: {
        name: 'x',
        color: '#000000',
        minIndex: 5,
        maxIndex: 10,
        ticks: 10
      },
      yAxis: {
        name: 'y',
        color: '#000000',
        minIndex: 5,
        maxIndex: 10,
        ticks: 10
```

```
31            }
32        }
```

Example usage:

```
1         linechart1.getProperties();
```

3.6 MapChart

The MapChart class represents a Map Chart in the Norris framework$_{|g|}$. It's defined by several properties, and it contains the list of its flows.

3.6.1 Method overview

3.6.1.1 getFlowByID This method returns the flow that's identified by the given ID.
Return value: *MapChartFlow*: the desired flow
Parameters:

- *String* ID: the ID of the flow to return.

Example usage:

```
1       var flow1 = graph1.getFlowByID('flow1');
```

3.6.1.2 createMapChartFlow This method allows the user to create a flow inside a Map Chart; the properties of the flow are passed as a parameter to the method. Each flow represents a group of markers in the graph, and each record represents a position of a marker. **Return value**: *MapChartFlow*: a reference to the created Map Chart flow.
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
1       {
2           ID:'flow1',
3           name: '22',
4           marker:{
5               'type': 'shape',
6           'shape': 'bus',
7               'icon': 'http://i.imgur.com/W0QgS4N.png'
8               'text': 'marker text',
9               'color' : '#FFC4F6'
10          },
11          trace:{
12          'type':'poly',
13          'coordinates': [
14                  [45.42946, 11.94090],[45.42941,
                      11.94081],[45.42955, 11.94065]
15          ],
16          'strokeColor' : '#DC271C',
17              'fillColor' : '#3a6d99'
18        },
19          latitudeKey: '1',
20          longitudeKey: '2',
21          objectKey: '0'
22      }
```

Only the ID field is mandatory; every other field can be omitted. Field description:

- ID
  - Type: String;
  - Description: text that univocally identifies the flow in the graph;
  - Valid values: any non-empty string.
- name
  - Type: String;
  - Description: name of the flow;
  - Default value: empty string;
  - Valid values: any non-empty string.
- marker
  - Type: object;
  - Description: contains the marker configuration; refer to the map marker section (3.12.5) for the structure description;
- trace
  - Type: object;
  - Description: contains the trace configuration; refer to the map trace section (3.12.6) for the structure description;
- latitudeKey
  - Type: String;
  - Description: defines which key represents the latitude in each record;
  - Default value: empty string;
  - Valid values: any non-empty string.
- longitudeKey
  - Type: String;
  - Description: defines which key represents the longitude in each record;
  - Default value: empty string;
  - Valid values: any non-empty string.
- objectKey
  - Type: String;
  - Description: defines which key represents the object name in each record;
  - Default value: empty string;
  - Valid values: any non-empty string.

Example usage:

```
1  var mapChartFlow=mapChart.createMapChartFlow({
2        ID:'flow1',
3        name: '22',
4        marker:{
```

```
            'type': 'shape',
        'shape': 'bus',
            'icon': 'http://i.imgur.com/W0QgS4N.png'
            'text': 'marker text',
            'color' : '#FFC4F6'
        },
        trace:{
        'type':'poly',
        'coordinates': [
                [45.42946, 11.94090],[45.42941,
                   11.94081],[45.42955, 11.94065]
        ],
        'strokeColor' : '#DC271C',
            'fillColor' : '#3a6d99'
      },
        latitudeKey: '1',
        longitudeKey: '2',
        objectKey: '0'
    });
```

3.6.1.3 deleteFlow This method deletes the flow that's identified by the given ID.
Return value: *boolean*: the deletion outcome
Parameters:

- *String* ID: the ID of the flow to delete.

Example usage:

```
graph1.deleteFlow('flow1');
```

3.6.1.4 deleteAllFlows This method deletes all the graph flows.
Return value: *void*
Example usage:

```
graph1.deleteAllFlows();
```

3.6.1.5 updateRecord This method updates a record in a flow.
Return value: *void*
Parameters:

- *String* flowID: the ID of the flow where the record is located;
- *int* ID: the record ID;
- *Object* record: the record that replaces the old one.

Example usage:

```
graph1.updateRecord('flow1', 'flow1123456', {lat: 45,
   lon: 50, object: 'bus1'});
```

3.6.1.6 updateMovie This method updates the flow in the movie paradigm.
Return value: *void*
Parameters:

- *String* flowID: the ID of the flow where the record is located;
- *Array* records: the records that update the flow.

Example usage:

```
graph1.updateMovie('flow1', [{lat: 45, lon: 50, object
   : 'bus1'}, {lat: 35, lon: 60, object: 'bus2'}]);
```

3.6.1.7 addRecord This method adds a record to a flow.
Return value: *String*: the ID generated for the new record
Parameters:

- *String* flowID: the ID of the flow where the record ought to be added;
- *Object* record: the record to add.

Example usage:

```
graph1.addRecord('flow1', {lat: 45, lon: 50, object: '
   bus1'});
```

3.6.1.8 updateProperties This method allows the user to update the graph properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
    title: 'APS',
    height: 600,
    width: 1000,
    enableLegend: true,
    legend: {
        position: 'NW',
        fontColor: '#00AA00',
        backgroundColor: '#FFAAFF'
    },
    latitude: 45.4113311,
    longitude: 11.8876318,
    mapType: 'roadmap',
    mapWidth: 2000,
    mapHeight: 2000,
    legendOnPoint: true
}
```

No field is mandatory. Field description:

- title
  - Type: String;
  - Description: title of the graph;
  - Default value: empty string;
  - Valid values: any non-empty string.
- width
  - Type: integer;
  - Description: width in pixel of the graph in the page;
  - Default value: 500;
  - Valid values: any positive integer.
- height
  - Type: integer;
  - Description: height in pixel of the graph in the page;
  - Default value: 400;
  - Valid values: any positive integer.
- enableLegend
  - Type: boolean;
  - Description: defines whether the legend is displayed or not in the graph;
  - Default value: false;
  - Valid values: true or false.
- legendOnPoint
  - Type: boolean;
  - Description: defines whether the legend on point functionality is enabled or not;
  - Default value: false;
  - Valid values: true or false.
- latitude
  - Type: number;
  - Description: defines the map center point latitude;
  - Default value: 0;
  - Valid values: any valid latitude value.
- longitude
  - Type: number;
  - Description: defines the map center point longitude;
  - Default value: 0;
  - Valid values: any valid longitude value.
- mapType
  - Type: String;

    - Description: defines the type of map displayed to the clients;
    - Default value: 'roadmap;
    - Valid values: 'roadmap', 'satellite', 'terrain', 'hybrid'.
  - mapWidth
    - Type: integer;
    - Description: defines the width in meters of the portion of map displayed;
    - Default value: 3000;
    - Valid values: any positive integer value.
  - mapHeight
    - Type: integer;
    - Description: defines the height in meters of the portion of map displayed;
    - Default value: 2000;
    - Valid values: any positive integer value.
  - legend
    - Type: object;
    - Description: contains the legend configuration; refer to the legend section (3.12.1) for the structure description;

Example usage:

```
mapchart1.updateProperties(
  {
      title: 'APS',
      height: 600,
      width: 1000,
      enableLegend: true,
      legend: {
          position: 'NW',
          fontColor: '#00AA00',
          backgroundColor: '#FFAAFF'
      },
      latitude: 45.4113311,
      longitude: 11.8876318,
      mapType: 'roadmap',
      mapWidth: 2000,
      mapHeight: 2000,
      legendOnPoint: true
  });
```

3.6.1.9 getProperties This method returns an object containing all of the graph properties.
Return value: *Object*: the object that contains the graph properties
Its structure is as follows:

```
{
  ID: 'map1',
  title: 'APS',
  height: 600,
  width: 1000,
  enableLegend: true,
  legend: {
    position: 'NW',
    fontColor: '#00AA00',
    backgroundColor: '#FFAAFF'
  },
  latitude: 45.4113311,
  longitude: 11.8876318,
  mapType: 'roadmap',
  mapWidth: 2000,
  mapHeight: 2000,
  legendOnPoint: true
}
```

Example usage:

```
mapchart1.getProperties();
```

3.7 Table

The Table class represents a Table in the Norris framework$_{|g|}$. It's defined by several properties, and it contains the list of its flows.

3.7.1 Method overview

3.7.1.1 getFlowByID This method returns the flow that's identified by the given ID.
Return value: *TableFlow*: the desired flow
Parameters:

- *String* ID: the ID of the flow to return.

Example usage:

```
1       var flow1 = graph1.getFlowByID('flow1');
```

3.7.1.2 createTableFlow This method allows the user to create a flow inside a Table; the properties of the flow are passed as a parameter to the method.
Each flow represents a group of rows in the graph, and each record represents a single row: therefore, they ought to contain as many elements as the number of column.
Return value: *TableFlow*: a reference to the created Table flow.
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
1       {
2               ID:'flow1',
3               name: 'autobus',
4               columnKeys: ['0','1','2'],
5               columnFormats: {'0':'toInt'},
6               maxItems: 20
7           }
```

  Only the ID field is mandatory; every other field can be omitted. Field description:

  - ID
    - Type: String;
    - Description: text that univocally identifies the flow in the graph;
    - Valid values: any non-empty string.
  - name
    - Type: String;
    - Description: name of the flow;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - columnKeys

    - Type: Array of Strings;
    - Description: defines which keys represents the value of each column in each record;
    - Default value: empty array;
    - Valid values: any non-empty array of strings.
  - columnFormats
    - Type: Object;
    - Description: defines which format each column should have; refer to the columnFormats section (3.12.7) for the structure description.
  - maxItems
    - Type: integer;
    - Description: max items contained in the flow;
    - Default value: 50;
    - Valid values: any integer number greater than 0.

Example usage:

```
var mapChartFlow=mapChart.createTableChartFlow({
        ID:'flow1',
        name: 'autobus',
        columnKeys: ['0','1','2'],
  columnFormats: {'0':'toInt'},
        maxItems: 20
    });
```

3.7.1.3 deleteFlow This method deletes the flow that's identified by the given ID.
Return value: *boolean*: the deletion outcome
Parameters:

- *String* ID: the ID of the flow to delete.

Example usage:

```
graph1.deleteFlow('flow1');
```

3.7.1.4 deleteAllFlows This method deletes all the graph flows.
Return value: *void*
Example usage:

```
graph1.deleteAllFlows();
```

3.7.1.5 updateRecord This method updates a record in a flow.
Return value: *void*
Parameters:

- *String* flowID: the ID of the flow where the record is located;

- *int* ID: the record ID;
- *Object* record: the record that replaces the old one.

Example usage:

```
graph1.updateRecord('flow1', 'flow1123456', {lat: 45,
   lon: 50, object: 'bus1'});
```

3.7.1.6 addRecord This method adds a record to a flow.
Return value: *String*: the ID generated for the new record
Parameters:

- *String* flowID: the ID of the flow where the record ought to be added;
- *Object* record: the record to add.

Example usage:

```
graph1.addRecord('flow1', {lat: 45, lon: 50, object: '
   bus1'});
```

3.7.1.7 updateProperties This method allows the user to update the graph properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
    ID: 'table1',
    title: 'Tabella',
    height: 600,
    width: 1000,
    sortable: true,
    maxItemsPage: 20,
    addRowOn: 'top',
    headers: ['IDMezzo', 'WGS84Fi', 'WGS84La'],
    sort: {
        column: ['IDMezzo', 'WGS84La'],
        ordering: ['DESC', 'ASC']
    },
    appearance: {
        horizontalGrid: {
            color: '#00AA00', //#xxxxxx,
            width: 1 // > 0
        },
        verticalGrid: {
            color: '#00AA00', //#xxxxxx,
            width: 1 // > 0
        },
        rowEven: {
            textColor: ['#00AB00', '#AA0000'],
              backgroundColor: ['#FAAFFF', '#
                 FFFAFF']
        },
        rowOdd: {
            textColor: ['#BB0000', '#BB0000'],
            backgroundColor: ['#AAFFFF', '#
               FAAAFF']
        },
        headers: {
            textColor: ['#00CC00', '#00CC00'],
            backgroundColor: ['#FFAAFF', '#
               FABCDF']
        }
    },
}
```

No field is mandatory. Field description:

- title
  - Type: String;
  - Description: title of the graph;
  - Default value: empty string;
  - Valid values: any non-empty string.
- width
  - Type: integer;
  - Description: width in pixel of the graph in the page;
  - Default value: 500;
  - Valid values: any positive integer.
- height
  - Type: integer;
  - Description: height in pixel of the graph in the page;
  - Default value: 400;
  - Valid values: any positive integer.
- sort
  - Type: object;
  - Description: defines the sorting rules; refer to the sort section (3.12.3) for the structure description;
- maxItemsPage
  - Type: integer;
  - Description: number of elements per each page of the Table;
  - Default value: 10;
  - Valid values: any positive integer.
- headers
  - Type: Array of String;
  - Description: contains the list of the header labels;
  - Default value: [];
  - Valid values: any array of String objects.
- addRowOn
  - Type: String;
  - Description: determines where the row are added to the Table;
  - Default value: 'bottom';
  - Valid values: 'top' or 'bottom'.
- appearance
  - Type: object;
  - Description: defines the appearance rules; refer to the appearance section (3.12.4) for the structure description;

Example usage:

```
linechart1.updateProperties({
      title: 'Tabella',
      height: 600,
      width: 1000,
      sortable: true,
      maxItemsPage: 20,
      addRowOn: 'top',
      headers: ['IDMezzo', 'WGS84Fi', 'WGS84La'],
      sort: [{
          column: 'IDMezzo',
          ordering: 'DESC'
      }],
      appearance: {
          border: {
              color: '#00AA00', //#xxxxxx,
              width: 1 // > 0
          },
          rowEven: {
              textColor: ['#00AB00', '#AA0000'],
                backgroundColor: ['#FAAFFF', '#FFFAFF'
                    ]
          },
          rowOdd: {
              textColor: ['#BB0000', '#BB0000'],
              backgroundColor: ['#AAFFFF', '#FAAAFF']
          },
          headers: {
              textColor: ['#00CC00', '#00CC00'],
              backgroundColor: ['#FFAAFF', '#FABCDF']
          }
      },
  });
```

3.7.1.8 getProperties This method returns an object containing all of the graph properties.
Return value: *Object*: the object that contains the graph properties
Its structure is as follows:

```
{
    ID: 'table1',
  title: 'Tabella',
  height: 600,
  width: 1000,
  sortable: true,
  maxItemsPage: 20,
  addRowOn: 'top',
  headers: ['IDMezzo', 'WGS84Fi', 'WGS84La'],
  sort: [{
```

```
            column: 'IDMezzo',
            ordering: 'DESC'
        }],
        appearance: {
            border: {
                color: '#00AA00', //#xxxxxx,
                width: 1 // > 0
            },
            rowEven: {
                textColor: ['#00AB00', '#AA0000'],
                  backgroundColor: ['#FAAFFF', '#FFFAFF']
            },
            rowOdd: {
                textColor: ['#BB0000', '#BB0000'],
                backgroundColor: ['#AAFFFF', '#FAAAFF']
            },
            headers: {
                textColor: ['#00CC00', '#00CC00'],
                backgroundColor: ['#FFAAFF', '#FABCDF']
            }
        },
    }
```

Example usage:

```
    table1.getProperties();
```

3.8 BarChartFlow

The BarChartFlow class represents a flow of a Bar Chart in the Norris framework$_{|g|}$.
It's defined by several properties, and it contains the list of its records.

3.8.1 Method overview

3.8.1.1 updateRecord This method updates a record in a flow.
Return value: *void*
Parameters:

- *int* index: the record index;
- *Object* record: the record that replaces the old one.

Example usage:

```
flow1.updateRecord(2, {time: 3, pressure: 10});
```

3.8.1.2 updateProperties This method allows the user to update the flow properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
  name: 'Flow 1',
  indexKey: 'index',
  valueKey: 'val',
  flowColor: '#A1B2C3'
}
```

  Field description:

  - name
    - Type: String;
    - Description: name of the flow;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - indexKey
    - Type: String;
    - Description: defines which key represents the bar index in each record;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - valueKey
    - Type: String;

  - Description: defines which key represents the bar value in each record;
  - Default value: empty string;
  - Valid values: any non-empty string.

- flowColor
  - Type: String;
  - Description: color of the flow;
  - Default value: random;
  - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.

Example usage:

```
flow1.updateProperties(
  {
    name: 'Flow 1',
    indexKey: 'index',
    valueKey: 'val',
    flowColor: '#A1B2C3'
  });
```

3.8.1.3 getProperties

This method returns an object containing all of the graph properties.
Return value: *Object*: the object that contains the graph properties
Its structure is as follows:

```
{
  ID: 'flow1',
  name: 'Flow 1',
  indexKey: 'index',
  valueKey: 'val',
  flowColor: '#A1B2C3'
}
```

Example usage:

```
flow1.getProperties();
```

3.9 LineChartFlow

The LineChartFlow class represents a flow of a Line Chart in the Norris framework$_{|g|}$. It's defined by several properties, and it contains the list of its records.

3.9.0.4 updateRecord This method updates a record in the flow.
Return value: *void*
Parameters:

- *int* ID: the record ID;
- *Object* record: the record that replaces the old one.

Example usage:

```
flow1.updateRecord('flow1123456', {time: 3,
   temperature: 10});
```

3.9.0.5 addRecord This method adds a record to the flow.
Return value: *String*: the ID generated for the new record
Parameters:

- *Object* record: the record to add.

Example usage:

```
flow1.addRecord({time: 3, temperature: 10});
```

3.9.0.6 updateProperties This method allows the user to update the flow properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
  name: 'Flow 1',
  xKey: 'x',
  yKey: 'y',
  flowColor: '#A1B2C3',
  marker: 'triangle'
}
```

  No field is mandatory. Field description:

  - name
    - Type: String;
    - Description: name of the flow;

    - Default value: empty string;
    - Valid values: any non-empty string.
  - xKey
    - Type: String;
    - Description: defines which key represents the x value in each record;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - yKey
    - Type: String;
    - Description: defines which key represents the y value in each record;
    - Default value: empty string;
    - Valid values: any non-empty string.
  - flowColor
    - Type: String;
    - Description: color of the flow;
    - Default value: random;
    - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.
  - marker
    - Type: String;
    - Description: defines which shape the marker of each point should have;
    - Default value: 'none';
    - Valid values: 'none', 'square', 'triangle', 'circle', 'diamond'.

Example usage:

```
flow1.updateProperties(
  name: 'Flow 1',
  xKey: 'x',
  yKey: 'y',
  flowColor: '#A1B2C3',
  marker: 'triangle'
});
```

3.9.0.7 getProperties This method returns an object containing all of the flow properties.
Return value: *Object*: the object that contains the flow properties
Its structure is as follows:

```
{
  ID: 'flow1',
  name: 'Flow 1',
  xKey: 'x',
  yKey: 'y',
```

```
        flowColor: '#A1B2C3',
        marker: 'triangle'
    }
```

Example usage:

```
    flow1.getProperties();
```

3.10 MapChartFlow

The MapChartFlow class represents a flow of a Map Chart in the Norris framework$_{|g|}$.
It's defined by several properties, and it contains the list of its records.

3.10.1 Method overview

3.10.1.1 updateRecord This method updates a record in the flow.
Return value: *void*
Parameters:

- *int* ID: the record ID;
- *Object* record: the record that replaces the old one.

Example usage:

```
1       flow1.updateRecord('flow1123456', {lat: 45, lon: 50,
           object: 'bus1'});
```

3.10.1.2 updateMovie This method updates the flow in the movie paradigm.
Return value: *void*
Parameters:

- *Array* records: the records that update the flow.

Example usage:

```
1       flow1.updateMovie([{lat: 45, lon: 50, object: 'bus1'},
            {lat: 35, lon: 60, object: 'bus2'}]);
```

3.10.1.3 addRecord This method adds a record to the flow.
Return value: *String*: the ID generated for the new record
Parameters:

- *Object* record: the record to add.

Example usage:

```
1       flow1.addRecord({lat: 45, lon: 50, object: 'bus1'});
```

3.10.1.4 updateProperties This method allows the user to update the flow properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the graph parameters.
  The params object has to respect the following structure:

```
{
    name: '22',
    marker:{
        'type': 'shape',
    'shape': 'bus',
        'icon': 'http://i.imgur.com/W0QgS4N.png'
        'text': 'marker text',
        'color' : '#FFC4F6'
    },
    trace:{
    'type':'poly',
    'coordinates': [
            [45.42946, 11.94090],[45.42941,
               11.94081],[45.42955, 11.94065]
    ],
    'strokeColor' : '#DC271C',
        'fillColor' : '#3a6d99'
  },
    latitudeKey: '1',
    longitudeKey: '2',
    objectKey: '0'
}
```

No field is mandatory. Field description:

- name
  - Type: String;
  - Description: name of the flow;
  - Default value: empty string;
  - Valid values: any non-empty string.
- marker
  - Type: object;
  - Description: contains the marker configuration; refer to the map marker section (3.12.5) for the structure description;
- trace
  - Type: object;
  - Description: contains the trace configuration; refer to the map trace section (3.12.6) for the structure description;
- latitudeKey
  - Type: String;
  - Description: defines which key represents the latitude in each record;
  - Default value: empty string;
  - Valid values: any non-empty string.
- longitudeKey

  - Type: String;
  - Description: defines which key represents the longitude in each record;
  - Default value: empty string;
  - Valid values: any non-empty string.

- objectKey
  - Type: String;
  - Description: defines which key represents the object name in each record;
  - Default value: empty string;
  - Valid values: any non-empty string.

Example usage:

```
flow1.updateProperties(
  {
      name: '22',
      marker:{
          'type': 'shape',
      'shape': 'bus',
          'icon': 'http://i.imgur.com/W0QgS4N.png'
          'text': 'marker text',
          'color' : '#FFC4F6'
      },
      trace:{
      'type':'poly',
      'coordinates': [
              [45.42946, 11.94090],[45.42941,
                11.94081],[45.42955, 11.94065]
      ],
      'strokeColor' : '#DC271C',
          'fillColor' : '#3a6d99'
    },
      latitudeKey: '1',
      longitudeKey: '2',
      objectKey: '0'
  });
```

3.10.1.5 getProperties This method returns an object containing all of the graph properties.
Return value: *Object*: the object that contains the graph properties
Its structure is as follows:

```
{
    ID:'flow1',
  name: '22',
    marker:{
        'type': 'shape',
  'shape': 'bus',
```

```
            'icon': 'http://i.imgur.com/W0QgS4N.png'
            'text': 'marker text',
            'color' : '#FFC4F6'
        },
        trace:{
        'type':'poly',
        'coordinates': [
                [45.42946, 11.94090],[45.42941,
                  11.94081],[45.42955, 11.94065]
        ],
        'strokeColor' : '#DC271C',
            'fillColor' : '#3a6d99'
      },
        latitudeKey: '1',
        longitudeKey: '2',
        objectKey: '0'
    }
```

Example usage:

```
    flow1.getProperties();
```

3.11 TableFlow

The TableFlow class represents a flow of a Table in the Norris framework$_{|g|}$. It's defined by several properties, and it contains the list of its flows.

3.11.1 Method overview

3.11.1.1 updateRecord

This method updates a record in the flow.
Return value: *void*
Parameters:

- *int* ID: the record ID;
- *Object* record: the record that replaces the old one.

Example usage:

```
1       flow1.updateRecord('flow1123456', {lat: 45, lon: 50,
           object: 'bus1'});
```

3.11.1.2 addRecord

This method adds a record to the flow.
Return value: *String*: the ID generated for the new record
Parameters:

- *Object* record: the record to add; inside this record the developer may add an additional field, called appearance, that contains an array, as big as the number of columns, composed of objects that contain two fields: bg and text. These fields must contain a valid color, and will change the appearance of respectively the background and the text color for each cell of the record, in the given order.

Example usage:

```
1       flow1.addRecord({
2         lat: 45,
3         lon: 50,
4         object: 'bus1',
5         appearance: [
6           {bg:'#FFFFFF', text:'#000000'},
7           {bg:'#FFFFFF', text:'#000000'},
8           {bg:'#FFFFFF', text:'#000000'}
9         ]
10      });
```

3.11.1.3 updateProperties

This method allows the user to update the flow properties according to what's specified in the passed parameters.
Return value: *void*
Parameters:

- *Object* params: the flow parameters.
  The params object has to respect the following structure:

```
        {
                name: 'autobus',
                columnKeys: ['0','1','2'],
                columnFormats: {'0':'toInt'},
                maxItems: 20
            }
```

No field is mandatory.Field description:

- ID
  - Type: String;
  - Description: text that univocally identifies the flow in the graph;
  - Valid values: any non-empty string.
- name
  - Type: String;
  - Description: name of the flow;
  - Default value: empty string;
  - Valid values: any non-empty string.
- columnKeys
  - Type: Array of Strings;
  - Description: defines which keys represents the value of each column in each record;
  - Default value: empty array;
  - Valid values: any non-empty array of strings.
- columnFormats
  - Type: Object;
  - Description: defines which format each column should have; refer to the columnFormats section (3.12.7) for the structure description.
- maxItems
  - Type: integer;
  - Description: max items contained in the flow;
  - Default value: 50;
  - Valid values: any integer number greater than 0.

Example usage:

```
    flow1.updateProperties({
            name: 'autobus',
            columnKeys: ['0','1','2'],
            columnFormats: {'0':'toInt'},
            maxItems: 20
        });
```

3.11.1.4 getProperties This method returns an object containing all of the flow properties.
Return value: *Object*: the object that contains the flow properties
Its structure is as follows:

```
{
        ID:'flow1',
        name: 'autobus',
        columnKeys: ['0','1','2'],
        columnFormats: {'0':'toInt'},
        maxItems: 20
    }
```

Example usage:

```
flow1.getProperties();
```

3.12 Extra objects

These objects are not classes of the framework, but they're used inside the method parameters, and the purpose of this section is explaining their structure.

3.12.1 Legend

The legend object describes the legend features in the BarChart, LineChart and MapChart classes. It is passed as a parameter to the functions that create the graphs and update their properties.
Field description:

- position
  - Type: String;
  - Description: describes the position of the legend inside the graph;
  - Default value: 'NE';
  - Valid values: 'N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW'.
- fontColor
  - Type: String;
  - Description: color of the text contained inside the legend;
  - Default value: '#000000';
  - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.
- backgroundColor
  - Type: String;
  - Description: color of the legend background;
  - Default value: '#000000';
  - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.

3.12.2 Axis

The axis object describes the axis configuration in the BarChart and LineChart classes. It is passed as a parameter to the functions that create the graphs and update their properties.
Field description:

- name
  - Type: String;
  - Description: the name of the axis;

  - Default value: ”;
  - Valid values: any non-empty string.

- color
  - Type: String;
  - Description: color of the axis line;
  - Default value: ’#000000’;
  - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form (’#FFFFFF’) and the 3 digit form (’#FFF’) are valid.

- minIndex
  - Type: number;
  - Description: minimum index represented on the axis;
  - Default value: null;
  - Valid values: any number.

- maxIndex
  - Type: number;
  - Description: maximum index represented on the axis;
  - Default value: null;
  - Valid values: any number.

- ticks
  - Type: integer;
  - Description: number of ticks represented on the axis;
  - Default value: 10;
  - Valid values: any positive integer number.

3.12.3 Sort

The sort object describes the sorting rules in the Table class. It is passed as a parameter to the functions that create the tables and update their properties.
Field description:

- column
  - Type: array of strings;
  - Description: names of the columns by which the table ought to be ordered; the order of the columns determines the ordering priority, with the first one having the highest priority;
  - Valid values: the names of some of the table columns.

- ordering
  - Type: array of strings;
  - Description: the type of orderings applied for the columns;
  - Valid values: 'ASC' or 'DESC'.

3.12.4 Appearance

The appearance object describes how a table is expected to look like, in terms of colors and borders. It is passed as a parameter to the functions that create the tables and update their properties.
Field description:

- horizontalGrid
  - Type: Object;
  - Description: appearance of the table horizontal grid;

  Fields:

  - color
    - Type: String;
    - Description: color of the table horizontal grid;
    - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.
  - width
    - Type: integer;
    - Description: width of the table borders;
    - Valid values: any positive integer.
- verticalGrid
  - Type: Object;
  - Description: appearance of the table horizontal grid;

  Fields:

  - color
    - Type: String;
    - Description: color of the table vertical grid;
    - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.
  - width
    - Type: integer;
    - Description: width of the table borders;

    * Valid values: any positive integer.

- rowEven
  - Type: Object;
  - Description: appearance of the table even rows;

  Fields:

  - textColor
    * Type: array of strings;
    * Description: color of the rows text; each element of the array represents the style of each cell, in order;
    * Valid values: for each array element, any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid. The developer can either use a single element, which will be used for all the cells, or as many elements as there are columns.
  - backgroundColor
    * Type: String;
    * Description: color of the rows background; each element of the array represents the style of each cell, in order;
    * Valid values: for each array element, any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid. The developer can either use a single element, which will be used for all the cells, or as many elements as there are columns.

- rowOdd
  - Type: Object;
  - Description: appearance of the table odd rows;

  Fields:

  - textColor
    * Type: String;
    * Description: color of the rows text; each element of the array represents the style of each cell, in order;
    * Valid values: for each array element, any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid. The developer can either use a single element, which will be used for all the cells, or as many elements as there are columns.
  - backgroundColor
    * Type: String;
    * Description: color of the rows background; each element of the array represents the style of each cell, in order;

        * Valid values: for each array element, any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid. The developer can either use a single element, which will be used for all the cells, or as many elements as there are columns.

- headers
  - Type: Object;
  - Description: appearance of the table header rows;

  Fields:

  - textColor
    * Type: String;
    * Description: color of the rows text; each element of the array represents the style of each cell, in order;
    * Valid values: for each array element, any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid. The developer can either use a single element, which will be used for all the cells, or as many elements as there are columns.
  - backgroundColor
    * Type: String;
    * Description: color of the rows background; each element of the array represents the style of each cell, in order;
    * Valid values: for each array element, any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid. The developer can either use a single element, which will be used for all the cells, or as many elements as there are columns.

3.12.5 Map marker

This object describes the marker appearance in a MapChart flow. It is passed as a parameter to the functions that create the flows and update their properties.
Field description:

- type
  - Type: String;
  - Description: type of the marker;
  - Default value: 'shape';
  - Valid values: 'shape', 'icon', 'text'.
- shape
  - Type: String;

  - Description: the shape of the marker; it should only be defined if the type field is set to 'shape';
  - Default value: 'circle';
  - Valid values: 'square', 'triangle', 'circle', 'diamond', 'bus', 'plane', 'ship', 'car', 'truck', 'flag, 'man, 'woman'.

- icon
  - Type: String;
  - Description: the URL of the image for the marker; it should only be defined if the type field is set to 'icon';
  - Default value: '';
  - Valid values: any string.

- text
  - Type: String;
  - Description: the text that the marker should contain; it should only be defined if the type field is set to 'text';
  - Default value: '';
  - Valid values: any string.

- color
  - Type: String;
  - Description: the color of the marker text; it should only be defined if the type field is set to 'text';
  - Default value: '';
  - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.

3.12.6 Map trace

This object describes the trace appearance in a MapChart flow. It is passed as a parameter to the functions that create the flows and update their properties.
Field description:

- type
  - Type: String;
  - Description: type of the marker;
  - Default value: 'none';
  - Valid values: 'none', 'area', 'poly'.

- coordinates

  - Type: array of arrays;
  - Description: inside this array are several arrays that, in the order they're listed, represent the coordinates of the trace; each of these array contains two numeric values that represent the latitude and longitude of each point;
  - Default value: [];
  - Valid values: any array of arrays made of couples of numbers.

- strokeColor
  - Type: String;
  - Description: the color of the trace stroke; should be used when the trace type is 'poly';
  - Default value: '';
  - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.

- fillColor
  - Type: String;
  - Description: the color of the trace area; should be used when the trace type is 'area';
  - Default value: '';
  - Valid values: any String object containing a color represented in Hexadeciamal format; both the 6 digits form ('#FFFFFF') and the 3 digit form ('#FFF') are valid.

3.12.7 Column formats

This object describes the format of a table columns. It is passed as a parameter to the functions that create the tables and update their properties. The object is composed of key:value pairs, where the key is the name of the column and the value is the desired format.
The format valid values are 'toInt' or 'toFloat'.
Example:

```
{
  'col1': 'toInt',
  'col2': 'toFloat'
}
```

3.13 Error codes

During the usage of the framework, several types of errors might occur; in order to handle this, the framework includes a set of error codes that identify which problem happened. Each code error will be printed on the console, and in some cases returned by the functions that cause them.

3.13.1 Record errors

- **111**: in a BarChartFlow, invalid record on update
- **112**: in a BarChartFlow, invalid index on update
- **121**: in a LineChartFlow, invalid record on update
- **122**: in a LineChartFlow, invalid ID on update
- **123**: in a LineChartFlow, invalid record on addition
- **131**: in a MapChartFlow, invalid record on update
- **132**: in a MapChartFlow, invalid ID on update
- **133**: in a MapChartFlow, invalid record on addition
- **134**: in a MapChartFlow, invalid ID on deletion
- **141**: in a TableFlow, invalid record on update
- **142**: in a TableFlow, invalid ID on update
- **143**: in a TableFlow, invalid ID on addition
- **151**: in a Flow, invalid ID on getRecordByID

3.13.2 Flow errors

- **211**: in a BarChart, flow not found
- **212**: in a BarChart, invalid parameters to the constructor
- **221**: in a LineChart, flow not found
- **222**: in a LineChart, invalid parameters to the constructor
- **231**: in a MapChart, flow not found
- **232**: in a MapChart, invalid parameters to the constructor
- **233**: in a MapChart, invalid parameters on the properties update
- **241**: in a Table, flow not found
- **242**: in a Table, invalid parameters to the constructor

- **251**: in a Flow, invalid parameters to the constructor
- **252**: in a Flow, invalid parameters on the properties update
- **262**: in a BarChart, invalid ID on getFlowByID
- **263**: in a BarChart, invalid ID on flow deletion deletion
- **272**: in a LineChart, invalid ID on getFlowByID
- **273**: in a LineChart, invalid ID on flow deletion deletion
- **281**: in a MapChart, invalid parameters on creation
- **282**: in a MapChart, invalid ID on getFlowByID
- **283**: in a MapChart, invalid ID on flow deletion
- **201**: in a Flow, invalid graph socket passed to the constructor

3.13.3 Graph errors

- **361**: in a BarChart, invalid parameters to the constructor
- **362**: in a BarChart, invalid graph socket passed to the constructor
- **363**: in a BarChart, invalid page passed to the constructor
- **371**: in a LineChart, invalid parameters to the constructor
- **372**: in a LineChart, invalid graph socket passed to the constructor
- **373**: in a LineChart, invalid page passed to the constructor
- **381**: in a MapChart, invalid parameters passed to the constructor
- **382**: in a MapChart, invalid graph socket passed to the constructor
- **383**: in a MapChart, invalid page passed to the constructor
- **301**: in a Flow, invalid graph socket passed to the constructor
- **302**: in a Flow, invalid page passed to the constructor

3.13.4 Page errors

- **421**: in a Page, invalid parameters passed to the constructor

3.13.5 Norris errors

- **421**: in a Norris object, invalid parameters passed to the constructor

3.13.6 Filter errors

- **611**: in a Filter, a condition is not a string or it's empty
- **612**: in a Filter, a condition is not well formed
- **621**: in a Filter, the conditions are not a string or they're empty
- **622**: in a Filter, there's no valid condition

4 Web integration

The Norris framework comes with a Web client that has to be configured properly for the browser user to see the streamed data.

4.1 App installation

The Norris-nrti Web-App can be found in our repository$_{|g|}$ at the following url$_{|g|}$: github.com/Deltagraphs/norris-nrti/frontend. Search for the file named **norris-nrti.js** and refer to it with no additional charge. The application requires the following minimal features in order to work:

- a browser$_{|g|}$that supports AngularJS$_{|g|}$ and its functionalities;
- an Internet$_{|g|}$ connection.

The developer has to perform the following preliminary actions to insert his web-app into a site:

- Creation of a reference to the web-app library, through a ¡link¿ tag in the document head.
  It can be done by adding the following rows to your code:

```
<link type="text/javascript" src="github.com/
   Deltagraphs/norris-nrti/frontend/norris-nrti.js
   "></link>
```

- Insertion of AngularJS and Norris-nrti directives in the document body through the following code:

```
<body ng-app='norris-nrti' ng-controller="
   RootController" ng-init="url = 'url.del.server/
   da/dove/arrivano/i/dati'">
  <div ng-view></div>
</body>
```

4.2 What is Web-App Norris-nrti

As previously introduced, Norris-nrti is a web-app that displays data, arranged in several types of graph, which updates in real time. The main purpose of the app is showing data to the user in a thorough yet simple way. Further explanations on the app will follow in the next chapters, with a step-to-step guide to interaction supplied by images.

5 Problems and malfunctioning

The application is still in its early stage, so the user might run into some runtime bugs or technical issues when using some devices. Should any malfunctioning be found, we ask you to please send us an e-mail with the problem description at the following e-mail address: deltagraphs@gmail.com.

6 Glossary

6.1 A

6.2 B

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. A vertical bar chart is sometimes called a column bar chart.

6.3 C

6.4 D

6.5 E

Express.js: is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

6.6 F

Framework: is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. A software framework is a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate development of software applications, products and solutions.

6.7 G

6.8 H

6.9 I

6.10 J

6.11 K

6.12 L

Line chart: is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments.

6.13 M

Map chart: is a type of chart which displays information about a map and its markers.

6.14 N

6.15 O

6.16 P

6.17 Q

6.18 R

6.19 S

Socket.IO: enables real-time bidirectional event-based communication. It works on every platform, browser or device, focusing equally on reliability and speed.

6.20 T

6.21 U

URL: a Uniform Resource Locator (URL) is a reference to a resource that specifies the location of the resource on a computer network and a mechanism for retrieving it.

6.22 V

6.23 W

6.24 X

6.25 Y

6.26 Z