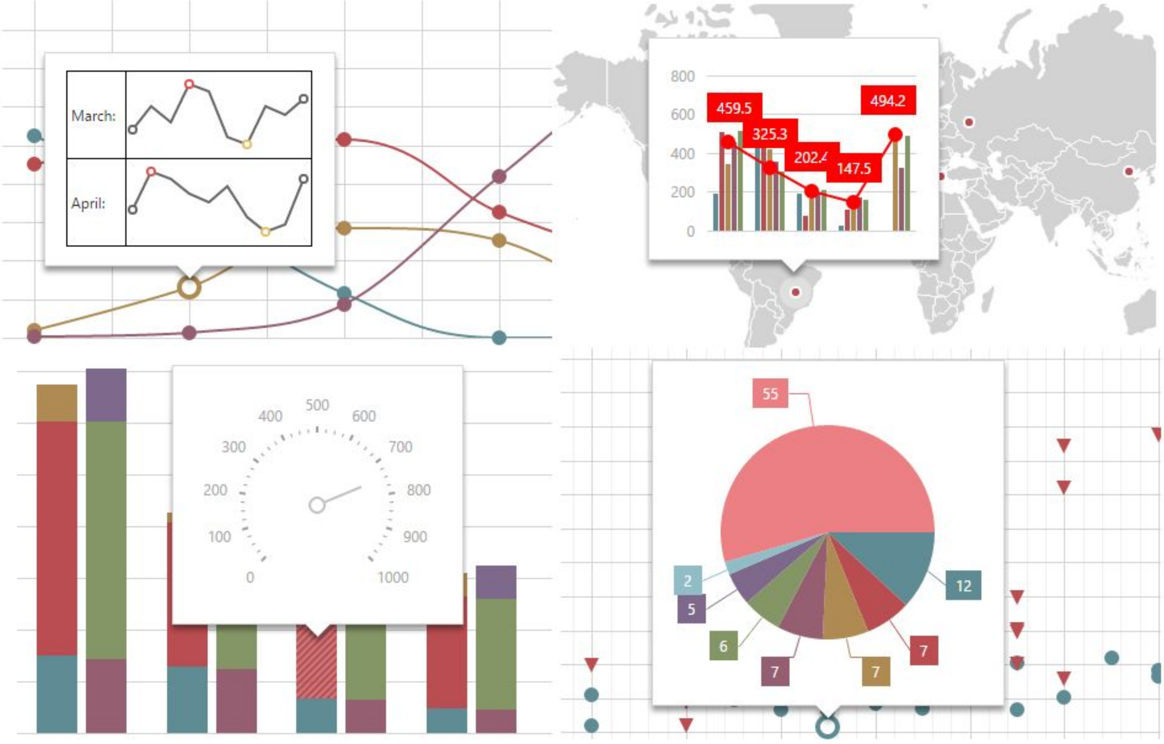
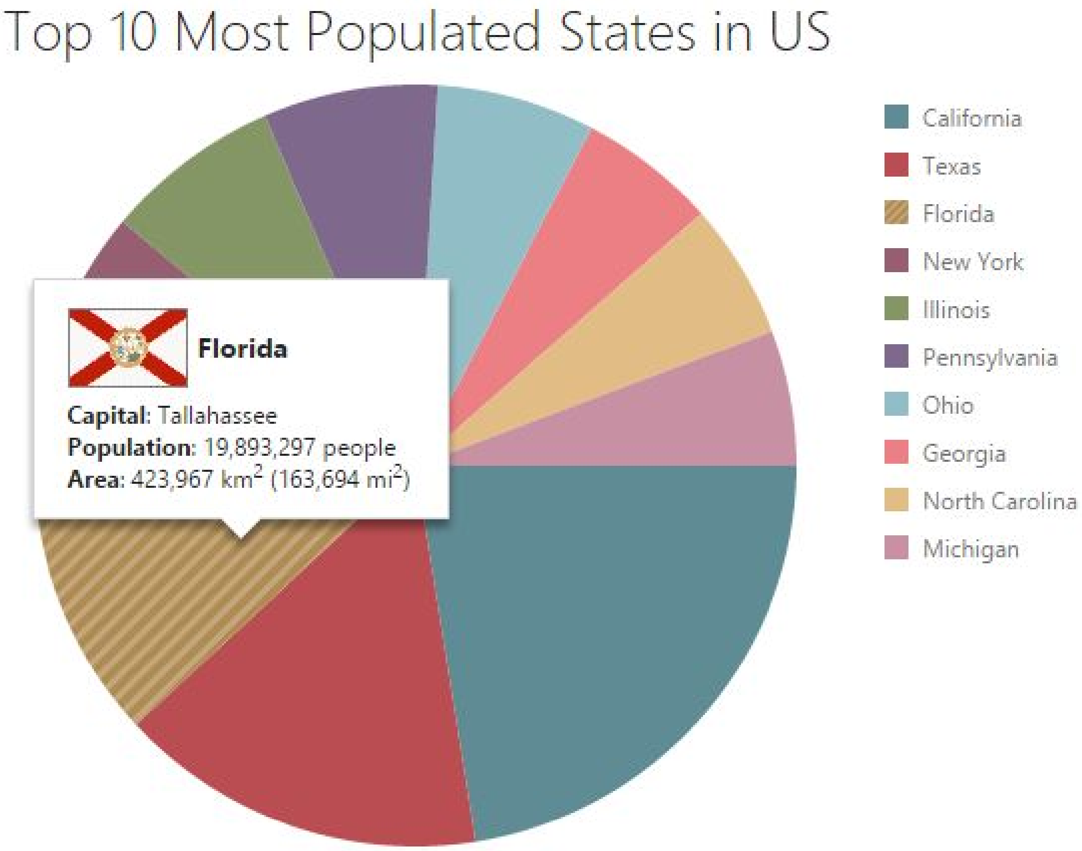
DevExtreme: The Boundless Customization by HTML-tooltip.



Recently we have presented the new version of our products - DevExpress 15.1. One of the new capabilities for data visualization widgets is creation of a tooltip containing HTML-markup. Thus, a user can fully customize a tooltip appearance. It allows to present more information using colorful and bright tooltip.

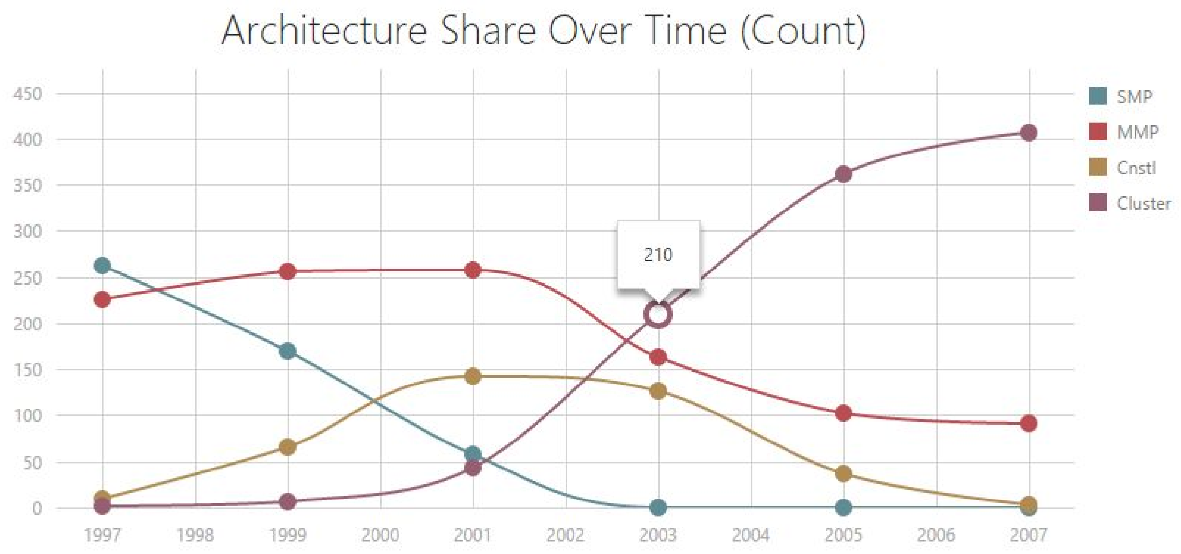


Florida

Capital: Tallahassee
Population: 19,893,297 people
Area: 423,967 km^2 (163,694 mi^2)

Is this capability difficult for using or not? Let's try to find out it in this blog.

Why could this feature be useful? The default view of a tooltip has limited number of ways for customization. In the common case, a tooltip displays main information about tooltip target. It could be extended by additional data got from the target. But, in any case, it has poor text view, which isn't the best way for data presentation.

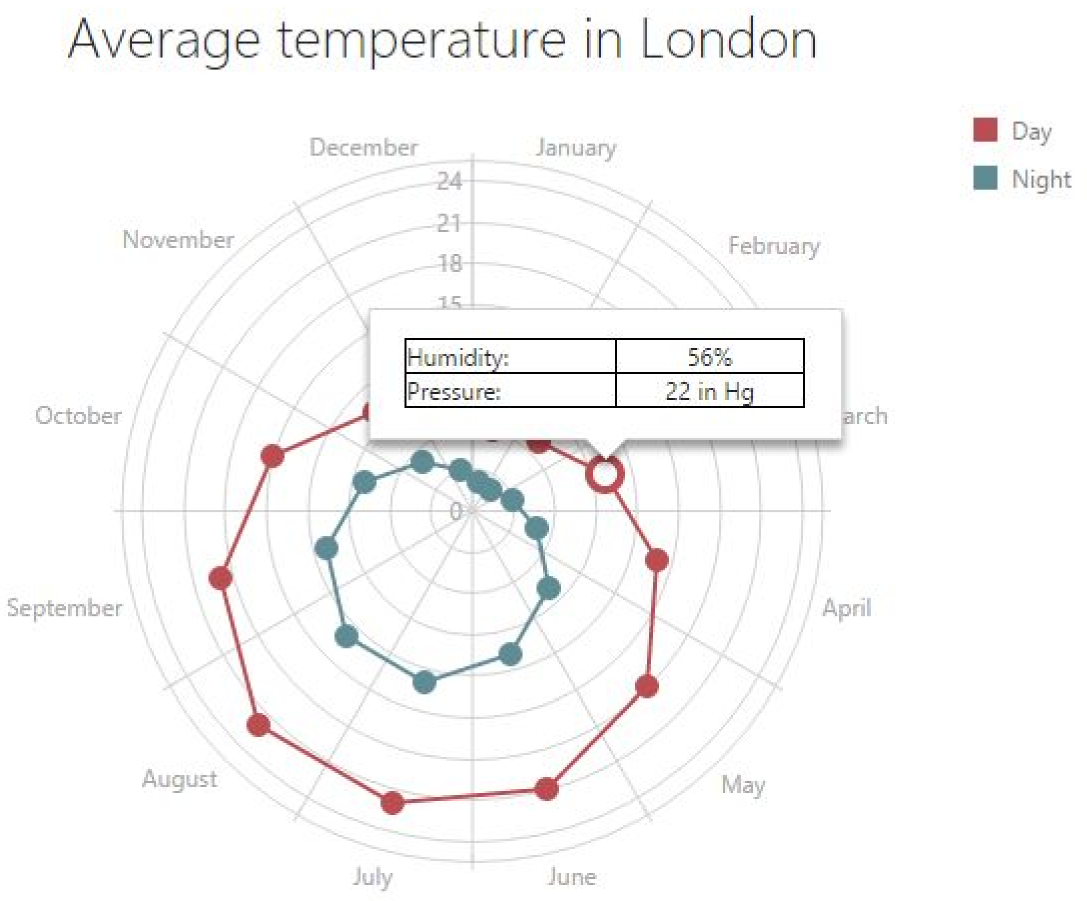


So we decided to extend tooltip customization abilities by HTML-markup. It allows you to use all flexible and powerful tools for HTML content creation.

We have implemented this feature by extending the **customizeTooltip** option by a new field - **html**. To use a tooltip containing a custom HTML-markup, just create a code structure like this:

```
$("#container").dxChart({
//...
      tooltip: {
            enabled: true,
            customizeTooltip: function () {
                  return {
                        html: "html content"
                  }
            },
            //...
      },
//...
});
```

This construction enables you to simply create a tooltip containing an HTML-table, for example, by adding relevant markup to the **html** field.



Humidity:	56%
Pressure:	22 in Hg

But what if you need to add more complex structure than an HTML-table? For example, another visualization widget. Is it more difficult? To answer this question, let's try to create a tooltip containing visualization widgets. Let's create an example in which a tooltip containing several widgets will be shown for the map widget.

In our example the tooltip will display weather information for some cities of the United States. We will get the weather data from the query.yahooapis.com server using an AJAX-query. he tooltip will display weather forecast, atmospheric humidity and pressure, and sunrise and sunset time.

Let's implement this example step by step.

Step 1. Let's create a map of the United States. Begin with creating the dxVectorMap widget with the **mapData** option holding resources for the USA map. The map size will be 1000*700 pixels and bounds will exclude Alaska:

```
html:
<script src="usa.js"></script>
<div class='container' style='width: 1000px; height: 700px;'></div>

script:
var map = $(".container").dxVectorMap({
      mapData: DevExpress.viz.map.sources.usa,
      bounds: [-118, 52, -80, 20]
}).dxVectorMap("instance");
```

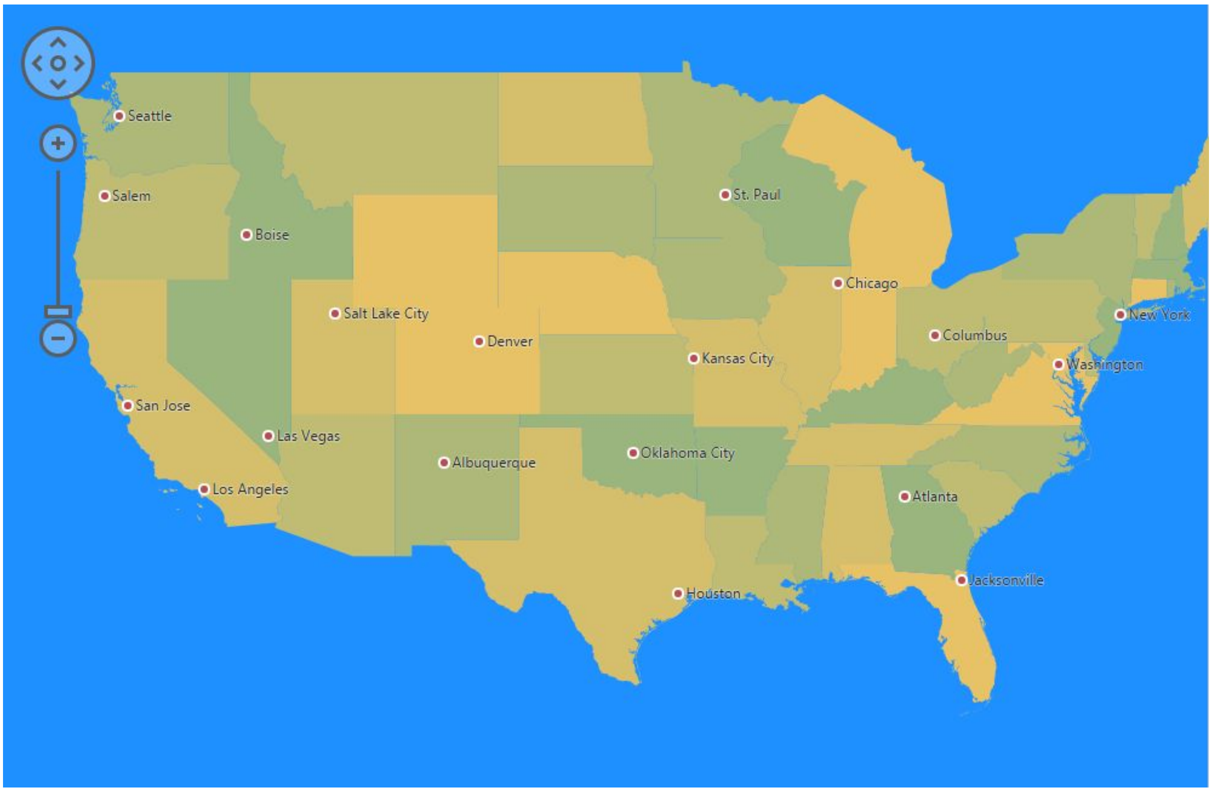
Let's add some colors for our map:

```
//...
var paletteIndex = -1;
var map = $(".container").dxVectorMap({
      //...
      background: {
            color: "#1E90FF"
      },
      markerSettings: {
            label: {
                  font: {
                        color: "#232323"
                  }
            }
      },
      areaSettings: {
            palette: "Soft",
            paletteSize: 5,
            borderColor: "none",
            hoverEnabled: false,
            customize: function () {
                  return { paletteIndex: (paletteIndex = (paletteIndex + 1)
% 5) };
            }
      }
}).dxVectorMap("instance");
```

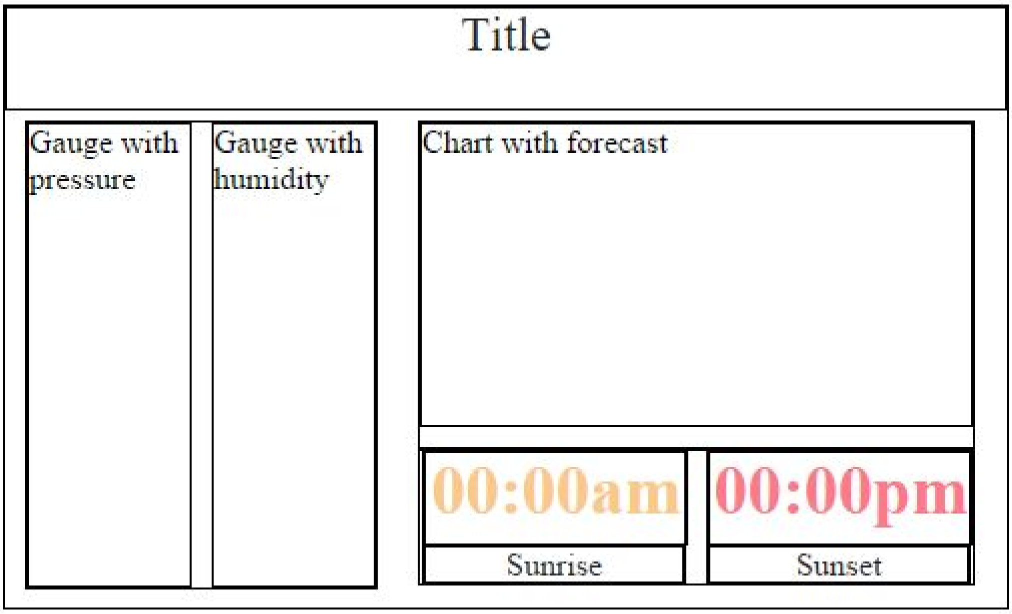
Step 2. Let's take care about map **markers**. I have chosen about twenty cities of the United States and saved their WOEID used to get weather data for them. Using the AJAX-query we can get the city name, coordinates, the weather forecast, atmospheric humidity and pressure, and sunrise and sunset time. Once data is got we can update the map widget by submitting the city markers. While the widget is being updated, it will show the loading indicator.

```
//...
var map = $(".container").dxVectorMap({
      //...
}).dxVectorMap("instance");
map.showLoadingIndicator();

var  cities  =  [2459115,  2514815,  2428344,  2383660,  2357024,  2464592,
2379574,  2487129,  2430632,  2424766,  2352824,  2391279,  2487610,  2366355,
2436704, 2442047, 2488042, 2487384, 2490383];

$.ajax({  url:  "http://query.yahooapis.com/v1/public/yql?q=select  *  from
weather.forecast where woeid in (" + cities.join(", ") + ")&format=json",
dataType: "jsonp" }).done(function (arg) {
      var markers = $.map(arg.query.results.channel, function (ch) {
      return {
            text: ch.location.city,
                  coordinates: [Number(ch.item.long), Number(ch.item.lat)],
            forecast: ch.item.forecast,
            humidity: ch.atmosphere.humidity,
            sunriseTime: ch.astronomy.sunrise,
            sunsetTime: ch.astronomy.sunset
      }
      });
      map.option('markers', markers);
});
```

After these two steps you can see the map of the USA with city markers:



Step 3. Let's customize the tooltip. We should decide what markup the tooltip shoul contain. I decided to use the following tooltip structure:



Container elements for widgets (gauges and chart) and dynamic content (title, sunrise and sunset time) should have a css class for jQuery-access to them. So, our markup is held in the **html** field of the **customizeTooltip** option:

```
css:
.tooltip-container {
      width: 500px;
      height: 300px;
}
.text {
      color: #232323;
      font-weight: 200;
      text-align: center;
}
.city-title {
      height: 50px;
      font-size: 24px;
}
.pressure-gauge {
      float: left;
      width: 80px;
      height: 230px;
}
.humidity-gauge {
      float: right;
      width: 80px;
      height: 230px;
}
.forecast-chart {
      height: 150px;
}
.time-text {
      height: 45px;
      width: 130px;
      font-weight: 600;
      font-size: 34px;
      text-align: center;
}
.sunrise-time {
      color: #FBC987;
}
.sunset-time {
      color: #FD7888;
}
.left-container {
      margin-top: 5px;
      float: left;
}
.right-container {
      float: left;
      margin-top: 5px;
      margin-left: 20px;
}
.time-container {
      float: left;
      width: 130px;
      height: 65px;
}
.left-margin {
      margin-left: 10px;
}
.top-margin {
      margin-top: 10px;
}

script:
//...
var map = $(".container").dxVectorMap({
      //...
      tooltip: {
            enabled: true,
            customizeTooltip: function (arg) {
                  if (arg.type === "marker") {
                  return {
                              html: "<div class='tooltip-container'><div
class='city-title text'></div><div class='left-container left-margin'><div
class='pressure-gauge'></div><div class='humidity-gauge left-margin'
></div></div><div class='right-container'><div
class='forecast-chart'></div><div class='top-margin''><div
class='time-container'><div class='sunrise-time time-text'></div><div
class='text'>Sunrise</div></div><div class='time-container
left-margin'><div class='sunset-time time-text'></div><div
class='text'>Sunset</div></div></div></div></div>"
                           }
                  }
            }
      }
}).dxVectorMap("instance");
//...
```

Step 4. Let's customize the required widgets. There are gauges for atmospheric pressure and humidity and a chart for the weather forecast. The pressure gauge with will have vertical orientation, title and scale with start and end values. Also, all elements of this gauge should have specific colors. Let's save the appropriate options before map creation:

```
var pressureGaugeOptions = {
        title: {
                text: "Pressure,\ninHg",
                font: {
                        color: "#232323"
                }
        },
        geometry: {
                orientation: "vertical"
        },
        scale: {
                startValue: 10,
                endValue: 40,
                label: {
                        font: {
                                color: "#7E8AB6"
                        }
                }
        },
        valueIndicator: {
                color: "#7E8AB6"
        }
};
//...
var map...
```

Let's customize the second gauge for atmospheric humidity in the same way:

```
//...
var humidityGaugeOptions = {
        title: {
                text: "Humidity,\n%",
                font: {
                        color: "#232323"
                }
        },
        geometry: {
                orientation: "vertical"
        },
        scale: {
                label: {
                        font: {
                                color: "#75C0E0"
                        }
                }
        },
        valueIndicator: {
                color: "#75C0E0"
        }
};
//...
```

```
var map...
```

Let's customize the weather forecast chart. We need one data series with argument and value fields. The chart legend should be invisible.

```
//...
var chartOptions = {
      series: {
            argumentField: "date",
            valueField: "high",
            color: "#556FA6"
      },
      legend: {
            visible: false
      }
};
//...
var map...
```

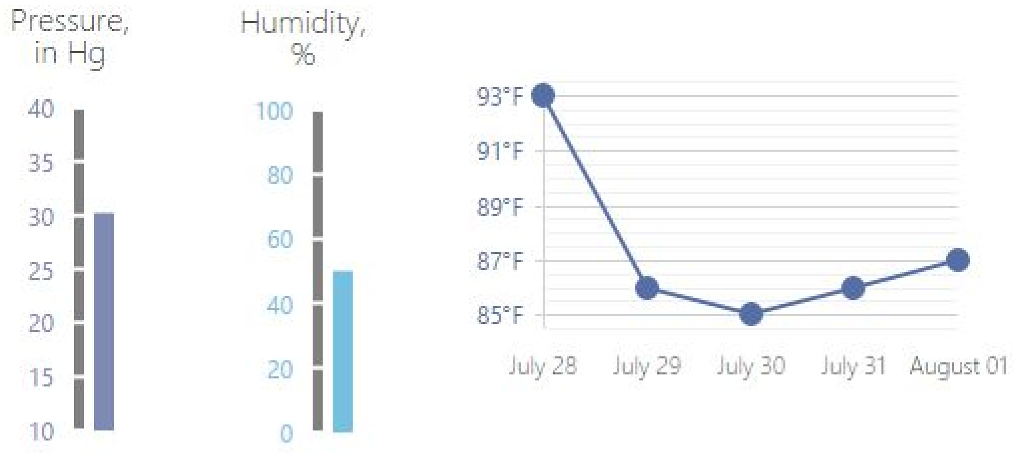
The argument axis should have information about data type. In our case, it's “datetime”. Also we need to customize margins and labels for this axis:

```
//...
var chartOptions = {
      //..
      argumentAxis: {
            argumentType: "datetime",
            valueMarginsEnabled: false,
            tickInterval: {
                  days: 1
            },
            label: {
                  format: "MonthAndDay",
                  overlappingBehavior: {
                        mode: "ignore"
                  },
                  font: {
                        color: "#232323",
                        weight: 200
                  }
            }
      }
};
```

The value axis also should have the information about data type. In this case, the type is “numeric”. This axis will have visible axis line, ticks and minor grids.

```
//...
var chartOptions = {
      //...
      valueAxis: {
            valueType: "numeric",
            visible: true,
            tick: {
                  visible: true
            },
            minorGrid: {
                  visible: true
            },
            tickInterval: 1,
            minorGridCount: 3,
            label: {
                  customizeText: function (arg) {
                        return arg.valueText + "&#176;F";
                  },
                  font: {
                        color: "#556FA6"
                  }
            }
      }
};
```

After steps described above, our widgets will look like the following:



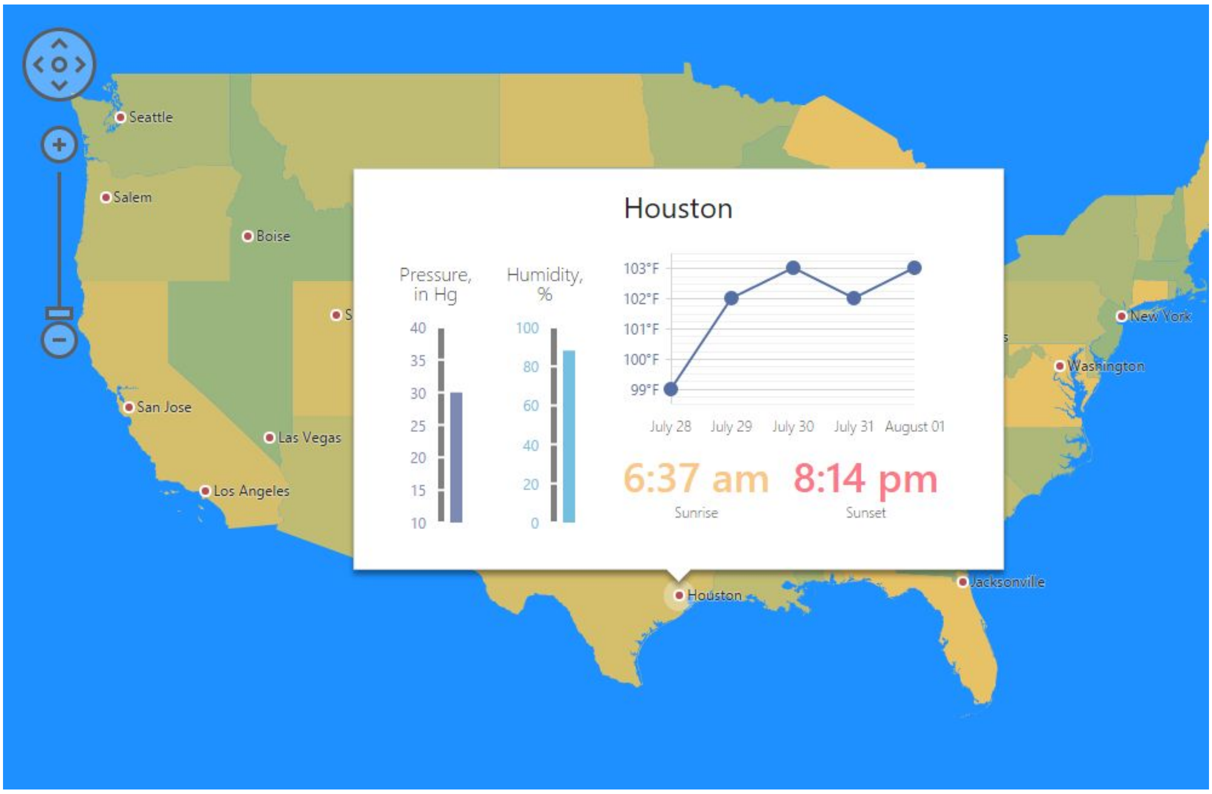
Step 5. Everything is ready to show the tooltip. Let's define the tooltip using the **onTooltipShown** option of our map widget. We'll get the target data from event argument object and submit this data to widgets and dynamic content of our tooltip:

```
//...
var map = $(".container").dxVectorMap({
      //…
      onTooltipShown: function (e) {
            var marker = e.target;

            $(".city-title").text(marker.text);
            $(".sunrise-time").text(marker.sunriseTime);
            $(".sunset-time").text(marker.sunsetTime);

            $(".pressure-gauge").dxLinearGauge(pressureGaugeOptions).dxLine
      arGauge("instance").value(marker.pressure);
            $(".humidity-gauge").dxLinearGauge(humidityGaugeOptions).dxLine
      arGauge("instance").value(marker.humidity);
            $(".forecast-chart").dxChart(chartOptions).dxChart("instance").
      option("dataSource", marker.forecast);
      }
}).dxVectorMap("instance");
//...
```

Step 6. To avoid memory leaks, let's implement the function disposing of tooltip's content and pass it to the **onTooltipHidden** option:

```
//...
var $tooltipContainer;
var map = $(".container").dxVectorMap({
      //...
      onTooltipShown: function (e) {
            //...
            $tooltipContainer = $(".tooltip-container");
      },
      onTooltipHidden: function () {
            $tooltipContainer.remove();
      }
}).dxVectorMap("instance");
//...
```

That's all! Look at result:



Let's see what is happening here. We have created the map and got markers for it using an asynchronous AJAX-query. We have defined the required html-markup of the tooltip content, which is created each time the tooltip is shown. We handle the event fired each time the tooltip is shown, apply our markup and submit the data to all widgets and dynamic content. On tooltip hiding we remove markup from the tooltip to avoid memory leaks.

This pattern could be applied for creation of a tooltip containing other widgets and not only for the map. For example, you can create a chart with a tooltip that contains a pie chart. This example demonstrates that the pattern of creation of a similar tooltip is very simple, so you can use it without any problems.

Generally creation of a tooltip with HTML-markup is a good way to draw flexible customized tooltip with any user information in any visual representation.