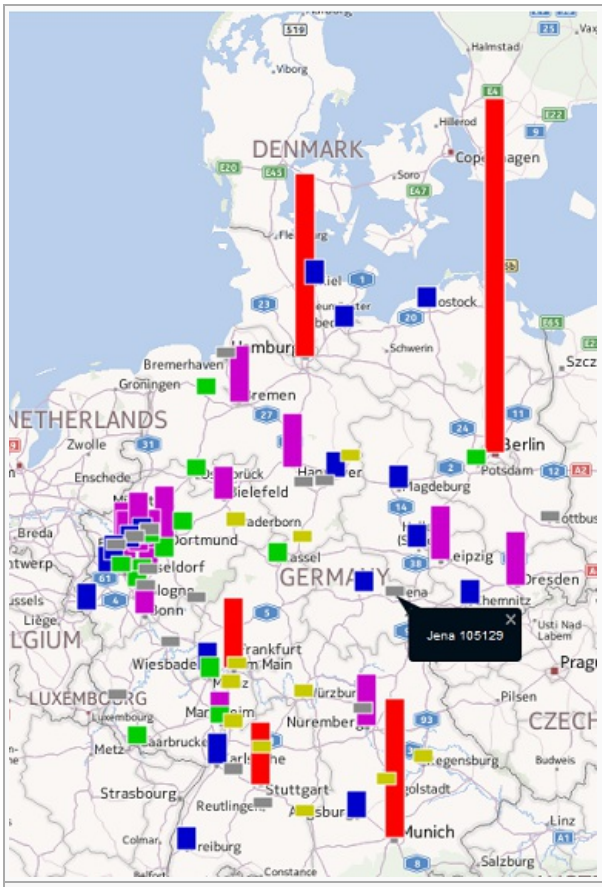
HERE Maps API - How to display a bar graph over the map

This article explains how to display graphical bar charts over specified locations on a map.

Introduction

One typical usage case for an online mapping application is to give geographic context for the display of data. Typically the data will consist of a geographic element (longitude/latitude) and some associated non-geographic data such as population data or sales figures. An excellent way to visualise this non-geographic data is through the use of a bar chart, as shown below:



Population distribution of all towns in Germany with more than 100,000 inhabitants.

Working example available at:

bar-chart-markers.html

The simplest way of creating bars for such a geographic bar chart is to use SVG graphics. The HERE Maps API is able to render SVG graphics as markers and these can be added to the map in the usual manner. This code example creates a framework to calculating and displaying bars of an appropriate length based on a minimal number of input parameters.

How to display Population Data on Screen

Loading in the data

The population data is held in a separate file, and loaded in by adding a `<script>` reference to the `<header>` of the file.

```
<script type="text/javascript" charset="UTF-8" src="./german_pop_lat_long.js"></script>
```

An array of `DataPoints` for the population data (`popData`) can be read in using the typical JSON format:

```
var popData =
[{""latitude":52.51607,"longitude":13.37698,"value":3460725,"name":"Berlin"},
```


(C) Copyright Nokia 2014. All rights reserved.

```
{"latitude":53.55334,"longitude":9.99245,"value":1786448,"name":"Hamburg"},
{"latitude":50.11208,"longitude":8.68341,"value":679664,"name":"Frankfurt am Main"}]
```

Note: Because the `popData` follows the standard nokia.maps.heatmap.Overlay.DataPoint format (i.e. `latitude`, `longitude`, `value`) , it could also be displayed as a heatmap using the following:

```
var heatmapProvider = new nokia.maps.heatmap.Overlay({
  max: 20, opacity: 0.6, type: "value", coarseness: 10
 });
heatmapProvider.addData(popData);
 display.overlays.add(heatmapProvider);
```

The separation of the data model from the display of the data (the view) is good architectural practice. It also means that different data sets can be displayed through reusing the same display code, by merely substituting an alternative data file.

The display of a single bar for each location is achieved through iterating through the data and running it into a `barChart` function described below. Each location is then added as a marker as shown.

```
barChart = new barChart(COLORS, THRESHOLDS, TICK_VALUE);
var i = popData.length;
while(i--) {
  var marker = barChart.createBar(popData[i]);
  display.objects.add(marker);
}
```

The bar chart function

A simple SVG rectangle can be displayed with the following syntax:

```
<svg width="16" height="32" xmlns="http://www.w3.org/2000/svg">
     <rect stroke="#FFF" fill="#FF0000" width="16" height="32"  />
</svg>
```

The Custom SVG graphics example from the Developer's Playground shows how to substitute placeholder in the SVG to alter the graphic displayed, for a bar chart, the following pair of values need to be dynamically updated:

- The color of the bar
- The height of the bar

The color is defined by comparing the data value with an array of "threshold" values each of a higher amount, and assigning an associated color to each threshold. The height of the bar is decided by setting a "tick" value i.e. the length for each pixel in height. By requiring these parameters to be passed in on construction, an infinitely wide range of data ranges can be catered for.

```
function barChart(stops, tick) {
  this.stops = stops;  // This map associates threshholds with colors
  this.tick = tick;
   this.barSVG = '<svg width="16" height="__VALUE__"
xmlns="http://www.w3.org/2000/svg">' +
 '<rect stroke="#FFF" fill="__MAINCOLOR__" width="16" height="__VALUE__"  />' +
 '</svg>'

  this.svgParser = new nokia.maps.gfx.SvgParser(),
  // Helper function that allows us to easily set the text and color of our SVG marker.
  this.createBar = function (data) {
```


(C) Copyright Nokia 2014. All rights reserved.

```
    var color;
     for (var i in this.stops) {
     if (data.value > i){
       color = this.stops[i];
      }
    }

   var height = Math.round(data.value/tick);
   var svg = this.barSVG
     .replace(/__VALUE__/g, height)
    .replace(/__MAINCOLOR__/g, color);

   return new nokia.maps.map.Marker( [data.latitude,data.longitude], {
   icon:  new nokia.maps.gfx.GraphicsImage(this.svgParser.parseSvg(svg)),
   anchor: new nokia.maps.util.Point (8, height)});
  };
};
```

Setting up the bar chart is done as follows:

```
var tick = 12000; // 12 thousand inhabitants = 1 pixel height
var stops = {
    "0": "#8A8A8A",       //Grey for 0-120,000 inhabitants.
    "120000": "#CACA00",  //Yellow for 120,001 - 150,000 inhabitants.
    "150000": "#00CC00", //Green for 150,001 - 200,000 inhabitants.
    "200000": "#0000CC",  //Blue for 200,001 - 300,000 inhabitants.
    "300000": "#CC00CC", //Purple for 300,001 - 600,000 inhabitants.
    "600000": "#FF0000"   //Red for over 600,000 inhabitants.
   };

barChart = new barChart(stops, tick);
```

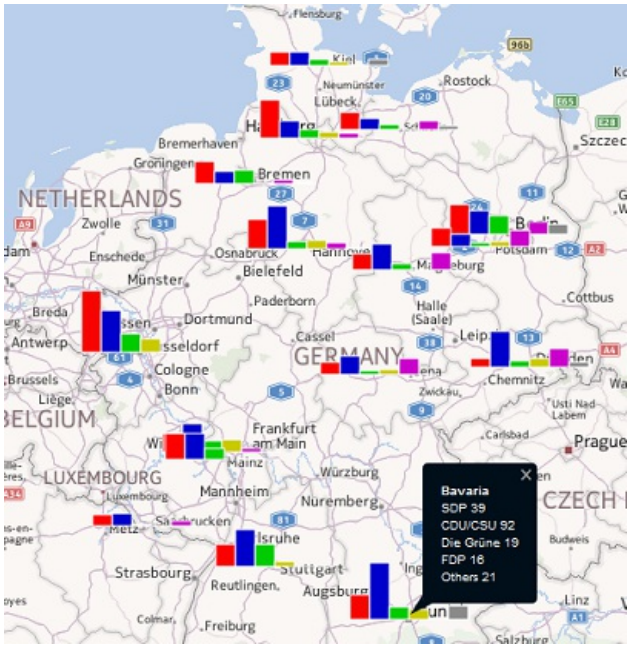
The bars can be added by iterating through the array:

```
var i = popData.length;
while(i--) {
    display.objects.add( barChart.createBar(popData[i]));
}
```

Summary

Through using the functions described above, it is possible to display graphical data on a HERE Map. An extension to cope with multiple bars per chart (shown below) is also included - feel free to adapt the code to your own use.

(C) Copyright Nokia 2014. All rights reserved.



Working example:

bar-chart-election-result.html

(C) Copyright Nokia 2014. All rights reserved.