Documentation interne

## Listes des modules

L’application est constituée de 2 parties principales. L’une qui est placée dans la librairie, est en fait un dérivé de classe qui produit 3 composants similaires.

On a ainsi dans cette librairie :

* Basic-linechart.ts : Classe générique comprenant la plupart des méthodes D3js.
* Number-linechart.components.ts, boolean-linechart.components.ts et enumeration-linechart.components.ts qui sont les 3 classes et composants dérivés de la générique.

Par ailleurs on a App.components.ts qui est sera a terme le composant principal que l’on pourra intégrer en tant que module. Ce composant utilise la librairie composée des 4 classes présentées précédemment.

## Objectifs des modules

Le module app.component est le module principal. Il peut pour l’instant être utilisé tel qu’il est fourni, mais n’est pas indispensable pour l'utilisation d’un composant de type linechart.

Il est présent uniquement pour faciliter le codage, les tests et la présentation des composants.

Par ailleurs, il introduit une approche en tant que composant unique. Ainsi on a plus à se soucier des 3 sous-composants et on peut alors modifier et introduire les graphes (linechart) depuis ce composant. Il propose donc un exemple de module avec diverses fonctions qui le définit.

Les composants Number-linechart.components.ts, boolean-linechart.components.ts et enumeration-linechart.components.ts sont le cœur des linechart.

Dérivés de basic-linechart, ils spécifient les particularités et différences entre les trois types de composant que l’on peut représenter, à savoir, les nombres, les booléens et les énumérations.Ainsi toute création de composant se fera par un de ces 3 composants.

## Définition des types

Pour la définition des types, on a regroupé la plupart de ces définitions dans un fichier nommé “interfaces.ts”. Il contient toutes les interfaces utilisées et qui servent à la construction des types utilisés :

* Interface DataG<T> : Interface de base nécessaire à l’implémentation des autres types.

export interface DataG<T> {

style: string;

label: string;

values: [number, T][];

interpolation?: unknown; // Non requis pour tout

color?: unknown

colors?: unknown

}

* Interface DataNumber : pour les linechart représentant des nombres.
* export interface DataNumber extends DataG<number> {
* style: "number";
* label: string;
* values: [number, number][];
* color: string; // La couleur de la courbe
* interpolation: "linear" | "step";
* }
* Interface DataBool : pour les linechart représentant des booléens.
* export interface DataBool extends DataG<boolean> {
* style: "boolean";
* label: string;
* values: [number, boolean][];
* color: string; // La couleur de quand c'est vrai
* }
* Interface DataEnum : pour les linechart représentant des énumérations.

export interface DataEnum<T extends string[]> extends DataG<T[number]> { //interface DataEnum<T extends string[]> extends DataG<T[keyof T]> {

style: "enumeration";

label: string;

// values: [number, T[number]][];

colors: {

[k in T[number]]: string; // Couleur de chaque valeur

};

}

* Le type générique qui est utilisé dans basic-linechart.ts

export type Data<T> =

T extends number ? DataNumber:

T extends boolean ? DataBool :

T extends string[] ? DataEnum<T> : never;

* L’interface CONFIG, qui gère tous les éléments de configuration qui vont communiquer entre le composant principal (ici par exemple app.component) et les sous composants.

export interface CONFIG {

width: number;

height: number;

domainY: [number, number];

speedZoom: number;

range: [number,number];

currentTime: number;

scrollBar: boolean;

knobCurrentTime: boolean;

peakSize: number

}

* L’interface colorMap : pour implémenter des couleurs par défaut. Les chaînes de caractères sont utilisées pour des couleurs définies en Hexadécimal.

export interface colorMap {

sunny: string;

rainy: string;

cloudy: string;

lineIndex: string[];

}

* Interfaces secondaires, facilitant la création de polygones, notamment pour les énumérations et booléens.

export interface polygonDef {

"name": string;

"points": points[];

"color": string;

}

export interface points {

"x": number;

"y": number;

}

Il y a en plus, une dernière interface, présente seulement dans la classe app.component, car uniquement utile pour la facilité d’utilisation des composants, une fois des données intégrées :

interface linechart {

dataStyle: string;

dataDisplay: number,

name: string,

config: Partial<CONFIG>,

peakSize: number

}

## Procédures externes

Il n’y a que deux méthodes, qui communiquent directement avec les sous composants présents dans la librairie :

public updateRange(range: [number,number]){

this.config = {...this.config, range};

}

public updateCurrentTime(currentTime: number ) {

this.config = {...this.config, currentTime};

}

Elles notifient directement tout changement fait sur le zoom et sur la position du curseur indiquant le temps visualisé.

Dans les composants on a une connection de ce type :

/\*

\* Output rangeChange that emit range

\*/

@Output() override rangeChange = new EventEmitter<[number,number]>();

/\*

\* Output currentTimeChange that emit currentTime

\*/

@Output() override currentTimeChange = new EventEmitter<number>();

## Variables externes

Concernant les variables qui communiquent, on à l’objet config, défini également dans les 3 composants comme tel :

@Input()

override get config(): Partial<CONFIG> {return this.\_config;}

override set config(c: Partial<CONFIG>) {

this.\_config = {...defaultConfig, ...c};

}