Visual

Visual es una biblioteca de Javascript para la visualización de datos desarrollada por el Idescat. Se basa en soluciones populares de código abierto. **Visual** ofrece una interfaz sencilla que encapsula la complejidad de dichas soluciones para el tipo de gráficos más comunes.

El Idescat utiliza esta biblioteca en las visualizaciones de datos que publica en su sitio web.

1. Código con licencia abierta

El código fuente de **Visual** está publicado con una licencia abierta en GitHub. Allí se puede descargar libremente para utilizarlo o modificarlo. Si quiere contribuir a mejorar la biblioteca, puede bifurcar el proyecto en GitHub, introducir cambios y solicitar su actualización con sus cambios.

2. Gráficos y mapas

Visual puede representar una gran variedad de visualizaciones, como mapas, gráficos de líneas, de barras o pirámides de población.

Catálogo de visualizaciones

Distribuciones
Series temporales
Rankings
Pirámides
Mapas

Una de las funcionalidades de **Visual** es la elaboración de **mapas de coropletas**. La biblioteca proporciona los siguientes mapas de Cataluña:

- Cataluña por municipios
- Cataluña por comarcas
- Cataluña por provincias
- 41 comarcas por municipios
- 4 provincias por municipios

Visual también le permite incorporar sus propios mapas.

3. Instalación y uso

Para comenzar a probar **Visual**, descárguese el código completo, consulte los ejemplos de la carpeta *test* y lea la documentación.

Adapte el fichero de configuración de acuerdo con sus características y necesidades.

4. Proyectos relacionados

La apertura del código fuente de **Visual** complementa las iniciativas de apertura de datos del Idescat, como la exposición de parte de sus datos a través de una colección de APIs o la puesta a disposición de otras webs de un conjunto de widgets.

5

5. The Visual object

Visual is executed by passing a visual object, or an array of visual objects, to the *visual* function.

```
visual( {...} )
```

If you already have a defined *visual* function, you can run **Visual** like this:

```
VisualJS.load( {...} )
```

The visual object accepts the following properties:

General properties

lang

String ("ca", "es", "en"). Language. Default (*ca*) is set in visual.setup.js.

title

String. Text of first level title.

footer

String. Text of footer.

geo

String. Geographical area.

time

String (optional) or array of strings (required). Time period or periods.

Visual will treat the following string time formats using the "quarter" and "month" properties in visual.setup.js: "20131" (first quarter of 2013) and "201301" (January 2013). Any other time pattern will be displayed untreated.

autoheading

Boolean. This determines whether the heading is built by composition from "title", "geo" and "time". If *false*, only "title" will be used as a heading. Default (*true*) is set in visual.setup.js.

legend

Boolean. This determines whether the chart legend should be shown. Default (*true*) is set in visual.setup.js.

grid

Object with the following properties: *border* (number: grid border width), *line* (number: line width), *shadow* (number: line shadow width) and *point* (number: point radius). Default border (*0*), line (*2*), shadow (*4*) and point (*1*) are set in visual.setup.js.

axis

Object with two properties: *x* (boolean) and *y* (boolean). These determine whether the axes should be shown. Default (*true*) is set in visual.setup.js.

unit

Object with three properties: *label* (string), *symbol* (string) and symbol *position* (string. "start", "end"). All properties are optional. Default (no label, no symbol, position: end) is set in visual.setup.js.

Warning: *label* and *symbol* cannot contain HTML entities when *type* is "cmap".

dec

Number. Number of decimals in the data. This is used in the tooltip and map legend. Although this is an optional property, it is highly recommended that the number of decimals is specified: otherwise, all unneeded trailing zeros will be removed and computed values could be shown with more decimals than the original values. Default value can be set in visual.setup.js.

type

String ("bar", "rank", "tsbar", "tsline", "pyram", "cmap"). Required. Chart type. This determines the *data* and *time* formats and the specific properties available.

data

Array. Required. This includes the data values but also series labels and IDs. The format is determined by *type*.

id

String. In simple mode and manual mode, this is the ID of the HTML element where the visualization has to be embedded.

fixed

Array. In simple mode and manual mode, this is the *[width, height]* in pixels of the visualization container.

callback

Function. This function will be called after the chart has been drawn. The *this* keyword will point to an object with the following properties: "id" (the chart's id: string), "chart" (a boolean indicating whether the chart is drawable or not: if false, VisualJS.chart would not be defined), "heading" (the HTML of the heading) and "legend" (an object with legend information).

The "legend" object is null unless the chart type is "cmap", data includes a *val* property and grouped is not specified. In this case, colors are automatically assigned to map areas.

When "legend" is not null, it has three properties (three arrays of size two): "color", "text" and "symbol". The first element in each array exposes information about the lighter color and the second element about the darker one. The elements of the "color" array are objects with three properties (numbers): "r", "g" and "b" (RGB color). The elements of the "text" array are strings (value plus unit information the lighter/darker color has been assigned to). The elements of the "symbol" array are three possible strings: "<=", "==" or ">=" (the comparison operators associated with the "text" elements). See example adv-05.html.

The *callback* property will be ignored if it is included in a visual object passed to VisualJS.iframe.

Examples: adv-01.html, adv-02.html, adv-03.html, adv-05.html.

show

Boolean. This determines whether the chart should be shown. When *false*, all the necessary files will be included but the chart will not be inserted: you will need to use a callback function that executes VisualJS.chart at some point. Default is *true*.

```
VisualJS.load({
    show: false,
    callback: function(){
        if(this.chart && window.confirm("Are you sure you want to see this chart?")){
            VisualJS.chart();
        }
    },
    ...
});
```

bar properties

Distribution of a categorical variable (vertical bar chart).

```
visual({
   title: "BAR example",
   geo: "Alt Camp",
   time: "2012",
   footer: "The source goes here.",
   unit: {label: "persones"},
   dec: 0,
   type: "bar",
   data : [
      ["0-14 anys", 7329],
      ["15-64 anys", 30231],
      ["65-84 anys", 6485],
      ["85 anys o més", 1254]
   ]
   /* Same as:
      by: [ "0-14 anys", "15-64 anys", "65-84 anys", "85 anys o més" ],
      data: [ 7329, 30231, 6485, 1254 ]
   */
});
```

Examples: bar.html

time

String. Time period.

data

Array of numbers or array of arrays. Required. If *by* has not been specified, the array contains as many elements as categories and each element is an array with two elements: a string (label) and a number (value). Otherwise, it is an array of numbers (values) while their label is specified in the *by* property.

by

Array of strings. See the *data* property.

range

Array of numbers (minimun, maximum). The first element must be lower than the second. This array sets the range of the y-axis.

rank properties

Ranking (horizontal bar chart).

```
visual({
   title: "RANK example (40 data)",
   geo: "Catalonia",
   time: "2009",
   footer: "The source goes here.",
   unit: {label: "milers", symbol: "€"},
   dec: 0,
   type: "rank",
   data : [
      ["Val d'Aran", 20300],
      ["Pallars Jussà", 19300],
      ["Ripollès", 19100],
      ["Urgell", 18900],
      ["Conca de Barberà", 18800],
      ["Gironès", 18700],
      ["Pallars Sobirà", 18700],
      ["Alta Ribagorça", 18600],
      ["Cerdanya", 18600],
      ["Garrotxa", 18600],
      ["Pla de l'Estany", 18600],
      ["Barcelonès", 18300],
      ["Priorat", 18300],
      ["Ribera d'Ebre", 18200],
      ["Segrià ", 18100],
      ["Garrigues", 18000],
      ["Baix Empordà", 17700],
      ["Maresme", 17700],
      ["Alt Camp", 17600],
      ["Noguera", 17600],
      ["Tarragonès", 17600],
      ["Terra Alta", 17600],
      ["Segarra", 17400],
      ["Alt Empordà", 17300],
      ["Baix Penedès", 17300],
      ["Solsonès", 17300],
      ["Vallès Occidental", 17300],
      ["Berguedà", 17200],
      ["Baix Camp", 17100],
      ["Pla d'Urgell", 17100],
      ["Montsià", 17000],
      ["Alt Penedès", 16900],
      ["Bages", 16900],
      ["Baix Ebre", 16900],
      ["Garraf", 16900],
      ["Alt Urgell", 16600],
      ["Selva", 16300],
      ["Osona", 16200],
      ["Vallès Oriental", 16200],
      ["Baix Llobregat", 16000],
      ["Anoia", 15800]
   ]
});
```

Examples: rank.html, rank10.html

time

String. Time period.

data

Array of arrays. Required. The first array contains as many elements as categories. Each element is an array of two elements: a string (label) and a number (value).

range

Number (multiplier) or array of numbers (minimun, maximum). By default, the multiplier is 1.02 (increase the x-axis by 2%). An array (where the first element must be lower than the second) can also be used to set the range of the x-axis.

tsbar properties

Stacked/non-stacked time series (vertical bar chart).

```
visual({
```

```
   title: "TSBAR example",
   geo: "Catalonia",
   time : [
      "1998", "1999", "2000", "2001", "2002",
      "2003", "2004", "2005", "2006", "2007",
      "2008", "2009", "2010", "2011", "2012"
   ],
   footer: "The source goes here.",
   unit: {label: "M", symbol: "€"},
   dec: 1,
   type: "tsbar",
   data : [
      {
         label: "Exportacions",
         val: [
            27147.8, 27890.6, 33796.5, 36694.5, 37275.9,
            37648.5, 39485.1, 42703.4, 47218.8, 49679.8,
            50515.7, 41461.7, 48871.6, 54999.9, 58321.7
         ]
      },
      {
         label: "Importacions",
         val: [
            36203.7, 40316.5, 48761.7, 50497.9, 51891.8,
            54344.7, 60731, 67813.3, 74787.8, 80363.4,
            77233.9, 57663.8, 67621.1, 72280.2, 69343.1
         ]
      }
   ]
});
```

Examples: tsbar.html, tsbar2.html, tsbarns.html

time

Array of strings. Required. Time periods.

data

Array of objects. Required. The array contains as many elements as series. Each element is an object with two properties: *label* (string) and *val* (array of values).

stacked

Boolean. Default: *false*. When bars are not stacked, only three series are allowed.

range

Array of numbers (minimun, maximum). The first element must be lower than the second. This array sets the range of the y-axis.

tsline properties

Time series (line chart).

```
visual({
   title: "TSLINE example",
   geo: "Catalonia",
   time : [
      "1998", "1999", "2000", "2001", "2002",
      "2003", "2004", "2005", "2006", "2007",
      "2008", "2009", "2010", "2011", "2012"
   ],
   footer: "The source goes here.",
   unit: {label: "M", symbol: "€"},
   dec: 1,
   type: "tsline",
   data : [
      {
         label: "Exportacions",
         val: [
            27147.8, 27890.6, 33796.5, 36694.5, 37275.9,
            37648.5, 39485.1, 42703.4, 47218.8, 49679.8,
            50515.7, 41461.7, 48871.6, 54999.9, 58321.7
         ]
```

```
      },
      {
         label: "Importacions",
         val: [
            36203.7, 40316.5, 48761.7, 50497.9, 51891.8,
            54344.7, 60731, 67813.3, 74787.8, 80363.4,
            77233.9, 57663.8, 67621.1, 72280.2, 69343.1
         ]
      }
   ]
});
```

Examples: tsline2.html

time

Array of strings. Required. Time periods.

data

Array of objects. Required. The array contains as many elements as series. Each element is an object with two properties: *label* (string) and *val* (array of values).

range

Array of numbers (minimun, maximum). The first element must be lower than the second. This array sets the range of the y-axis.

pyram properties

Population pyramid.

```
visual({
   title: "PYRAM example",
   geo: "A country",
   time: "2012",
   footer: "The source goes here.",
   unit: {label: "persones"},
   dec: 0,
   type: "pyram",
   by: [
      "0-4", "5-9", "10-14", "15-19", "20-24", "25-29",
      "30-34", "35-39", "40-44", "45-49", "50-54", "55-59",
      "60-64", "65-69", "70-74", "75-79", "80-84", "85-89",
      "90-94", "95-99", "100+"
   ],
   data: [
      {
         label: "Homes",
         val: [
            130229, 132460, 109072, 115983, 133972, 166757,
            207016, 211782, 195472, 176832, 152151, 122107,
            117375, 99405, 80274, 73283, 47597, 24195, 6997, 1532, 260
         ]
      },
      {
         label: "Dones",
         val: [
            124757, 112944, 103163, 104773, 122879, 152743,
            196767, 193411, 194849, 174780, 155177, 133712,
            126386, 117169, 98444, 99468, 76448, 47515,17929, 4284, 548
         ]
      }
   ]
});
```

Examples: pyram.html

time

String. Time period.

data

Array of objects. Required. The array contains two elements: one for each sex. Each element is an object with two properties: *label* (string) and *val* (array of values).

by

Array of strings. Required. Each element is an age label.

range

Number (multiplier) or array of numbers (minimun, maximum). By default, the multiplier is 1.02 (increase the x-axis by 2%). An array (where the first element must be lower than the second and it will be ignored if it is different than zero) can also be used to set the range of the x-axis.

cmap properties

Choropleth map

```
visual({
   title: "CMAP example with missing data",
   geo: "Catalonia",
   time: "2001",
   footer: "The source goes here.",
   unit: {symbol: "%"},
   type: "cmap",
   dec: 2,
   by: "com",
   data: [
      {id: "01", val: 85.50},
      {id: "02", val: 79.40},
      {id: "03", val: 80.91},
      {id: "04", val: 86.50},
      {id: "05", val: 83.01},
      {id: "06", val: 79.04},
      {id: "07", val: 82.74},
      {id: "08", val: 77.31},
      {id: "09", val: 86.48},
      {id: "10", val: 79.94},
      {id: "11", val: 65.79},
      {id: "12", val: 73.04},
      {id: "13", val: 70.35},
      {id: "14", val: 89.96},
      {id: "15", val: 84.79},
      {id: "16", val: 91.06},
      {id: "17", val: 75.31},
      {id: "18", val: 92.95},
      {id: "19", val: 89.95},
      {id: "20", val: 82.50},
      {id: "21", val: 77.03},
      {id: "22", val: 86.48},
      {id: "23", val: 90.73},
      {id: "24", val: 86.06},
      {id: "25", val: 88.94},
      {id: "26", val: 91.67},
      {id: "27", val: 88.38},
      {id: "28", val: 88.68},
      {id: "29", val: 92.49},
      {id: "30", val: 90.66},
      {id: "31", val: 88.24},
      {id: "32", val: 86.17},
      {id: "33", val: 83.71},
      {id: "34", val: 77.71},
      {id: "35", val: 90.53},
      {id: "36", val: 74.20},
      {id: "34", val: 91.87},
      {id: "38", val: 88.66},
      {id: "39", val: 77.98},
      {id: "40", val: 71.31},
      {id: "41", val: 75.56}
   ]
});
```

Examples: cmap.html, cmap-com.html, cmap-f0.html, cmap-f020.html, cmap-groups1.html, cmap-groups2.html

time

String. Time period.

data

Array of objects. Required. The array contains as many elements as map areas. Each element is an object with at least one property: the area *id* (string). In this case, a map will be created with the included areas highlighted. If *val* (number) is included, it will be used to automatically assign colors to areas, unless grouped has been specified. In that case, the *group* property (counter starting with 1) is required and will be used to assign colors, but *val* can still be specified if needed.

grouped

Object with at least one property: "label" (array of strings). Each element in this array is a group label string (the first label will be attached to areas with a *group* property of 1 in **data**, and so on). A second property ("color", array of strings) can be provided to assign a custom color to each group. Colors must be specified as three two-digit hexadecimal numbers, starting with a # sign (for example, "#000000" means black).

by

String. Required. Selects a certain map. Possible values ("mun", "com", "prov", etc.) are set in visual.setup.js.

range

Number or array. This determines the color assignment. When it is a number, it must be between 0 and 0.49. Default: 0.05, which means color assignment excludes values below the 5th percentile and above the 95th percentile. When it is an array, it defines a range: it has two and only two elements. The first (number) is a minimum and the second (number) is a maximum. Colors will be assigned between those values.

Maps

The following sample maps are provided:

- Catalonia by municipalities, counties and provinces (3 maps)
- Counties of Catalonia by municipalities (41 maps)
- United States of America by states (1 map)

These are stored in the maps folder. The map maker allows you to preview these maps and fine-tune them.

A map is a UTF-8 Javascript file that adds a new property (the name of the map) to the VisualJS.map object. The value of this new property is a Visual map object.

Visual map object

GeoJSON properties: features

Geographic information must be provided in the GeoJSON format: it must be a feature collection object (a GeoJSON object with the type "FeatureCollection"). Simply copy the "features" property of the GeoJSON object into the Visual map object.

Projection properties: projection, scale, center

"projection" (string) must be a valid D3 geo projection function name. "scale" is the projection scale (a number) and "center" is the projection center (a coordinate array).

```
projection: "mercator",
scale: 9000,
center: [1.74, 41.7],
```

If a projection does not support centering (for example, Albers USA), "center" is optional and, if present, will be ignored.

Visual does not currently support rotation.

ID properties: id, label

Use "id" and "label" to specify the properties in "features" that contain the regions' IDs and the regions' labels.

```
id: "STATE",
label: "NAME",
```

Canvas properties: area, legend

Use "area" to provide the size in pixels (width, height) of the canvas where your projection will be drawn. Use "legend" to specify the location in pixels (width, height) of the map legend in the canvas.

```
area: [500, 500],
legend: [280, 345],
```

These values will not determine the final size of your map (maps will scale to the available space): they are only important for determining the "scale" and "center" values.

Map setup

Maps must be declared in visual.setup.js. To include a new map, edit visual.setup.js and create a new property with the name of the map inside VisualJS.setup.map. This name must match the name in VisualJS.map (in the Javascript map file) and will be used in the by property. The value of this property must be an object with two properties: the address of the map ("js") and an existence function ("exists").

Once your map has been added to visual.setup.js, use the map maker to fine-tune it.

Public functions

VisualJS.load

This is the main function. It loads the data and, if the property **show** is *true* (default), draws the chart using VisualJS.chart. It only accepts one argument: a visual object or an array of visual objects.

```
VisualJS.load( {...} ); //Same as: visual( {...} );
```

This function is used in webpage mode.

VisualJS.chart

This function does the actual drawing of the chart. It does not accept any argument.

```
VisualJS.chart( );
```

It will usually be invoked inside a callback function.

Examples: adv-01.html, adv-03.html.

VisualJS.iframe

In simple mode, this function is used to embed visualizations. It accepts two arguments: a visual object and a string (a CSS file address or CSS rules). If the visual object contains a callback property, it will be ignored.

```
VisualJS.iframe( {...} , "http://mydomain/path/iframe.css" );
```

Example: simple.html.

VisualJS.compare

This function creates a comparison visualization (two charts side by side). It accepts one argument: an object with the following properties:

title

String. Text of title.

footer

String. Text of footer.

load

Array of two visual objects (required).

css

String or array of two strings. The strings can be CSS file addresses or CSS rules. When two strings are provided, the first style is used in the left chart and the second one is used in the right chart.

Example: adv-04.html.

Dependencies

Visual uses the following libraries internally:

- LazyLoad
- D3
- jQuery, required by Flot
- Flot (stack, categories), Flot orderBars, Flot Pyramid
- ExplorerCanvas

These libraries are only loaded when needed.

For convenience, they are included in the lib folder but you can use any location (for example, a CDN) in visual.setup.js.

Known limitations

D3 requires a modern browser (versions of Internet Explorer prior to 9 are not supported). **Visual** uses D3 only for choropleth maps (chart type: "cmap").

The non-stacked time series chart supports a maximum of three series. This is not a technical limitation but a visual one.

How to contribute

You are welcome to contribute to this project! Areas where your participation can be very useful are, for example:

- Support for new **chart types**
- **Maps** of your territory

To contribute, fork this repository, push changes to your personal fork and send a pull request.