

Dynamic Data Display for JavaScript

Version 1.0

Contents

Introduction	2
How to start?.....	2
Basic Definitions	3
Plot, Figure and Chart.....	6
Palette	8
Legend	10
Navigation	11
Bing Maps.....	12
Axes	13
GridLines Plot	14
Polyline Plot.....	14
Marker Plot	15
Heatmap Plot	16
DOM Markers Plot.....	17
D3 in Windows Store applications	18

Introduction

Dynamic Data Display for JavaScript (or D^3_{JS} for short) is a set of extensible controls for scientific visualization. Visualization is a graphical representation of specified input data; picture reflecting some aspects of input data. In most cases data comes as a set of points with additional optional information about some of the points. Data are considered dynamic, and visualization reflects the changes.

Specificity of scientific visualization is that it must support a large amount of dynamically changing data. Interactivity and navigation are highly important for exploratory purposes.

Dynamic Data Display has cross-browser compatibility. Supported browsers are: Microsoft Internet Explorer 9 (with restrictions), Microsoft Internet Explorer 10, Google Chrome 23.0, Mozilla Firefox 16.0, Safari 5.1 and higher. Some specific features used by D^3_{JS} and their support are listed in the table below.

Feature	Required by	Supported browsers
HTML5 Canvas and CSS3 properties (Transforms, HSLA and RGBA colors, Border-radius, Box-shadow)	All D^3_{JS} elements	IE 9, IE 10, Mozilla Firefox from 16.0, Google Chrome from 23.0, Safari from 5.1
Web Workers	Heatmap	IE 10, Mozilla Firefox from 3.5, Google Chrome, Safari from 4.0

If the heatmap plot is used in the Internet Explorer 9, it is replaced with the message that the plot is not supported by this browser. All other features are still enabled.

Dynamic Data Display can be used in Windows Store applications (see more detail below) and supports multitouch on Surface and iPad devices.

Note. There are versions of Dynamic Data Display for WPF¹ and Silverlight² as well.

How to start?

To start using Dynamic Data Display in your web page it is necessary to follow the steps:

1. Get the release package *d3-1.0.1.zip*.

¹ <http://dynamicdatadisplay.codeplex.com/>

² <http://research.microsoft.com/en-us/um/cambridge/groups/science/tools/d3/dynamicdatadisplay.htm>

2. Copy files *d3.min-1.0.1.js* and external dependencies *jQuery.min-1.8.0.js*, *rx.js*, *rx.jQuery.js* to the */script* folder of your web site.
3. Heatmap plot requires copying of additional files *d3transforms-1.0.1.js* and *d3heatmapworkers-1.0.1.js* to the */script* folder.
4. Copy file *d3.css* to the */css* folder of your web site.
5. Add references to the script files on your web page:

```
<link rel="stylesheet" type="text/css" href="../css/d3.css" />
<script src="../script/jquery-1.8.0.min.js"> </script>
<script src="../script/rx.js"> </script>
<script src="../script/rx.jQuery.js"> </script>
<script src="../script/d3-1.0.1.min.js"> </script>
```

6. Add a `<div>` element to the page which should be turned to a D3 plot; add the attribute `data-d3-plot` to the `<div>` (see more details on the attribute in the section Plot, Figure and Chart) and set its width and height as follows:

```
<div id="myplot" data-d3-plot="chart" style="width: 800px; height: 600px;">
```

7. Initialize the plot in JavaScript code by calling the `D3.asPlot()` function with the `<div>` passed as a parameter (either its id or DOM element):

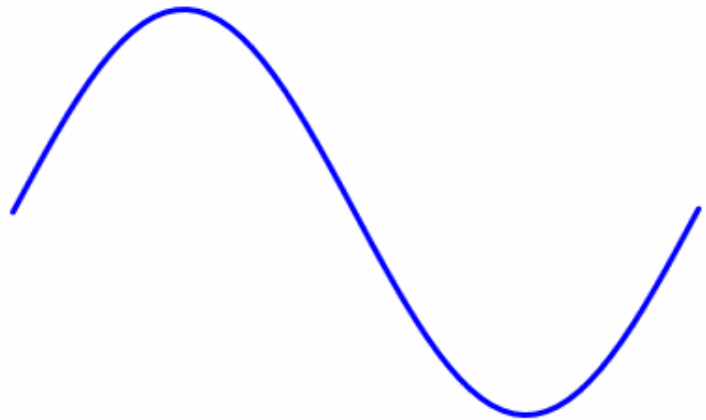
```
D3.asPlot("myplot");
```

See also samples (*D3Samples.html*) in the release package file.

Basic Definitions

A *plot* is an algorithm that maps data elements onto a set of graphic elements such as lines, markers or raster images using HTML5 `<canvas>`. Location, size, color and other visual properties of graphic elements could be bound to the data or have some fixed values. For example, line graph plot transforms series of points (x_i, y_i) into a polyline with vertices coordinates equal to (x_i, y_i) . Source data and resulting plot are shown below.

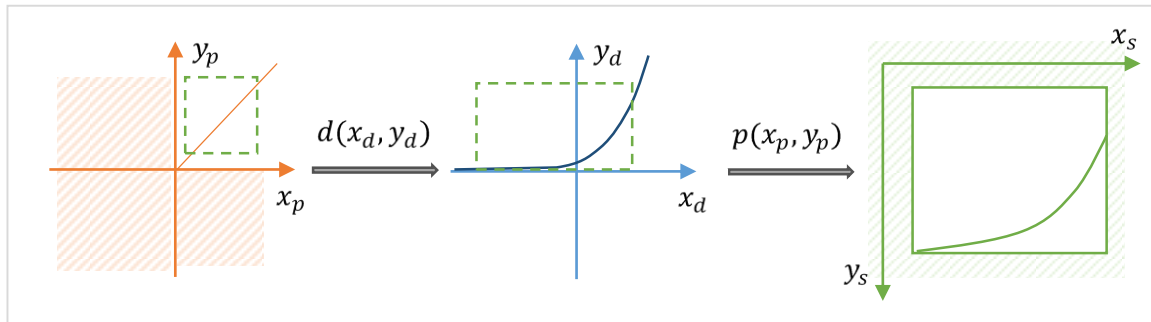
X	Y
0	0
0.314159265	0.309016994
0.628318531	0.587785252
0.942477796	0.809016994
1.256637061	0.951056516
1.570796327	1
...	...
5.969026042	-0.309016994
6.283185307	0



Each plot has *coordinate transform*, a function that converts data values in plot coordinates to screen values in pixels. Coordinate transforms in D^3_{JS} are orthogonal, i.e. x and y coordinates are transformed independently of each other.

Before coordinate transform is applied sometimes it is necessary to convert data coordinates into plot coordinates. For this purpose, there is data transform that computes coordinates of data element on plot plane. Plot plane can be thought of as an abstract sheet of graph paper which size is limited only by representation of floating values. Domain of a given data transform is a possibly infinite segment on which the transformation functions are defined. Data transform can be non-linear and can consider some details about the nature of the data. If data transform is undefined then it equals to identity transformation. D^3_{JS} supports two predefined data transforms: logarithmic and Mercator (available as `D3.logTransform` and `D3.mercatorTransform` in `d3transforms-1.0.0.js`). To create a custom data transform there is a method `D3.DataTransform(dataToPlot, plotToData, domain, type)`. Its constructor gets two functions: `dataToPlot` to convert data values to plot, `plotToData` to convert plot values to data, domain region (can be undefined than it is considered to be infinite) and type, the name of transform.

Each plot transformation may consist of data and coordinate transforms:

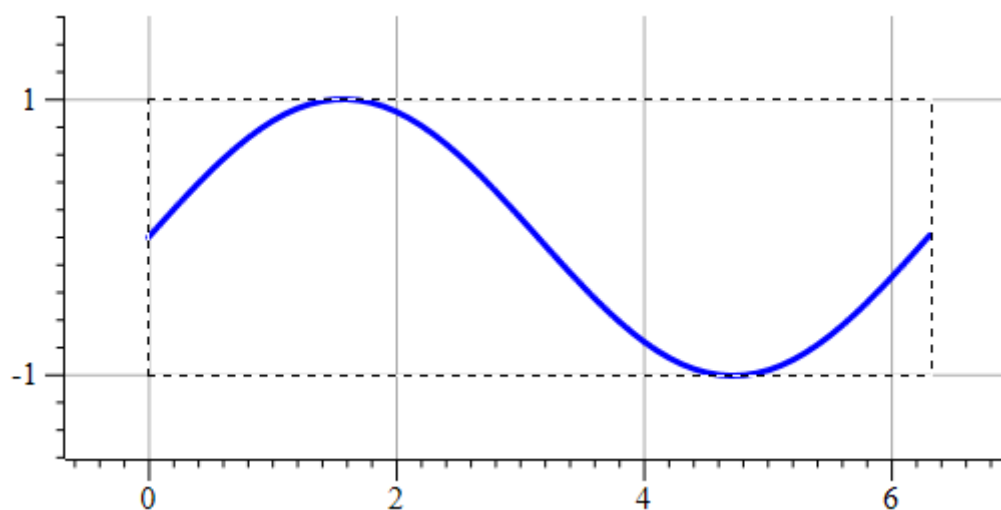


Each plot transform has *aspect ratio*, a coefficient that shows the ratio of screen and plot sizes. If aspect ratio is equal to 1 the transformation is uniform by both axes. It is possible to fix aspect ratio, i.e. to guarantee that aspect ratio remains the same through all plot or screen transformations. The mode in which the invariability of aspect ratio is guaranteed is called *auto fit mode*. Auto fit mode is turned on by default, but any user actions (for example navigation) turns it off. Besides that the user can control auto fit by setting `isAutoFitEnabled` property.

Auto fit rectangle is computed from *plot bounding rectangle* and *screen padding*.

Plot bounding rectangle is minimum rectangle in plot coordinates that contains all points of graphic elements produced by inner plots. Plot bounds computational algorithm is plot specific and if not defined than default rectangle $[0, 1] \times [0, 1]$ is visible.

Screen bounding rectangle of a plot is a minimum rectangle that contains all screen representations of graphic elements of inner plots. It adds screen padding to plot bounding rectangle. Default padding is 20 pixels and it can be changed.



On the figure above a line graph is drawn. Dotted rectangle shows plot bounding rectangle. Padding is the same to all directions and equals to default constant.

In auto fit mode if data is changed dynamically auto fit rectangle changes so that all the data is always visible and aspect ratio remains the same. It is possible to show auto fit rectangle manually by performing fit-to-view (by mouse double-click if navigation is available or by calling `D3.Plot.fitToView()` method).

Plot, Figure and Chart

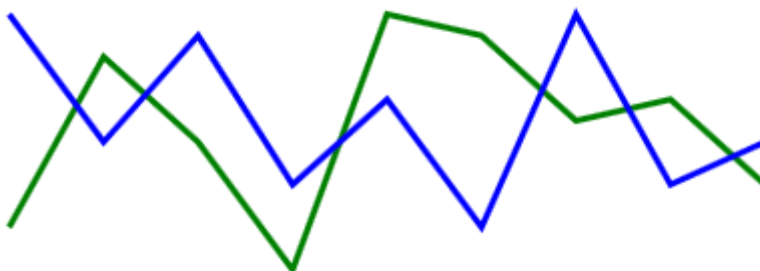
The basic object of D^3_{JS} is a *plot*. Plot is a ready to use control that provides algorithms to layout and render graphic elements. Plot is hosted in a DIV element (only one plot can be hosted in a single DIV element) used as a visual output of the plot. This DIV element determines the screen region for the plot.

Plot that has children plots is called master plot, and its children are called dependent plots. Therefore plots can have tree-structure, but all children consider the root plot as their master. Each master plot layouts its children plots so that they all have the same output screen region.

Plot is hosted in a DIV element which must be marked with the attribute "data-d3-plot". Its value is specific according to particular plot type (e.g. "chart", "plot", "polyline" etc). Then the element is passed to the `D3.asPlot()` method which turns the DIV to the particular plot and initializes it.

The following sample code renders two polylines:

```
<script type="application/javascript">
    $(document).ready(function () {
        var plot = D3.asPlot($("#plot"));
        plot.get("line1").draw({ y:[-0.2, 0.2, 0, -0.3, 0.3, 0.25, 0.05,
                                0.1, -0.1] });
        plot.get("line2").draw({ y:[0.3, 0, 0.25, -0.1, 0.1, -0.2, 0.3,
                                -0.1, 0] });
    }
</script>
...
<div id="plot" data-d3-plot="plot" style="width: 400px; height: 300px">
    <div id="line1" data-d3-plot="polyline"
        data-d3-style="thickness:3; stroke:green"></div>
    <div id="line2" data-d3-plot="polyline"
        data-d3-style="thickness:3; stroke:blue"/></div>
</div>
```



There are predefined factories to use as “data-d3-plot” attribute value. For master plots it can be one of the following: “plot”, “figure” and “chart”. To register new factory, there is a method `D3.register(key, factory)`, where key is a string, factory is a function $jqDiv \times master \rightarrow plot$. If the key already exists, it is replaced, but key “plot” cannot be replaced.

Figure is a master plot which maintains and layouts a number of DIV elements with user-defined placement. Figure overrides the default plot layout algorithm and makes a layout of inner plots, axes and other elements in accordance with the data-d3-placement attribute set in HTML or placement argument in imperative API. Placement is set when initializing and it cannot be changed later. Supported placement values are: left, right, top, bottom, and center. All other elements are positioned by default within the host element of the figure.

Plots are placed in the center of a figure, axes and titles are placed in the side slots. Figure object provides two-pass algorithm that prevents well-known loop occurring on resize of figure with fixed aspect ratio: figure resize forces update of plot-to-screen transform which adjusts labels on the axes. Change of label size may result in change of central part size which again updates plot-to-screen transform which in turn leads to axes label updates and so on.

Besides the get method of plot, figure has special method to find particular axis by its placement: `getAxes(placement)`. If parameter is undefined then the method will return all axes added to the figure.

Chart element is prepackaged figure with horizontal and vertical axes, grid lines, legend and title. All chart elements (axes, plots and grid) uses shared data transform. If chart’s data transform is changed, it is automatically propagated.

Several plots located on one page can be bound. That means binding of their visible rectangles, i.e. if one of the plot is navigated, the bound plot shows the same visible rectangle. Note that screen sizes of the plots may differ, only visible rectangle in plot coordinates is the same. Binding is always two-way, to make it one-way navigation at one of the plots can be disabled. It is possible to bind just vertical or horizontal ranges of the visible rectangles or full plot region.

An example of binding creation is below. The filter parameter is optional and can be one of the following values: “vh”, “v”, “h”. Default value is “vh”.

```
var binding = D3.bindPlots(plot1, plot2, filter);
```

Only master plots can be bound. In case of attempting to bind dependent plot, its master plot is used instead. Method `destroy()` is called to remove the binding.

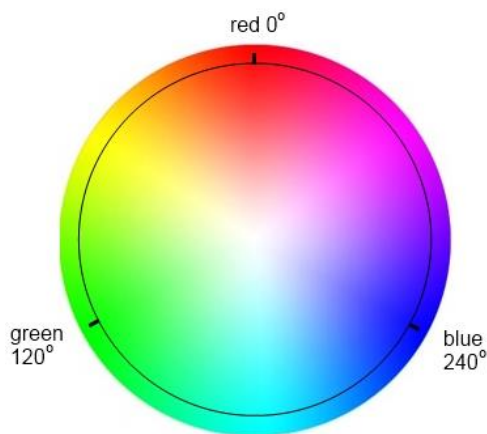
Palette

Color palette maps numeric values into colors.

Palette can be normalized or absolute. A normalized palette defines mapping from segment $[0, 1]$ and always covers the entire data range which is projected to this segment. Absolute palette binds colors to fixed values.

Palette is defined by sequence of points. Each point has assigned color and decimal value from palette data range. Color can be assigned by its name (“red”, “green”, “blue”, etc.) or using hexadecimal syntax (“#rrggbb” or “#aarrggbb” where a is transparency).

Our palette uses HSL (Hue-Saturation-Lightness) model for interpolation between two colors. Hue, saturation and lightness components are interpolated independently. HSL model guarantees that all colors between two brightest (for example blue (#0000ff) and red (#ff0000)) will also have maximum brightness. The figure below shows interpolation circle between red, green and blue colors.



HSL definition is not standardized, so the formulas that we used for RGB to HSL conversion are listed below:

$$M = \max(R, G, B),$$

$$m = \min(R, G, B),$$

$$c = M - m,$$

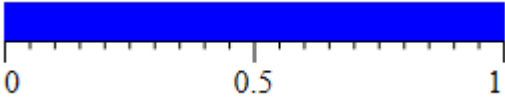
$$H = 60^\circ \cdot \begin{cases} \text{undefined}, c = 0, \\ \frac{G - B}{c} \bmod 6, M = R, \\ \frac{B - R}{c} + 2, M = G, \\ \frac{R - G}{c}, M = B, \end{cases}$$

$$L = \frac{M + m}{2},$$

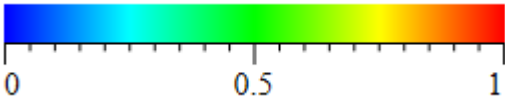
$$S = \begin{cases} 0, c = 0 \\ 1 \\ 1 - |2L - 1| \end{cases},$$

Each palette can be defined as string using special language. At first we consider normalized palettes.

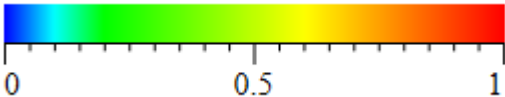
Simple palette maps all values to single color.

Palette	String representation
	"blue"

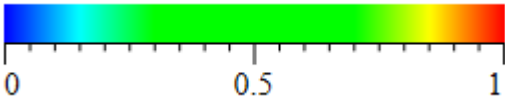
More points can be added separated by comma, then HSL interpolation is used and the result is gradient palette.

Palette	String representation
	"blue, #00ff00, red"

Decimal values can be assigned to palette points explicitly. Symbol '=' maps specific color to specific point by its value. Note that in the following palette green color is shifted to the left.

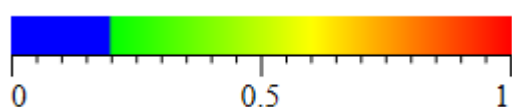
Palette	String representation
	"blue, #00ff00 = 0.2, red"

Regions of solid colors can be defined by using both right and left value assignments.

Palette	String representation
	"blue, 0.3 = #00ff00 = 0.7, red"

It is possible to create palette point that has one color on the left side and another on the right.

Palette	String representation
---------	-----------------------

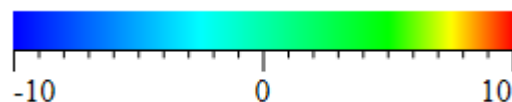


“blue = 0.2 = #00ff00, red”

Absolute palettes are constructed the same way, but its minimum and maximum values are specified at the left and right of border points. All values of the middle points are also specified as absolute.

Palette

String representation



“-10 = blue, #00ff00 = 5, red = 10”

Palette can be created from string using method `parse(definition)`. Method `create(colors)` creates new normalized palette from set of colors. Methods `absolute(min, max)` and `normalize()` can be used to change palette mode (colors remain the same). Method `banded(number|array)` makes the palette discrete with solid colors. If the argument is integer value then palette is divided into the given number of segments. If the argument is an array that contains values within the range of palette, than they represent boundaries of solid segments. Once palette was created methods `getRgba(value)` and `getHsla(value)` can be used to get color for specific decimal value of data range.

Legend

Legend shows additional information about a plot, for example name, color or size information.

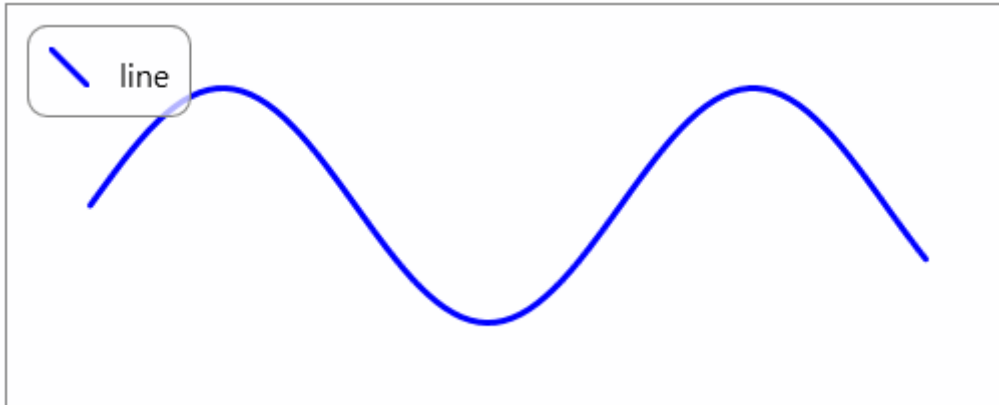
Two main controls used in legend are `colorPaletteViewer` and `sizePaletteViewer`. `ColorPaletteViewer` is used to show palette and corresponding numeric axis. Note that if palette is normalized then axis range can represent full range of data set in “`dataRange`” property explicitly. `SizePaletteViewer` is similar and shows information about size range. Legend is dynamic: it changes with data changings to show only actual information.

Legend of a specific plot can be shown in any DIV element (it can be placed either inside or outside of the plot) by setting attribute “`data-d3-legend`”. Value of this attribute should be the id of a DIV element for legend. Master plot merges all legends of its dependent plots into one, but only chart has predefined legend. The content of legend DIV can be changed by overwriting `getLegend()` method of plot. This method should return jQuery DIV element containing legend content.

Legend can be made invisible by changing “`isVisible`” property (chart has specific style field “`isLegendVisible`” for this purpose). To remove legend from DIV element call `remove()` method.

An example of creating legend for figure is shown below:

```
<div data-d3-plot="figure" data-d3-legend="legend"
      style="width: 500px; height: 200px">
  <div id="line" data-d3-plot="polyline" data-d3-style="stroke: blue"></div>
  <div id="legend" style="float: left"></div>
</div>
```



Navigation

Navigation is an object that provides control of visible region of a plot. All user actions are translated into a sequence of gestures. Gesture is an object describing one of supported navigation actions:

- pan gesture changes the visible region keeping the zoom level;
- zoom gesture changes the zoom level;
- zoomTo gesture provides zooming to specific point on a plot plane;
- pin gesture stops current animated navigation process.

Navigation subscribes to a source of incoming gestures, and computes target visible rectangle in plot coordinates for each gesture in accordance with the gesture type. Each master plot has a single non-replaceable navigation object, dependent plots use its master's navigation. To subscribe to gesture source use property `gestureSource`, to unsubscribe assign it to null or undefined value.

Navigation starts new animation each time the new gesture is received. Animation object gets start and end visible regions and provides a sequence of rectangles on plot plane. Navigation subscribes to this created source and updates visible plot region as new information appears. Each animation can have only one subscriber, so any new navigation uses new animation object.

Navigation can be performed both by automatic actions (mouse or touch gestures if it is subscribed to an appropriate source) and manually by calling method `setVisibleRect()` of navigation object. Any new started animation stops the previous one. To stop animation manually there is method `stop()`.

D³_{JS} supports three types of navigation: by mouse or by touch interface on iPad or Windows 8. There is standard animation, `panZoomAnimation`, which is used in chart objects. It works as a spring and the formula is given below:

$$x = x_0 + (x_1 - x_0) * k * dt,$$

where x_0, x_1 – edges of segment, x – point inside the segment, k – coefficient of elasticity and dt – time interval.

The following code shows how to create navigation for a figure. First gesture source stream is created from host element of figure object, then this stream is assigned to “gestureSource” property of figure navigation:

```
var figure = D3.asPlot($("#figure"));
var gestureSource = D3.Gestures.getGesturesStream(figure.host);
figure.navigation.gestureSource = gestureSource;
```

Bing Maps

Bing maps plot is a special type of a plot representing a map to be used as a background layer for other plots. It is based on Bing Maps AJAX Control version 7. To enable the Bing Maps plot on a page:

- Include a `<script>` with a link to the control (see the sample below).
- Declare a plot using the “data-d3-plot” attribute equal to “bingMaps”.
- Set the attribute “data-d3-mapKey” with correct value (keys can be received at <http://www.bingmapsportal.com/>).

```
<script type="text/javascript"
src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=7.0"></script>
...
<!--Adding a map plot-->
<div id="map" data-d3-plot="bingMaps" data-d3-mapKey="..."></div>
```

Bing maps plot always uses the Mercator transform as its data transform. Other plots visualizing data bound to the geographical coordinates should also use the Mercator transform (`D3.mercatorTransform`).

Bing map sets restrictions on a visible region of its master plot: on x axis the map is looped and always tries to show the region inside segment $[-360, 360]$, on y axis it is limited in the segment $[-\text{maxLat}, \text{maxLat}]$ where `maxLat` is a constant taken from the Bing Maps documentation and is equal to 85.05112872.

Bing map plot allows the user to choose different maps. Supported maps are: Bing Maps, ESRI and Open Street Maps with different map types. To change type of map use plot's method `setMap(type)`.

Navigation of bing maps plot uses specific animation (`BingMapAnimation` object) that considers restrictions of a map listed above. We suggest not to change animation type unless it is necessary. Note that while zooming plot rectangle can be larger than selected visible region because of zooming levels in bing maps control.

Bing maps plot allows adding custom layers with plots (markers, heatmap, etc.) on the map. To increase performance while animation is in progress these plot primitives are not rendered and are replaced with figure shapes. When animation is completed plots are fully rendered.

Axes

Axes controls show how numeric values of different nature correspond to screen coordinates. Each shown value is presented with axis tick. There are text labels drawn for each major tick. Shorter minor ticks divide space between major into equal segments.

An axis is defined by its range and orientation mode. Range determines the range of values to show on axis. Orientation defines placement of ticks and labels, it may be "bottom", "top", "left" or "right".

"DataTransform" property is used to specify additional transformation between values of specific range and their positions on screen: values of range are considered to be plot coordinates and values to show on labels are considered to be data coordinates. By default data transform is identity.

Axis is hosted in DIV element marked with the attribute "data-d3-axis" to specify the type of axis (it can be "numeric", "logarithmic" or "labelled"). Axis orientation mode is specified by attribute "data-d3-placement". New range can be passed through `update(newRange)` method. Methods `destroy()` or `remove()` can be used to remove axis from host element or from its parent plot.

The following code shows how to define axis with mode = "bottom" and range = [0, 100].



```
<script type="application/javascript">
    $(document).ready(function () {
        var axis = new D3.NumericAxis($("#axis"));
        axis.update({ min: 0, max: 100 });
    }
</script>
...
<div id="axis" data-d3-placement="bottom" style="width: 450px"></div>
```

Prototype of each axis is ticksRenderer object that performs main algorithm of calculating and rendering ticks and labels. TicksRenderer refers to tickSource object to calculate ticks for particular range.

There are three types of axis:

1. Numeric Axis is a standard decimal axis. It uses NumericTickSource to calculate ticks and labels. The formula for tick value t_i is shown below. Parameter p is chosen from (1, 2, 5) and q is chosen from integer numbers to archive optimal major ticks placement (not too dense, not too sparse).

$$t_i = i \cdot p \cdot 10^q$$

2. Logarithmic axis has predefined fixed data transform: base ten logarithm.
3. Labelled axis shows fixed number of predefined ticks and labels given with initialization as arrays.

It can be in one of two modes:

- if number of ticks is equal to number of labels (or ticks array is not defined) than each label has a corresponding tick;
- if number of ticks is one more than number of labels than each label corresponds to interval and is placed between two neighboring ticks.

Ticks are values in plot coordinates, if not defined they are considered to be sequential indexes.

Labels are string values that can be passed to innerHTML of label host. The labels can be rotated to become vertical, then parameter 'rotate' in constructor should be set to 'true'.

GridLines Plot

GridLines plot renders coordinate grid. This type of the plot is named "grid". It can be bound to specific axes with attributes "data-d3-xaxis" and "data-d3-yaxis". Style of the plot can be changed with attribute "data-d3-style", it support changing of stroke color and thickness.

Polyline Plot

A polyline is a plot that renders a line based on a set of points. To define a polyline plot in HTML assign "data-d3-plot" attribute to "polyline", in code behind use `D3.Plot.polyline(name, data)`, or `D3.PolylinePlot.draw(data)`.

Polyline plot supports array-based data sources. In HTML it is possible to create a table, which columns are data series with same header names as in the parameter of the `plot()` method. The "data-d3-datasource" attribute specifies the function that parses the table. Default method is "readCsv". Main data series are "x" and "y", they represents points coordinates. "X" data series can be missing then it is

supposed to be sequential integers. If any value in an array is NaN then there will be a break in a line and a single point at the meaning value.

```
<div data-d3-plot="polyline" data-d3-style="stroke: blue; thickness: 2px">  
  x y  
  0.1 20  
  0.2 10  
</div>
```

In JavaScript code each data series is passed as an array.

```
polyline.draw({  
  x: [0, 1, 2, 3, 4, 5],  
  y: [-1.8, 1.3, 0, -2.5, 4.1, 3.2],  
});
```

Appearance of a polyline can be customized by setting “data-d3-style” attribute. Possible variables are “stroke” (can be either name or hexadecimal value), “thickness” (in pixels, for example “3px”), “lineCap” and “lineJoin”. All this values can also be passed to polyline in plot method and their changings causes plot rerendering automatically.

Marker Plot

Markers plot renders values of multiples variables as a collection of markers of different shape, size and color. Some of this variables define the position of markers, others represents visual properties. To define a marker plot in HTML assign “data-d3-plot” attribute to “markers”, in code behind use `D3.Plot.markers(name, data)`, or `D3.MarkerPlot.draw(data)`.

Data series for marker plot can be defined the same way as for polyline plot. If either “x” or “y”, or any required data series has value NaN, this particular marker is not rendered.

With attribute “data-d3-style” (or in method “plot”) it is possible to set shape, border color, size and color of markers. Shape can be one of predefined (“box”, “circle”, “cross”, “diamond” or “triangle”) custom. Custom marker shape should be an object with required method “draw” (to render each marker) and additional methods “getBoundingBox” (to find bounding rectangle for each marker) and “getLegendItem” (to render sample marker in a legend). Color can be single value or array. In case of array of decimal values if “colorPalette” property is defined then values of color array are considered to be values of palette. If “color” is an array of colors then they are taken sequentially without palette usage. Size is set the same way using “size” and “sizePalette” properties.

Heatmap Plot

Heatmap plot renders values defined on a rectangular grid using color palette. To define a heatmap plot in HTML assign “data-d3-plot” attribute to “heatmap”, in JavaScript code use `D3.Plot.heatmap(name, data)`, or `D3.HeatmapPlot.draw(data)`.

Data for heatmap is a set of data series (x, y, f, p) , where “x” and “y” data series represent arrays of corresponding coordinates of grid points, “f” data series is two-dimensional array of values in these grid points and “p” is color palette. If “x” or “y” series are missing they are considered to be sequential integers. Note that both “x” and “y” arrays should have at least 2 elements. Values of data array can be NaN then the point will be skipped.

If number of values in arrays of coordinates are equal to corresponding dimensions of data array then heatmap is rendered in a gradient mode, when each cell of the grid has 4 colors defined in the corners of the cell and color within the cell is interpolated. If number of coordinate values are greater by one than dimensions of data then heatmap is rendered in a matrix, or bitmap, mode, when a cell (i, j) is built so that each point (x_i, y_j) is in the middle of that cell which color is defined by f_{ij} .

Heatmap plot supports declarative definition. The “data-d3-style” contains attribute “palette” for a string representation of the color palette to be used in the heatmap. Initial data (passed as a table where each column is a single data series) can be read using data initializer (either default, which is `D3.readCsv2d`, or provided through “data-d3-datasource” attribute).

It is possible to modify heatmap transparency using not only palette but “opacity” property, or “data-d3-style” attribute. Domain of this parameter is $[0, 1]$, where 0 means transparent, 1 means opaque heatmap.

Heatmap plot uses web workers to render the data. The rendering algorithm is as follows: first intersection of the visible rectangle and plot’s grid is built. If it is not empty the placeholder is rendered on a screen area where the actual heatmap will appear. Then heatmap is ready to be rendered: the worker takes a task and pixel by pixel fills the image data. When the worker completed the task, a new pending task can be passed to it and next frame is requested. Heatmap uses `D3.SharedRenderWorker` to enqueue tasks into a single queue so that only one web worker is used to handle tasks one by one.

According to use of web workers heatmap plot can use only data transforms defined in the `d3transforms.js` file, and the type property should have unique name. This name is passed to the background worker to identify the transform. To enable custom data transform, add it to the `d3transforms.js` file and give it a name.

DOM Markers Plot

DOM markers plot is a plot that contains one or more div elements, defined declaratively in HTML or added imperatively. For each DOM element, coordinates (x, y) of the left-top corner are set in the parent plot's data space. Width and height of the element are computed in different ways depending on a scale mode. There are three modes of scaling: element (default), content and none.

If the scale mode is "element" then width and height of the element in the plot data space are provided by a user (through HTML attribute "data-d3-size" or through the Java Script API). This scale mode changes the element's size and position on the screen while zooming or panning but layout of the elements content is arrange by the browser itself. For example, this means that font size doesn't depend on the zoom level.

Scale mode "content" also requires width and height in the data space to be provided by a user, but in this case entire element with its content is scaled to fit into the new screen size. In particular, this means that font size is also scaled.

Scale mode "none" doesn't use width and height in the data space. Instead, the elements size is specified by a user via CSS style attributes width and height, though position of its left-top corner changes as the coordinate transform changes. This mode is especially useful to create pushpins bound to a certain point in the data space.

To define new DOM markers plot in HTML use "dom" value of "data-d3-plot" attribute and specify position ("data-d3-position") and size ("data-d3-size" or width and height) of element on plot plane.

```
<div id="dom" data-d3-plot="dom" style="width: 500px; height: 500px">
  <div data-d3-position="0.35 0.15" data-d3-size="0.15 0.1"
    data-d3-scale="content">
  </div>
  <div data-d3-position="0 0.4" data-d3-size="0.3 0.2"
    data-d3-scale="element">
  </div>
  <div data-d3-position="0.3 0.4" style="width: 150px; height: 150px"
    data-d3-scale="none">
  </div>
</div>
```

For imperative adding or removing of the DOM element, use methods `Plot.addDOM(element, x, y)` and `Plot.removeDOM(element)`. To change the position there is method `Plot.setPosition(element, x, y, width, height)`.

D3 in Windows Store applications

Dynamic Data Display library can be used in Windows Store applications in the same way as in a Web application, except Bing Maps plot:

- Add references to the D3-related scripts, which should be copied to the */script* folder (other folder breaks the heatmap plot), and d3.css.

Note. It is possible to modify d3.css to make the plot's background black and foreground white, if required for Windows Store application.

- Add a `<div>` element marked with the `data-d3-plot` attribute.
- Initialize the plot using `D3.asPlot()` function in the JavaScript code. In most cases this can be done in the `app.onactivated` handler:

```
app.onactivated = function (args) {  
    if (args.detail.kind === activation.ActivationKind.launch) {  
        ...  
        args.setPromise(WinJS.UI.processAll()  
            .done(function () {  
                D3.asPlot("chart");  
            }));  
    }  
};
```

- Bing Maps plot requires adding a project reference to the Windows Extension module “Bing Maps for JavaScript”. Also add the references to the page:

```
<!-- Bing Maps references -->  
<script type="text/javascript" src="ms-appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>  
<script type="text/javascript" src="ms-appx:///Bing.Maps.JavaScript/js/veapimodules.js"></script>
```

It is important to load the Bing Maps module before initializing the D3 plot:

```
app.onactivated = function (args) {  
    if (args.detail.kind === activation.ActivationKind.launch) {  
        ...  
        args.setPromise(WinJS.UI.processAll()
```

```
.done(function () {  
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: initMap, homeRegion: 'US', culture: 'en-us' });  
    });  
}  
};  
function initMap() {  
    D3.asPlot("chart").yDataTransform = D3.mercatorTransform;  
}
```