THREE D GRAPHICS

Js:Chart

JavaScript Charting Engine

API GUIDE

MARCH, 2013

Three D Graphics | 11340 West Olympic Blvd., #352 | Los Angeles, CA. 90064

Telephone: 1.310.231.3330 | Fax: 1.310.231.3303

Web: http://www.threedgraphics.com

Copyright:

Three D Graphics

11340 West Olympic Blvd., #352, Los Angeles, CA. 90064

Telephone: 1.310.231.3330

Fax: 1.310.231.3303

Web: http://www.threedgraphics.com

MARCH, 2013

u Table of Contents u


THREE D GRAPHICS

Section 1: About Js:Chart

The JavaScript Chart Engine (Js:Chart) draws a chart in an HTML environment using a defined data set and Java Script Object Notation (JSON) definitions. The JSON object(s) define chart properties that control the appearance of the chart and chart objects.

Creating a chart is trivial.

Define the tdgchart.js file in the script tag in your HTML file.

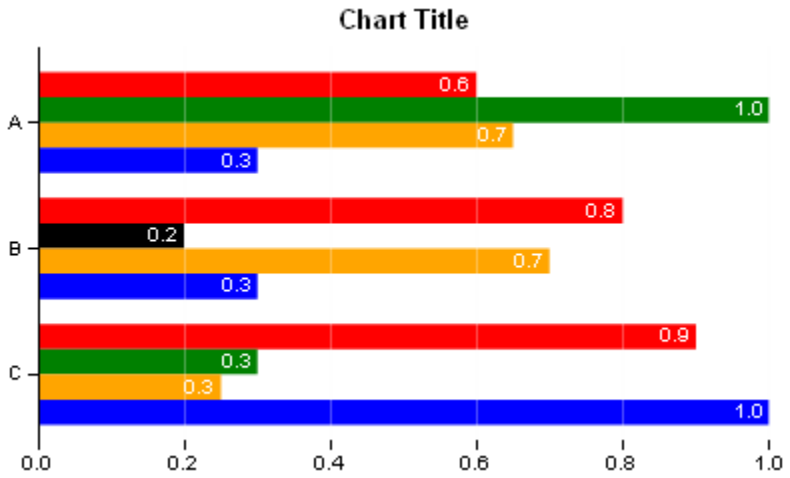
```
<script type="text/javascript" src="tdgchart.js"></script>;
```

Add a ** ** in your HTML code to define where you want to draw the chart. Example:

```
<div class='chart' id='myFirstChart'></div>
```

In any <script> tag, include:

```
<script type="text/javascript">
var chart = new tdgchart();
chart.draw('myFirstChart')
</script>
```



JSON objects define values for chart properties. In addition to complete JSON objects, Js:Chart can accept JSON objects that only include one or a few properties. These property values are simply used in place of any existing settings.

Properties can also be set via 'dot' notation. Example:

```
chart.title.text = 'My New Chart Title';
```

Requirements

A modern browser (Firefox, Safari, Chrome, IE 9) with SVG support and JavaScript enabled

Getting Started

These steps define the information you need to include in your HTML file to draw a chart.

1) Define the tdgchart.js file in the script tag.

```
<script type="text/javascript" src="tdgchart.js"></script>;
```

2) Define the target location in your web page to draw the chart. Example:

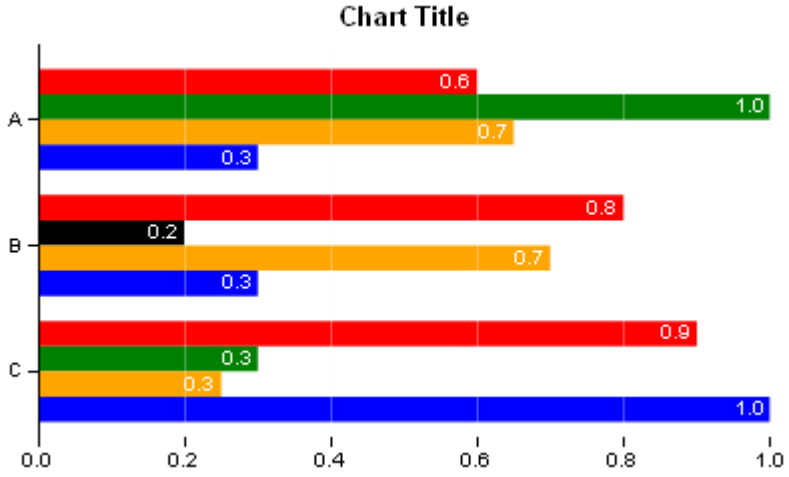
```
<div class='chart' id='chart1'></div>;
```

3) Create the chart in a script type tag:

```
<script type="text/javascript">
var chart = new tdgchart();
```

4) Draw the chart.

```
chart.draw('chart1');
</script>
```

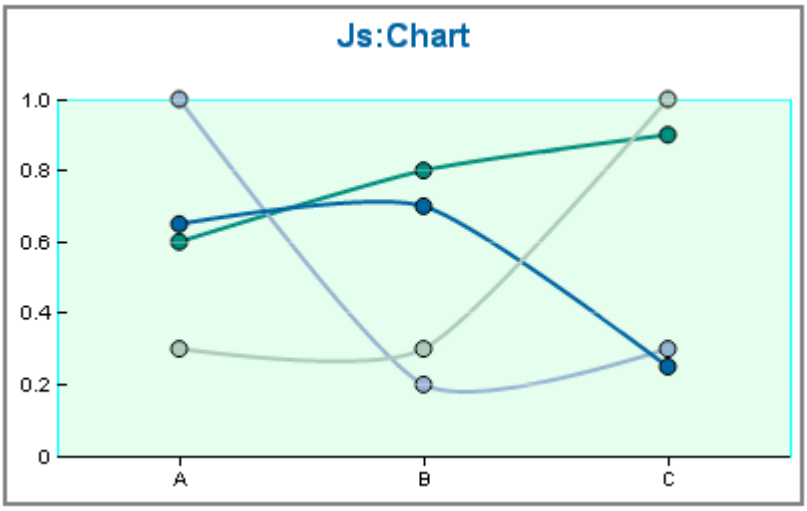


A single parameter may be supplied with the tdgchart() function to specify properties that define the appearance of a chart.

EXAMPLE:

```
var data = [
[0.6, 1.0, 0.65, 0.3],
[0.8, 0.2, 0.7, 0.3],
[0.9, 0.3, 0.25, 1.0]
];
var props = {
   data: data,
   chartType: 'line',
   border: {width: 2, color: 'grey'},
   title: {
         text: 'Js:Chart',
         font: 'bold 12pt Sans-Serif', color:'rgb(0,101,163)'},
   chartFrame: {
         border: {width: 1, color: 'cyan'},
         fill: {color: 'hsla(140, 100%, 50%, 0.1)'}},
   blaProperties: {orientation: 'vertical',lineConnection: 'curved'},
   dataLabels: {visible: false},
   series: [
   {series: 0, color:'rgb(0,142,126)', marker: {visible: true}},
```

```
   {series: 1, color:'rgb(152,181,211)', marker: {visible: true}},
   {series: 2, color:'rgb(0,101,163)', marker: {visible: true}},
   {series: 3, color:'rgb(173,204,189)', marker: {visible: true}},
   ],
};
var chart = new tdgchart(props);
chart.draw('chart1');
```



NOTES:

- Different chart types may require more than one value to draw each riser or marker (see the chartType property for details). The default properties file (e.g., properties.js) includes an example data set that will draw a bar, line, or area chart.
- If properties are not specified, property settings from a default properties file (e.g., properties.js) will be used. Your Js:Chart package may include multiple default properties files for different chart types.
- After the tdgChart() constructor, additional properties can be set using dot notation (e.g., chart.title.text = 'A New Chart Title';).
- The *backend* and *allowBackendFallback* properties can be used to choose a rendering technology (Java Script or Flash). These properties MUST be passed to the tdgchart() constructor.
- The chart instance is retrievable from the target DOM node. This allows you to retrieve the generated chart instance out of the HTML DOM node that was created to contain the chart. This makes it possible to access the chart anywhere, even if it was created by an entirely separate process. You can use this chart as any other chart (i.e., change properties, add event handlers, redraw the chart, etc.). After a chart.draw(), getElementById() provides direct access to the chart instance. Example:

```
<div id = 'myChartDiv'>
var chart = new tdgchart();
chart.draw('myChartDiv');
var div = document.getElementById('myChartDiv'); Standard javascript
var chart = div.chart;
```

These three blocks of code (creating the *div*, creating the chart, pulling the chart out of the *div*) would typically reside in different parts of an application.

Js:Chart dynamically adds a chart property to the standard *div* element in the HTML page that was retrieved by getElementById(). This allows you to set Js:Chart properties and call Js:Chart methods using div.chart. Examples:

```
div.chart.title.text = 'New Title Text';
div.chart.redraw();
```

The last line of the code segment (var chart = div.chart) just creates a convenience variable that stores 'chart' directly and allows you to set Js:Chart properties and call Js:Chart methods using chart. Examples:

```
var chart = div.chart;
chart.title.text = 'New Title Text';
chart.redraw();
```

Properties Overview

Colors & Gradients

Color properties define the color of an object. All objects can be assigned a color and transparency setting. Area and line objects can be assigned a color definition or a gradient definition.

Color Definitions

color: 'string': For color and transparency settings, the string can be one of the following:

- a color name (e.g., 'coral'),
- three RGB values (i.e., 'rgb (r,g,b)'),
- three RGB values and a transparency setting (i.e., 'rgba(r,g,b,a)'),
- three Hue-Saturation-Lightness values (i.e., 'hsl(h,s,l)'),
- three HSL values and a transparency setting (i.e., 'hsla(h,s,l,a)'), or
- hex values (e.g., '#f00' or '#ff00AA').

The World Wide Web Consortium (W3C) web site at http://www.w3.org/TR/css3-color/#colorunits provides details about color specifications.

Gradient Definitions

Gradients can be defined by a string or a JSON object.

JSON Object: This JSON object definition defines a linear or radial gradient:

```
color:  {
   type: 'string',
   start:{
         x: number | 'string',
         y: number | 'string'
   },
   end:{ // Linear Gradients Only
         x: number | 'string',
         y: number | 'string'
   },
   radius: 'string',  // Radial Gradients Only
   stops: [
               {
                     offset: number | 'string',
                     color: 'string'
               },
```

```
            ...
        ]
}
```

type: a string that defines the type of gradient: 'linear' or 'radial'

start/x: specifies the starting X-coordinate in the destination object space.

start/y: specifies the starting Y-coordinate in the destination object space.

end/x: If type is 'linear', specifies the ending X-coordinate in the destination object space.

end/y: If type is 'linear', specifies the ending Y-coordinate in the destination object space.

radius: If type is 'radial, specifies the radius of the gradient in the destination object space.

stops: a comma separated list of "offset color" pairs. This array can be any length, and can be a list of objects or arrays.

The *start: x/y* and *end: x/y* parameters can be specified as numbers (e.g., 0/20) or as strings (e.g., '5%'/'20%'). Numbers are interpreted as raw Scalable Vector Graphics (SVG) pixel coordinates. For example, start: x:0/y:0 is the object's top left corner. Negative numbers are not valid; any number less than zero is treated as zero. Numeric start/end coordinates are only useful for objects where you can set the area dimensions (e.g., the chart background area). For example, if the chart background/draw area is set to 200 by 200 pixels and start x:0/y:0 and stop x:100/y:0 is used to define a linear gradient, the gradient will be applied from the left edge to the center of the rectangle. If an object's size is calculated by the library (e.g., legend area, chart frame, risers, etc.), string percentages are typically used. For radial gradients, start: x/y defines the center of the gradient (e.g., start: x: '50%'/y: '50%' draws the gradient in the center of the object).

For radial gradients, the *radius* property defines the distance from the center to the outermost edge of the gradient. For example if an object is 200 by 200 pixels with a gradient start: x:'50%'/y:'50%' and radius: '100%', the outermost gradient edge will be 100 pixels beyond the edges of the object. In this example, 100% of 200 pixels is 200 pixels, so the gradient is actually 400 pixels from extreme left to extreme right. For a radial gradient that starts in the center (start: x:'50%'/y:'50%'), radius: '50%' is normally used to draw the gradient from the center to the outermost edges of the object.

The *stops:offset* values can be numbers or strings with '%'. Only numbers between 0 and 1 are valid, and are treated as percentages. A value 0.5 is the same as "50%". Numbers less than zero are treated as zero, and numbers greater than one are treated as 1.

string: To define a gradient in a string, use one of the following:

For a linear gradient, use a string in the following format:

```
linear-gradient(startX, startY, endX, endY, stopsOffset
stopsColor,...stopsOffset stopsColor)
```

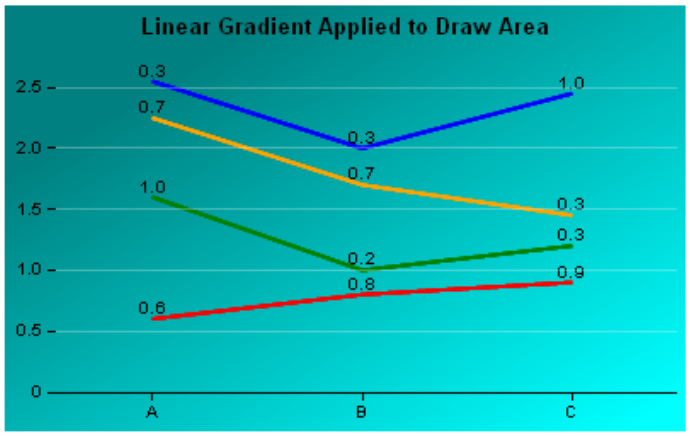
For a radial gradient, use a string in the following format:

```
radial-gradient(startX, startY, radius, stopsOffset,
stopsColor,...stopsOffset stopsColor)
```

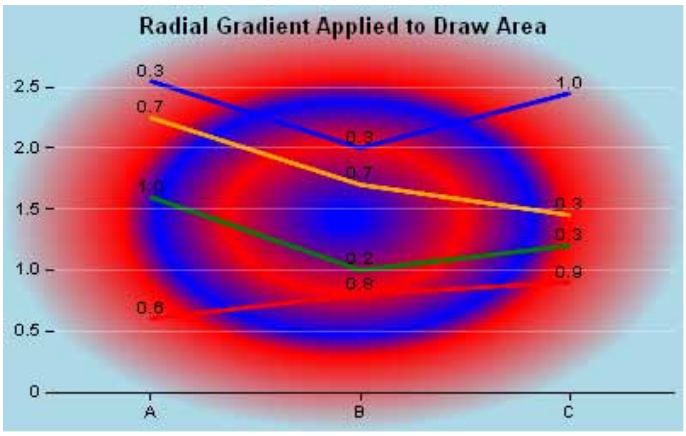
Examples:

```
fill: {
   color: 'linear-gradient(0,0,100%,100%, 20% teal, 95% cyan)'
```

```
   },
```




```
fill: {
   color: 'radial-gradient(50%,50%,50%, 20% blue, 35% red, 55% blue,
75% red, 1 lightblue)'
   },
```



You can learn more about SVG and defining gradients at: http://www.w3.org/TR/SVG/pservers.html.

Font Properties

All font properties are specified as a string (e.g., '10pt Sans-Serif') using the format defined at: http://www.w3.org/TR/CSS2/fonts.html#font-shorthand

Formatting Numbers

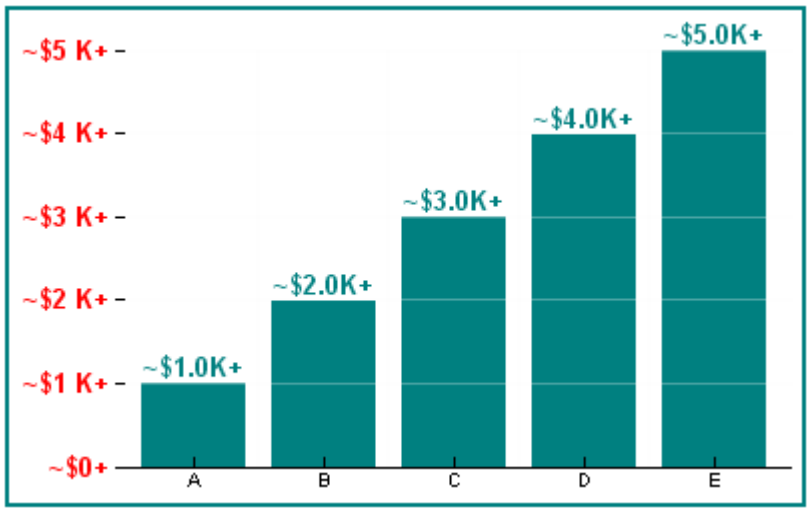
The numberFormat property specifies the format of numeric labels using one of the following:

- 'auto' (automatic - the default)
- a traditional JSON object
- a format string
- a function

auto: Use numberFormat: 'auto' to let the charting engine choose the number format based on the chart data. Prefix and suffix characters can be added to the chosen format by enclosing auto in brackets.

Example:

```
data: [[1000,2000,3000,4000,5000]],
title: {visible: false},
border: {width: 2, color: 'teal'},
blaProperties: {orientation: 'vertical'},
yaxis: {labels: {font: 'bold 10pt Sans-Serif', color: 'red'},
   numberFormat: '~${{auto}}+'
   },
dataLabels: {font: 'bold 10pt Sans-Serif', color: 'teal',
   numberFormat: '~${{auto}}+'},
series: [{series: 0, color: 'teal'}]
```



JSON object: The traditional JSON object includes the following properties to format the numeric labels:

```
numberFormat: {
   mode: 'numeric', // or 'percent', 'currency', 'scientific'
   thousandSep: ',',
   decimalSep: '.',
   decimalPlaces: 2,
   grouping: 'K', // One of 'K', 'M', 'B', 'T'
   prefix: 'a ',  // added to the front of the label
   suffix: ' b',  // added to the end of the label
}
```

mode: Use 'numeric', 'percent', 'currency', or 'scientific'

thousandSep: Character to separate values above and in multiples of a thousand (e.g., 1,000,000).

decimalSep: Character to separate decimals (e.g., 1.00)

decimalPlaces: Number of decimal places (i.e., number of digits to show to the right of the decimalSep character).

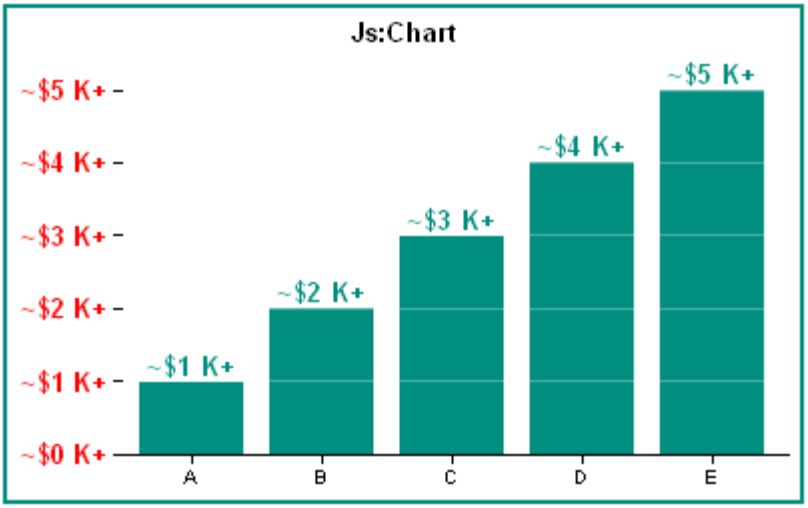
grouping: Use 'K', 'M', 'B', or 'T' to group large numbers by thousands (i.e., 1,000 = 1K), millions (i.e., 1 million = 1M), billions (i.e., 1 billing = 1B), or trillions (i.e., 1 trillion = 1T).

prefix: Character(s) to add to the front of the label

suffix: Character(s) to add to the end of the label

Example:

```
yaxis: {
   labels: {font: 'bold 10pt Sans-Serif', color: 'red'},
   numberFormat: {
         mode: 'currency',
         decimalPlaces: 0,
         grouping: 'K',
         prefix: '~',
         suffix: '+',
   }
}
```



Format String: You can also specify the format of numeric labels using a format string:

```
numberFormat: 'string'
```

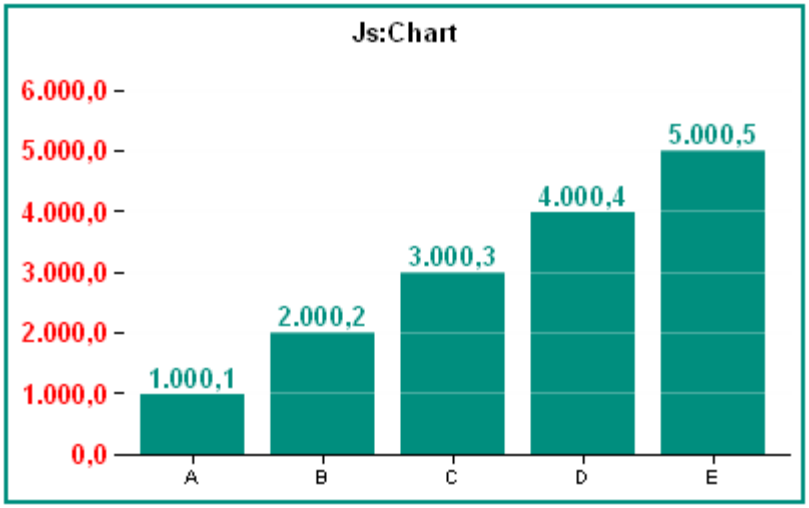
The number format string must be in the format defined by the MSDN .NET Framework 4 Custom Numeric Format Strings defined at: http://msdn.microsoft.com/en-us/library/0c899ak8(vs.71).aspx

In addition to the format specifiers defined here, you may also use the following characters:

- 't' to specify a thousands separator character.
- 'd' to specify a decimals separator character.
- back tick (`): A percent sign (%) in a format string causes a number to be multiplied by 100 before it is formatted. Add a back tick in front of the percent sign (`%) to disable the multiplication.

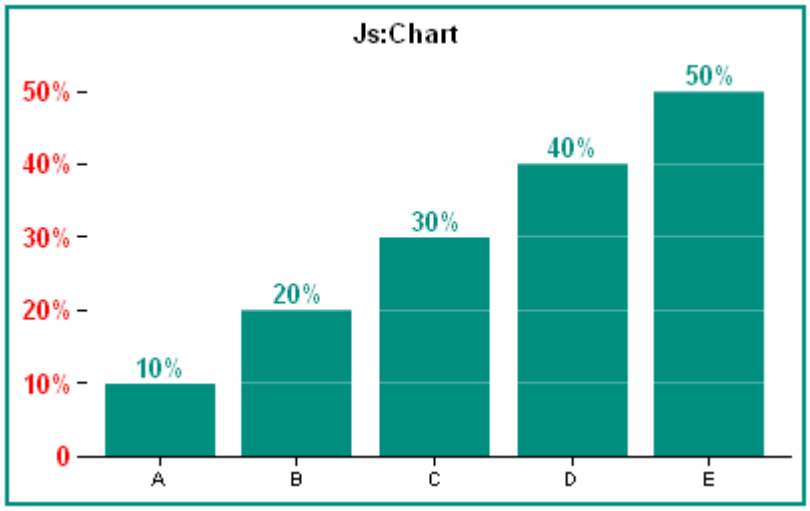
Example:

```
// format 4000.5 as '4.000,5' (European standard)
chart.yaxis.numberFormat = 't.d,#,#.#'
```



Example:

```
// format percentage without multiplier
chart.yaxis.numberFormat = '[>9]#`%;[<0]-#`%;[<=9]#`%;[=0]0'
```



Function: The numberFormat property can also process a user-defined function for fully custom number formatting callbacks. The expected function takes one argument (the number to format) and returns a string. Example:

```
// Return number with a '$' in front
chart.yaxis.numberFormat = function(n){ return '$' + n; };
```

This cannot be used inside the traditional JSON-style blocks { x: y }. It can only be used with the 'dot' notation as shown in this example.

NOTES:

- Except with numberFormat auto, do not use a semicolon (;) as a prefix, suffix, thousands separator, or decimal separator character. A semicolon is only valid is for separating multiple conditional formats (e.g., '[>=0]#,#;[<0]-#.#').

- Except with numberFormat auto, do not use a pound sign (#), period (.) or question mark (?) as a prefix or suffix character. These are special characters that cannot be interpreted as a string.
- If a percent sign (%) is used as the prefix or suffix character in a JSON object definition, the number format will use the percent mode regardless of the mode setting.
- If you use the string format to specify the range of data to which the number format is applied, make sure the format string includes the entire range of data. Example:

```
numberFormat: '[>9]#,#;[<0]-#.#;[<9]#.#;[=0]0'
```

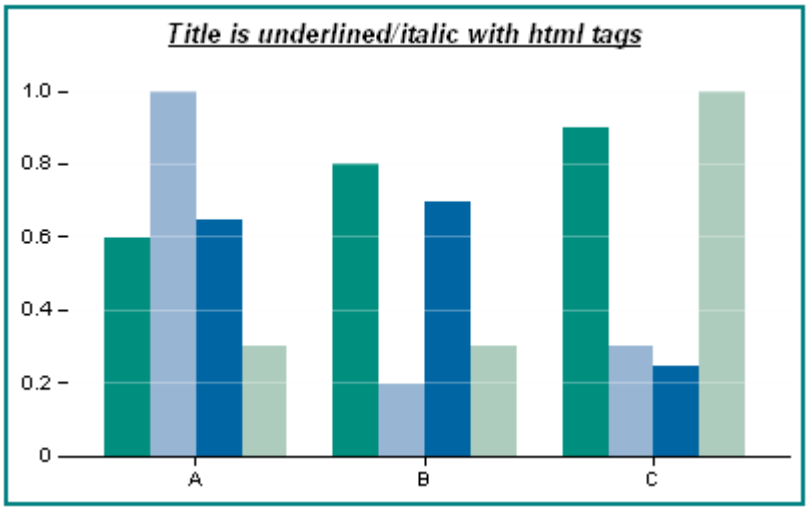
This format string defines the format for values less than nine and greater than nine but does not specify the format for a value equal to nine. The following corrected format string will cover the entire range of data:

```
numberFormat: '[>=9]#,#;[<0]-#.#;[<9]#.#;[=0]0'
```

HTML Codes in Strings

HTML codes can be embedded in any label or title strings: HTML codes in titles and labels are not processed by the library. They are simply passed on to the browser and will appear as they are rendered by the browser. Example:

```
title: {text: '<u><em>Title is underlined/italic with html
tags</em></u>'},
```



Data & Property Definitions

The data that draws a chart is defined in the form of one or more arrays. Different chart types have different data requirements. The chartType property defines the number and format of values required to draw each chart type.

Null Data Support

You can use null, undefined, or multiple commas (,,) in a data array to identify missing data points. Js:Chart will allocate space for the missing data but the riser/marker will not appear in the chart. Examples of valid missing data definitions:

```
[[1,2,null,4]]
```

```
[[1,2,undefined,4]]
[[1,2,,3]]
```

You cannot replace an entire array of data with null but an empty array is valid:

```
[[1,2], null, [3,4]] // invalid
[[1,2], [], [3,4]] // valid
```

Property Values

All properties can be assigned a value of undefined or null (not quoted strings). When undefined or null is used as a property value, the property is ignored. The default value assigned to the property in the default properties file (e.g., properties.js) is used.


THREE D GRAPHICS

Section 2: Methods

tdgchart

This method creates a new chart with the properties specified by the input parameter *props*. If properties are specified, default properties will be used.

SYNTAX:

```
var chart = new tdgchart(properties);
```

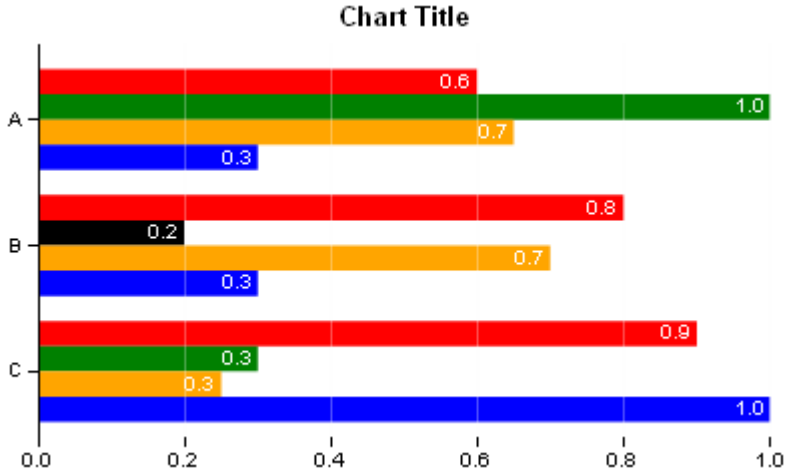
PARAMETERS:

properties; (optional) If specified, identifies a variable where properties are defined

EXAMPLES:

This example creates and draws a chart using the default properties.

```
var chart = new tdgchart();
chart.draw(aChartName);
```



This example creates and draws a chart using the properties defined in *props*.

```
var props = {
   chartType: 'line',
   border: {width: 2, color: 'grey'}
};
var chart = new tdgchart(props);
chart.draw(aChartName);
```

NOTES:

The *backend* and *allowBackendFallback* properties can be used to specify an output format (i.e., Java Script or Flash). These properties MUST be passed to the tdgchart() constructor. Example:

```
var props = {
   backend: 'flash',
   allowBackendFallback: false
};
var chart = new tdgchart(props);
```

draw

After the tdgchart() chart constructor, this method draws a chart.

SYNTAX:

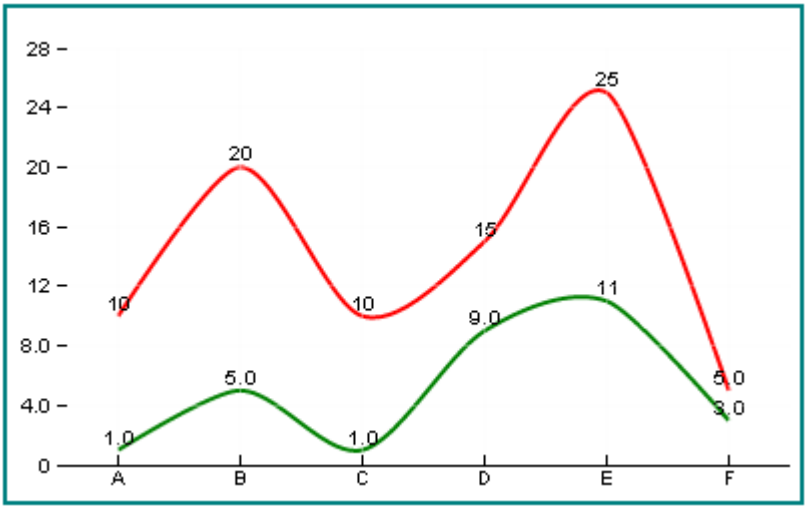
```
draw(aChartName);
```

PARAMETERS:

aChartName; identifies a chart name defined in a <div ... id='chartname'> tag.

EXAMPLE:

```
<div class='chart' id='chart1'>Optional text for unsupported
browsers.</div>;
props = {
catchErrors: false,
data: [[10,20,10,15,25,5],[1,5,1,9,11,3]],
title: {visible: false},
border: {width: 2, color: 'teal'},
chartType: 'line',
blaProperties: {orientation: 'vertical', lineConnection: 'curved'}
};
var chart = new tdgchart(props);
chart.draw(chart1);
```



linkDataSelectionCharts

This method provides an easier way to link several chart instances together, intstead of setting each chart.dataSelection.linkedCharts array manually (using the dataSelection property).

SYNTAX:

```
linkDataSelectionCharts([array of tdgchart instances to link]);
```

EXAMPLE:

```
var a = new tdgchart(), b = new tdgchart(), c = new tdgchart();
// link 3 charts together
a.linkedCharts = [b, c];
b.linkedCharts = [a, c];
c.linkedCharts = [a, b];
// OR, link 3 charts together:
tdgchart.linkDataSelectionCharts([a, b, c]);
```

morph

This method allows you to draw a chart, change properties, and then redraw the chart with a 'morph' animation that transitions between the initial and final chart.

SYNTAX:

```
morph(properties);
```

PARAMETERS:

properties; (optional) If specified, identifies a variable where properties are defined

EXAMPLE:

```
// Draw a bar chart, then morph it into a pie chart.
var chart = new tdgchart();
chart.chartType = 'bar';
chart.draw('div');
chart.chartType = 'pie';
chart.morph();
```

NOTES:

chart.draw must be called at least once before chart.morph

redraw

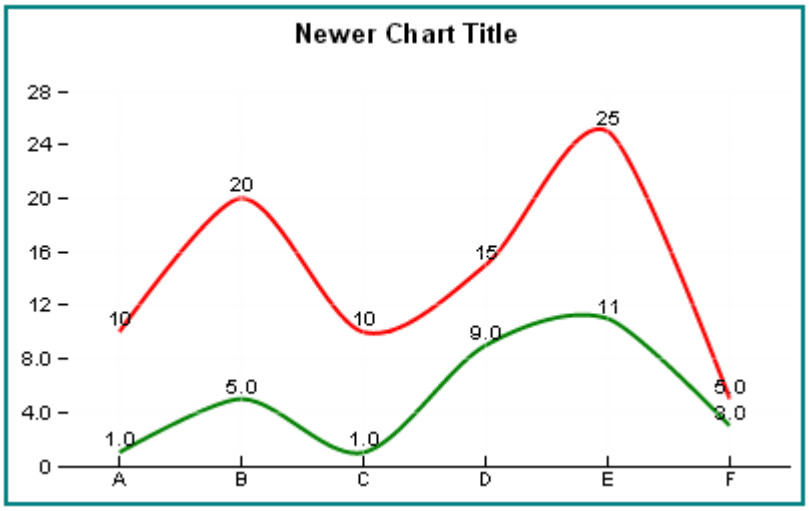
If chart.draw('htmlElement') has already been called, this function does a redraw to the same element. It will produce an error if draw() is ever called without a target or if redraw() is called before a target is set.

SYNTAX:

```
redraw();
```

EXAMPLE:

```
props = {
catchErrors: false,
data: [[10,20,10,15,25,5],[1,5,1,9,11,3]],
title: {text: 'New Chart Title'},
border: {width: 2, color: 'teal'},
chartType: 'line',
blaProperties: {orientation: 'vertical', lineConnection: 'curved'}
};
var chart = new tdgchart(props);
chart.draw('chart1');
chart.title.text = 'Newer Chart Title';
chart.redraw();
```



registerEvent

This method identifies a function to be called when an event (e.g., mouse click) occurs related to a chart object.

SYNTAX:

```
registerEvent(callback, event, object, userInfo, series, group, misc,
context);
```

PARAMETERS:

callback; the function to be called when the *event* is triggered.

event; a string matching the standard Document Object Model (DOM) events ('click', 'mousemove', etc.). In addition to standard DOM events, Js:Chart supports 'renderComplete'. When chart.registerEvent(callback, 'renderComplete') is used, the chart engine will call the passed in callback function when the chart has totally finished drawing (including animation). You can register as many 'renderComplete' events as you wish; they will be called in creation order.

object; a string referring to a chart object (e.g., 'title', 'riser').

userInfo: any object, which will be passed directly back to the callback.

series; an integer, used to add an event for a specific series only.

group; an integer, used to add an event for a specific group only.

misc; miscellaneous ID (string) of the object to add an event to. A '!' prefix can be used to denote 'not'.

context: an object to be used as executing context for the callback function (i.e., the 'this' value inside the callback).

NOTES:

- The first three arguments are required, the remaining are optional.
- registerEvent() must be called before you draw the graph. Otherwise, you must issue a redraw in order to have the event hander applied.
- The *event* parameter is required to get FireFox events (e.g., titleClick(obj, objName, userInfo, event).
- registerEvent() also supports a single object argument, with keys matching the input parameters. Example:

```
chart.registerEvent({
callback: mycallback,
event: 'click',
object: 'riser',
context: myObject
});
```

- registerEvent() will pass a single object to the callback function. The callbackObj contains the following information:

```
callbackObj = {
   chartObj,
   chartObjName,
   userInfo,
   series,
   group,
   misc,
   svgElement,
   chart,
```

```
   event,
};
```

chartObj: The object inside the chart instance that was interacted with

chartObjName: The object name (a string) that was interacted with

userInfo: The user object passed in as the 4th parameter to registerEvent()

series: series ID (a number)

group: group ID (a number)

misc: miscellaneous ID (a string)

svgElement: the SVG element within the browser DOM that was interacted with

chart: the chart instance that was interacted with

event: the DOM event generated by the interaction

The registered callback function should expect only one argument. Example:

```
function myCallback(callbackObj) {
    var chartObj = callbackObj.chartObj;
    var seriesID = callbackObj.series;
    var xPos = callbackObj.event.clientX;
    etc, etc
}
```

Example Drill Down:

```
props = {
   catchErrors: true,
   dataSubset: {startGroup: 0, stopGroup: 4},
   border: {width:2, color:'blue'},
   subtitle:{visible: true},
   blaProperties: {orientation: 'vertical'},
   legend: {visible: true, position: 'bottom'},
   dataLabels: {numberFormat: '#'},
   yaxis: {bodyLineStyle: {color: 'black'}},
   series: [
         {series: 0, color: 'red'},
         {series: 1, color: 'green'},
         {series: 2, color: 'blue'},
         {series: 3, color: 'yellow'},
   ]
   };
var yearData = [
   [8, 9, 8, 3, 6, 9, 3, 2, 1, 4],
   [7, 7, 3, 8, 5, 1, 2, 9, 7, 1],
   [3, 3, 9, 3, 4, 7, 7, 7, 2, 3],
   [4, 2, 8, 1, 9, 4, 3, 8, 4, 5]
];
// This is a data grid of 4 (years) x 4 (quarters) x 2 (sales
regions) x 4 (products)
var quarterData = pv.range(4).map(function(el){
   return pv.range(4).map(function(el) {
   return pv.range(2).map(function(el) {
   return pv.range(4).map(function(el){return pv.random(0, 50);});
   });
   });
   });
var yearLabels = '2009 2010 2011 2012'.split(' ');
var quarterLabels = 'Q1 Q2 Q3 Q4'.split(' ');
var chart = new tdgchart(props);
```

```
byYear(chart);
var zoomed = false;
function riserClick(callbackObj) {
   if (zoomed)
         return;
   this.title.text += " - Click here to zoom out";
   this.subtitle.text = this.groupLabels[callbackObj.group] + " Sales
for " + this.getSeries(callbackObj.series).label;
   this.data = quarterData[callbackObj.group][callbackObj.series];
   this.groupLabels = quarterLabels;
   this.setSeriesLabels(['East Coast Sales', 'West Coast Sales']);
   this.redraw();
   zoomed = true;
   }
function titleClick() {
   if (!zoomed)
         return;
   byYear(this);
   this.redraw();
   zoomed = false;
}
function byYear(chart) {
   chart.title.text = "Drill Down Example";
   chart.subtitle.text = "Sales by year";
   chart.data = yearData;
   chart.groupLabels = yearLabels;
   chart.setSeriesLabels(['Cars', 'Boats', 'Bicycles',
'Motorcycles']);
}
chart.registerEvent(titleClick, 'click', 'title');
chart.registerEvent(titleClick, 'click', 'subtitle');
chart.registerEvent(riserClick, 'click', 'series');
chart.draw('chart1');
```

selectData

This method allows a developer to tell the charting library to select one or more risers in the chart. It should only be used *after* the chart has been drawn (e.g., chart.draw('chart1');). Calling chart.selectData() before the chart has been drawn has no effect. This method will redraw the chart.

SYNTAX:

```
chart.selectData(selList | groupLabel, preventCallback);
```

PARAMETERS:

selList is an array of {series: x, group: y} objects, which define the set of risers to select.

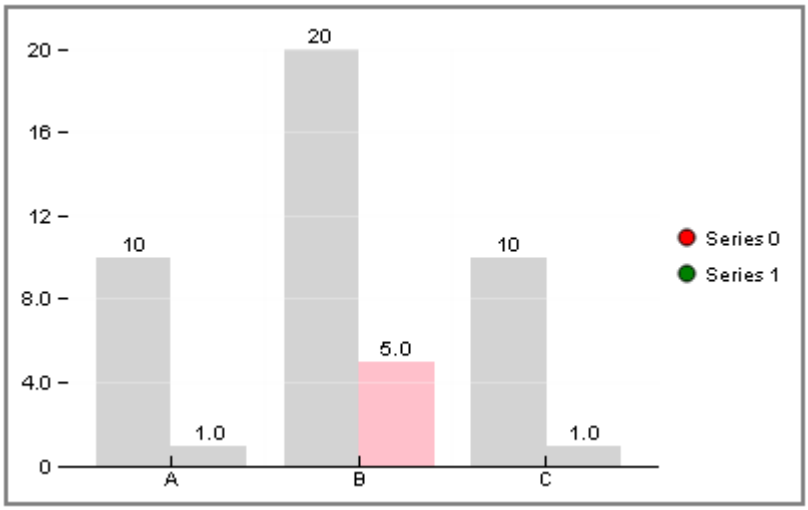
groupLabel: An optional groupLabel can be specified for charts such as treemaps that do not have series/group IDs. It can be a single node (e.g., groupLabel: 'california') or a hyphenated list of 'parent-child-child' labels (e.g., region-west-california'). If the group label matches any treemap node label or any of its ancestor's labels, the node will be selected. For example, groupLabel: 'East' will select all nodes and all children of nodes with a label of 'East'. For other chart types, a group label will select all risers that have a matching group label.

preventCallback is an optional boolean that tells the charting library if it should call any registered event callbacks when updating this selection list. If false, chart.dataSelection.eventCallback will be called (if it is defined). If true, that callback will *not* be called. This parameter is useful when two charts are not directly linked via tdgchart.linkDataSelectionCharts() and you want to manually maintain the selection lists between the two charts.

EXAMPLE:

In a bar chart with two series and three groups, this example will set the series zero/group one riser to "selected" and all other risers as "unselected". The dataSelection.selectedColor and dataSelection.unselectedColor properties define the color of selected and unselected risers.

```
props = {blaProperties: {orientation: 'vertical'},
dataSelection: {enabled: true, selectedColor: 'pink',unselectedColor:
'lightgrey'},legend: {visible: true}
};
var chart = new tdgchart(props);
chart.draw('chart1');
chart.selectData({series:1, group:1});
```



set

This method sets a group of chart properties.

SYNTAX:

```
set({group:{propertyName: value}});
```

PARAMETERS:

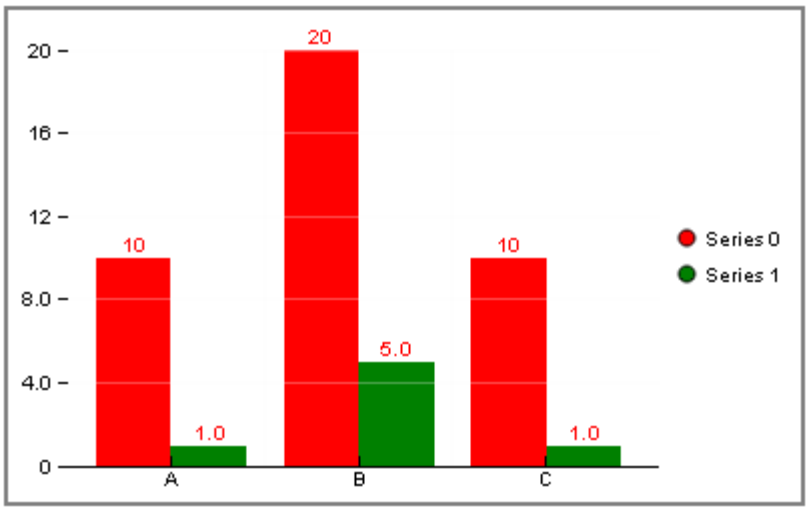
group; a property group name: blaProperties, pieProperties, legend, title, subtitle, footnote, yaxis, dataLabels, series.

propertyName; a property name from the property group

value; a valid property value

EXAMPLE:

```
props = {
catchErrors: false,
data: [[10,20,10],[1,5,1]],
title: {visible: false},
border: {width: 2, color: 'grey'},
blaProperties: {orientation: 'vertical'},
legend: {visible: true}
};
var chart = new tdgchart(props);
chart.set({dataLabels:{color: 'red'}});
chart.draw('chart1');
```



setSeriesColors

This method uses the passed in array of colors as the series colors.

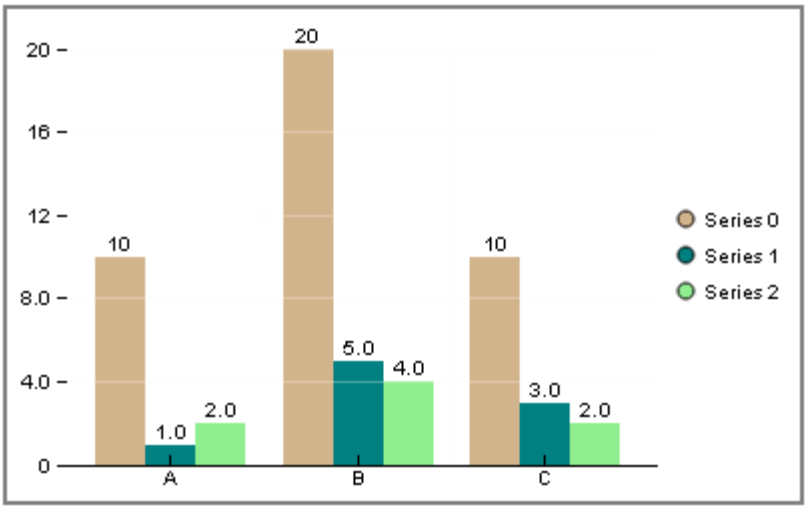
```
setSeriesColors(array)
```

PARAMETERS:

array; an array of colors

EXAMPLE:

```
props = {
catchErrors: false,
data: [[10,20,10],[1,5,3],[2,4,2]],
title: {visible: false},
border: {width: 2, color: 'grey'},
blaProperties: {orientation: 'vertical'},
legend: {visible: true}
};
var chart = new tdgchart(props);
chart.setSeriesColors(['tan', 'teal', 'lightgreen']);
chart.draw('chart1');
```



setSeriesLabels

This is a convenience function used to set all series labels with one function.

SYNTAX:

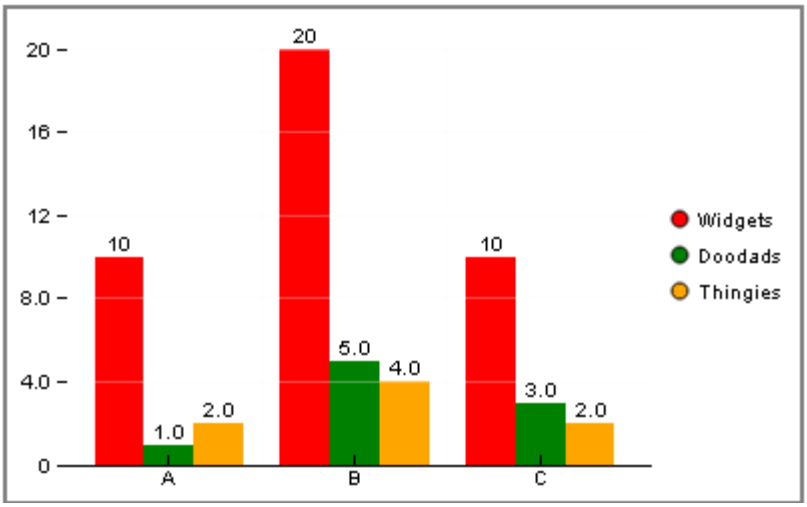
```
setSeriesLabels(labels)
```

PARAMETERS:

labels; an array of series label strings. The array is expected to be initialized to the desired length, with the desired series entries, before calling setSeriesLabels(). If a series does not exist, the associated label is ignored.

EXAMPLE:

```
props = {
catchErrors: false,
data: [[10,20,10],[1,5,3],[2,4,2]],
title: {visible: false},
border: {width: 2, color: 'grey'},
blaProperties: {orientation: 'vertical'},
legend: {visible: true}
};
var chart = new tdgchart(props);
chart.setSeriesLabels(['Widgets', 'Doodads', 'Thingies']);
chart.draw('chart1');
```



setSeriesProperty

This method assigns an array of property values to a property for multiple series.

```
setSeriesProperty(prop, propArray)
```

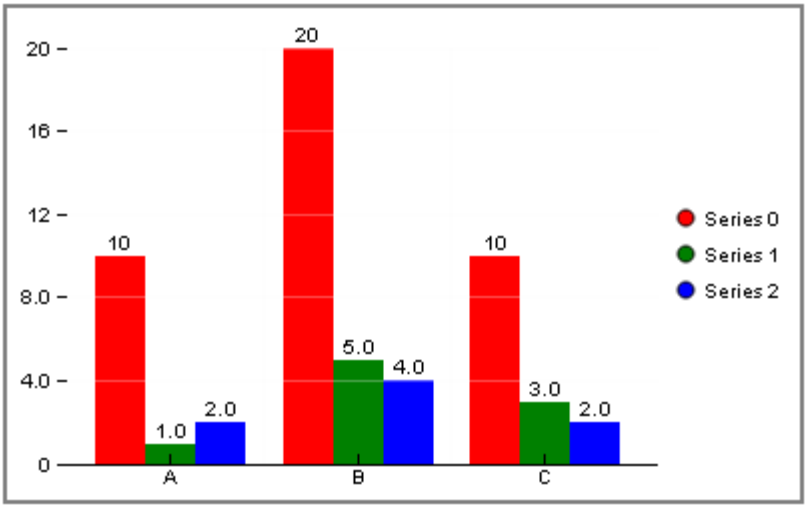
PARAMETERS:

prop: a string matching a top level series dependent property (like 'color').

propArray a list to set the chosen property to.

EXAMPLE:

```
props = {
catchErrors: false,
data: [[10,20,10],[1,5,3],[2,4,2]],
title: {visible: false},
border: {width: 2, color: 'grey'},
blaProperties: {orientation: 'vertical'},
legend: {visible: true}
};
var chart = new tdgchart(props);
chart.setSeriesProperty('color', ['red', 'green', 'blue']);
chart.draw('chart1');
```



Abstraction Methods

buildClassName

buildClassName is an abstraction method to generate a class name.

SYNTAX:

```
buildClassName = function(obj, s, g, m)
```

PARAMETERS:

obj; a string describing a chart object.

s: series number

g: group number

m: a number that identifies a specific instance of an object

EXAMPLE:

```
.className(function(){return
chart.buildClassName('waterfallProperties-connectorLine',
this.parent.index, this.index);})
.className(function(){return chart.buildClassName('marker',
this.parent.index, this.index);})
```

classNameLookup

classNameLookup is an abstraction method to generate a class name.

SYNTAX:

```
classNameLookup = function(miscID)
```

PARAMETERS:

miscID: a string defining a chart object.

EXAMPLE:

```
.className(chart.classNameLookup('line'))
.className(chart.classNameLookup('wedge'))
```


THREE D GRAPHICS

Section 3: Properties

Properties define the overall appearance of the chart and individual chart objects.

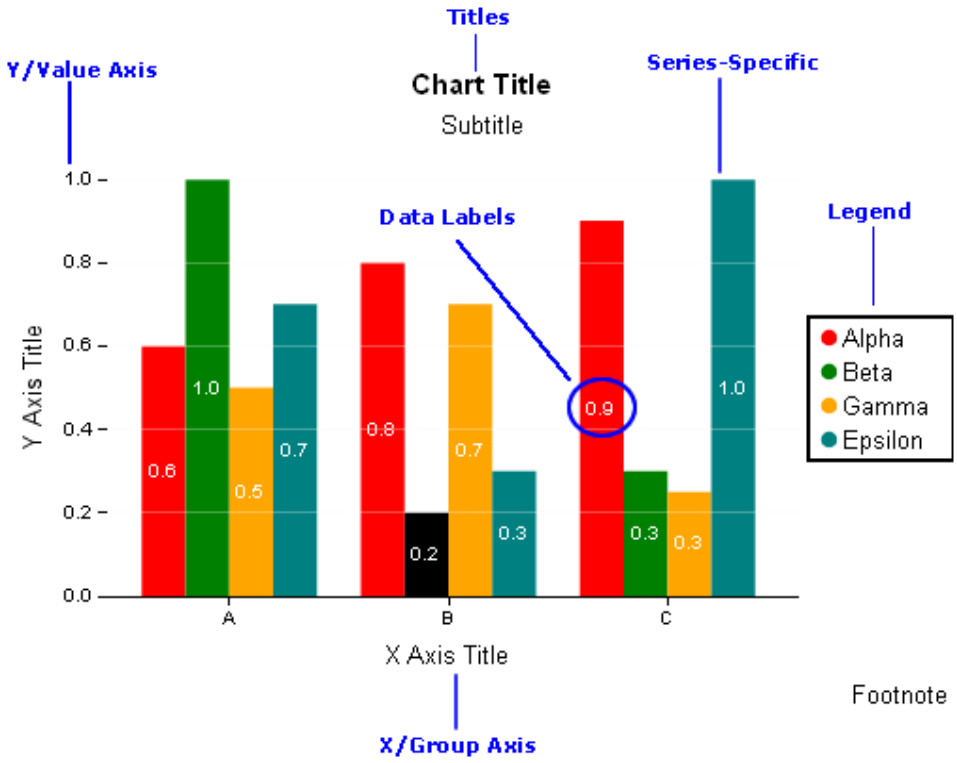


Chart Types (chartType)

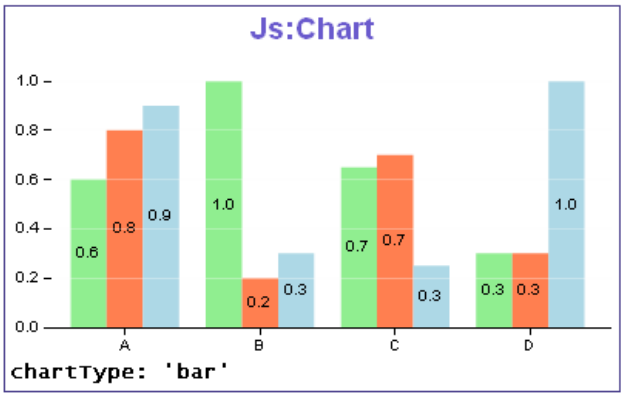
This property defines the chart type.

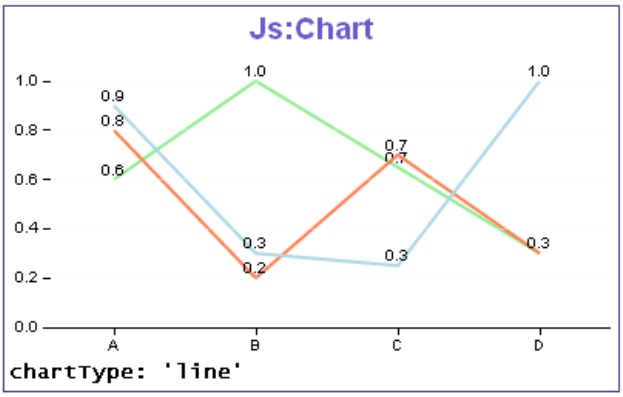
```
chartType: 'string'
```

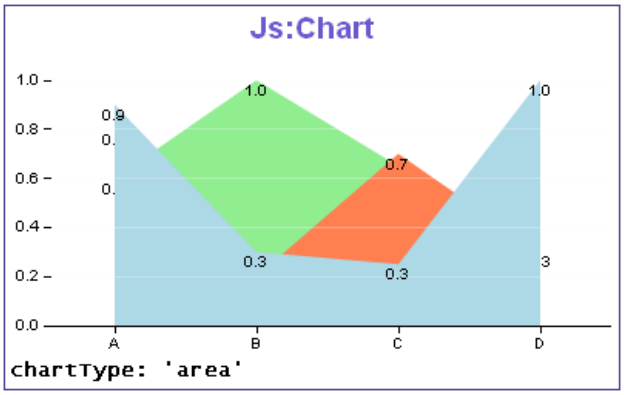
PARAMETERS:

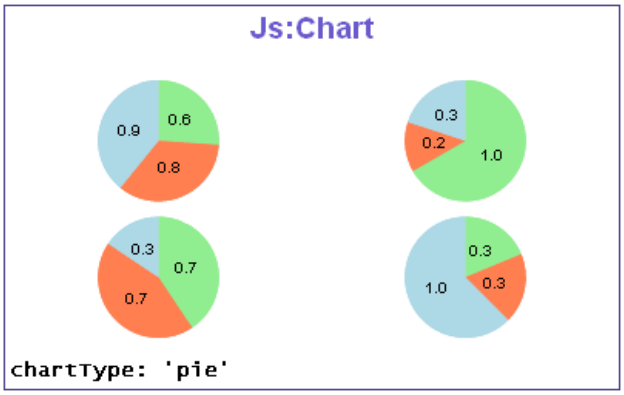
chartType: a string that defines the chart type: 'area', 'area3d', 'bar', 'bar3d', 'boxplot', 'bubble', 'bullet', 'funnel', 'gantt', 'gauge', 'heatmap, 'histogram', 'legend', 'line', 'mekko', 'parabox', 'pareto', 'pie', 'polar', 'radar', 'scatter', 'sparkline', 'stock', 'streamgraph', 'surface3d', 'tagcloud', 'treemap', 'waterfall'. The default value is 'bar'.

EXAMPLES:









NOTES:

- chartType: **'area3d'**, **'bar3d'**, **'surface3d'**: Use threedProperties to define the general layout of these charts.
- chartType: **'area'**: Use blaProperties to define the general layout of an area chart.
- chartType: **'bar'**: Use blaProperties to define the general layout of a bar chart.
- chartType: **'boxplot**: Each series and group requires one array of five values. For example to define 1 series with 3 groups (3 boxes), you need an array of arrays of arrays such as: data = [[[1,2,3,4,5],[2,3,4,5,6],[3,4,5,6,7]]]. For a given series and group box, the first value is the minimum (lower 'hat'), the second defines the box bottom, the third value is median line, the fourth value defines the box top, and the fifth value defines the location of the top hat. Values must be in ascending order. Use boxPlotProperties to define the general appearance of a box plot. Use blaProperties.orientation to draw a horizontal or vertical chart. Box plot fill color and border are defined by the series color and border.
- chartType: **'bubble'**: A bubble chart requires three values: [(X-Position, Y-Position, Size] to draw each bubble marker. A negative size value is treated as positive (i.e., the absolute value of the data). A null, undefined, or zero size will eliminate the marker. Sizes are proportional according to the range of size values in the data set. For example, a data set with values such as [[1,1,10], [2,2,20], [3,3,30]] would draw each marker comparatively larger. Disparate marker size values such as [[1,1,10], [2,2,2000]] will draw a large marker that exceeds the draw area. The series.marker.size property defines the size of the *smallest* marker. All other markers are scaled up from that smallest marker. Also, markers are scaled such that the *area* of the marker is proportional to its data value. X/Y Position values can be a number or a string.

  The size value can be replaced by an array of values that define slices in pie markers. Each marker must be defined by: [X-position, Y-position, [slice1, slice2, ...slice*n*]]. The size of the marker is calculated from the slice values. Example:

```
chart.data = [[
//     x     y     pie slices
    [   1,  2,     [1,2,4]],  // pie 0
    [ 2.5,  6.25,  [4,3,4]],  // pie 1
    [3.75,  3.5,   [6,11,9]]  // pie 2
]];
```

When this data format is used to define a bubble chart, set series: 'all', marker: {shape: 'pie'} to create the pie markers. pieProperties can be applied to the pie markers. You can also create pie markers on bubble/scatter charts using colorMode: byMetric.

- chartType: **'bullet'**: Bullet is a microchart that is a variation of the bar chart. It is designed to show a quantitative measure against qualitative ranges. For example, a quantitative measure such as profit or revenue could be visualized against quality (good, better, best) and related markers. Chart title and subtitle properties define labels to draw on the edge of the chart. blaProperties.orientation draws the chart in horizontal/vertical format.
- chartType: **'funnel'**: A funnel chart is basically a pie chart that shows only one group of data at a time. The series in the group are stacked in the funnel with the first series at the top and the last series at the bottom. Use funnelProperties to define the general layout of a funnel chart. Series-specific properties control the color of funnel segments.
- chartType: **'gantt'**: Gantt charts require either two or three values for each riser. The first value is always a start time. The second value is a stop time (if ganttProperties.durationValues=false), or a duration (if ganttProperties.durationValues=true). The optional third value defines the series each riser belongs to. To define and format time values on the Y-axis, the first and second values must be formatted as time strings. See the examples in the HTML document for details. Use ganttProperties to define the general appearance of a gantt chart.
- chartType: **'gauge'**: Use gaugeProperties to define the general layout of the gauge. The numeric yaxis properties control scaling, tick marks, and colors assigned to segments of the gauge axis.
- chartType: **'heatmap'**: Heatmaps expect the same kind of data as Bar/Line/Area charts (an array of arrays). Each internal array forms a row in the heatmap table. Series labels are plotted on the left edge according to settings in zaxis properties. Group labels are plotted on the bottom edge according to xaxis properties. The yaxis.colorScale.color property controls the color of the cells in the heatmap table.
- chartType: **'histogram'**: A histogram is a graphical representation that shows a visual impression of the distribution of data. Use histogramProperties to control the distribution of data. Use blaProperties.orientation to draw the chart in horizontal or vertical format. Use blaProperties.barGroupGapWidth to control the width of histogram risers. Use yaxis properties to control the format of Y-axis title and labels. Use xaxis properties to control the format of X-axis title and labels. In a vertical histogram, the Y-axis is on the left side of the chart and the X-axis is drawn on the bottom of the chart. In a horizontal histogram, the X-axis is on the left side of the chart and the Y-axis axis is drawn on the bottom of the chart.
- chartType: **'legend'**: This chart type draws a chart where the legend *is* the chart. Selecting this chart type will resize the legend to the chosen width and height. Legend.position controls the entries layout: 'bottom' gives a horizontal legend. Any other position draws a vertical legend chart. Horizontal aligns all entries on the left edge, evenly spaced vertically. Vertical aligns entries along the top edge, evenly spaced horizontally. Use legend properties to control the format of the legend chart.
- chartType: **'line'**: The data that defines a line riser in a line chart can be an array of single values (one for each series/group data point) or two values nested in arrays that define a data point and a marker size (e.g.,

[[1,2],[2,12],[1,4]], etc.). To use this format, set series: 'all', marker: {visible: true}. A negative marker size value is treated as positive (i.e., the absolute value of the data). A null, undefined, or zero marker size will eliminate the marker. Marker sizes are proportional according to the range of size values in the data set. For example, a data set with values such as [[1,10], [2,20], [3,30]] would draw each marker comparatively larger. Disparate marker size values such as [[1,10], [2,2000]] will draw a large marker that exceeds the draw area. The series.marker.size property defines the size of the *smallest* marker. All other markers are scaled up from that smallest marker. Also, markers are scaled such that the *area* of the marker is proportional to its data value. If marker sizes are not defined in the data set, the series:marker properties can be used to control the visibility, size (in pixels), and format of markers at each data point. Use blaProperties to define the general layout of a line chart.

- chartType: **'mekko'**: A mekko (also called marimekko) chart is a percent bar chart, except that the width of each bar riser is based on the stack's overall value. You can use any data set that works for a regular or stacked bar chart.
- chartType: **'parabox'**: Parallel coordinates is a popular method of visualizing high-dimensional data using dynamic queries. The parabox chart type is similar to a regular line chart, except that each group in the line chart has a unique and interactive numeric axis. Each colored line represents one series of data. Each vertical bar represents a numeric axis. You can click and drag along each of the axes to select (filter) the lines that pass through that part of the axis. You can add a filter to multiple axes, which lets you narrow down the data set to match certain criteria. A single click on one axis makes that axis 'active'. In the default configuration, the series lines are colored by interpolation, such that the lines passing through the top of the active axis are red and the bottom of the axis are blue. The groupLabels property defines the set of vertical axes and their labels. Chart data is an array of arrays, where each inner array becomes a single colored line (series) in the chart. Each inner array must have a value corresponding to each vertical axis (if it doesn't, the array is ignored). You can set the active axis programmatically using the paraboxProperties.activeGroup property. Js:Chart also supports *categorical* (ordinal) vertical axes. These are drawn with a bubble marker on the vertical axis. The size of the bubble corresponds to the number of lines passing through it. Whether an axis is numeric or categorical is defined by the *first* series of data. If a value in the first array is a number, that axis is numeric. If a value in the first array is a string, that axis is categorical. Categorical axes are sorted automatically such that the biggest bubble is on top and bubble size descends from there. These charts have 3 coloring modes: 'bySeries', 'byGroup', and 'byHeight'. The default byHeight color mode uses the colors defined in colorMode.colorList for the byHeight interpolation. The bySeries color mode treats each line as a unique series, and uses the colors defined in the series array (e.g., If you have 50 lines, you need to define 50 series with 50 colors). The byGroup color mode also uses the colors in the series array, but if either the *first* vertical axis, or the *active* vertical axis is *categorical*, then all lines passing through the first bubble on that axis are colored with the same series1 color; all lines passing through the second bubble are colored with the series 2 color, etc. If the first axis and the active axis are both numeric, then byGroup is ignored and the chart is drawn byHeight. The xaxis.labels property can be used to format the labels above each ordinal division.
- chartType: **'pareto'**: A pareto chart is similar to a combo chart. The first series (series zero) draws as an absolute bar chart on the Y-axis. The second series (series one) draws as a cumulative percent line on the Y2-axis.

- chartType: **'pie'**: Use pieProperties to define the general layout of the pie. Use the series-specific properties to color pie slices, show/hide data values, and explode/delete pie slices.
- chartType: **'polar'**: A polar chart is a circular scatter chart. Like scatter charts, a polar chart requires two values to draw each marker. polarProperties control the general appearance of a polar chart. yaxis properties control the circular grid around polar chart markers. These properties also control the appearance of axis labels and gridlines. xaxis properties control the appearance of labels and values on the X-axis (around the outside edge of the circular grid) and X-axis gridlines
- chartType: **'radar'**: A radar chart is a circular line chart. Radar charts require one value for each line segment (and marker if shown). The yaxis.majorGrid property draws the circular grid around the risers. yaxis.labels properties control the appearance of labels and values inside the circular grid. Group axis (xaxis) labels draw around the outside edge of the circle. polarProperties can be used to create a radar area chart, extrude axis labels, and draw straight/circular grid lines.
- chartType: **'scatter'**: Scatter charts require two values (X-Position/Y-Position) to draw each marker. X/Y Position values can be a number or a string. To use pie markers on a scatter chart, each marker must be defined by the X/Y-position and the values for each slice to draw in the pie marker. Example:

```
chart.data = [[
//     x     y     pie slices
    [   1,  2,     [1,2,4]],  // pie 0
    [ 2.5,  6.25,  [4,3,4]],  // pie 1
    [3.75,  3.5,   [6,11,9]]  // pie 2
]];
```

  When this data format is used to define a scatter chart, set series: 'all', marker: {shape: 'pie'} to create the pie markers. pieProperties can be applied to the pie markers. You may also create pie markers on bubble/scatter charts using colorMode: byMetric.

- chartType: **'sparkline'**: Sparkline is a microchart that has no titles, labels, or legends. It is a single line chart that is intended to be drawn in a very small area (e.g., height: 20, width: 50). Only the first series of data is plotted. Use the series-specific properties for color / line formatting. Use the yaxis properties to control automatic or manual scale and min / max values. The chart:fill and chart:border properties can be used to format the chart frame. The chart:labelPadding property can be used to define the empty space between the chart border and the line. blaProperties.orientation can be used to draw a horizontal or vertical chart.blaProperties.lineConnection can be used to control the format of the line: linear, curved, stepAfter, stepBefore, or stepBetween.
- chartType: **'stock'**: A high/low/open/close stock chart requires an array of 4 numbers for each riser: [high, low, open, close]. The first two numbers (high/low) define the high/low line that draws up and down from the box. The second two numbers (open/close) draw the open/close box riser. If either high or low is null, the high-low line is not drawn. If either open or close is null, the open-close box is not drawn. Use stockProperties to control the general appearance of a stock chart.
- chartType: **'streamgraph'**: A streamgraph is a simplified version of a stacked area chart. In a streamgraph, there are no axis or gridlines. The baseline is free which makes it easier to perceive the thickness of any given layer across the data. A streamgraph does not use data text labels. The data required to

draw a streamgraph is the same format required to draw an area chart. However, streamgraphs are normally given many series (10 or more), each with many data point points (100 or more). A typical streamgraph would include 20 series with 400 data points in each series. The blaProperties that control the format of an area chart do not work for streamgraphs. You may use series#.visible=true/false to control the visibility of individual series. The xaxis properties that are applicable to an ordinal axis can be used to format labels, title, gridlines and tick marks.

- chartType: **'tagcloud'**: A tagcloud is a visual representation of group labels. The size of each label is proportional to its data value. Labels are defined in the groupLabels property. For example, chart.groupLabels = ['A', 'B', 'C'] and chart.data = [[10, 20, 30]] would draw three labels 'A', 'B', 'C', and the 'C' label would be 3 times larger than 'A'. The tagcloud chart also supports an optional second series of data, which is used to drive the *color* of each label. Colors are chosen from the yaxis.colorScale.colors property. The tagcloudProperties.maxNumberOfTags property can be used to limit the number of labels. The xaxis.min and xaxis.max properties control the minimum/maximum label scaling.

- chartType: **'treemap'**: Treemaps can be defined by a data object or an array of arrays. An object can be nested arbitrarily deep. Any properties in the data object with numeric values will be drawn as a single rectangle in the treemap. For an array definition, each array in the top level array creates one new node in the tree (either a parent or a leaf). The internal arrays should have three entries:

```
[nodeName (string), parentName (string or undefined), value
(number or undefined)]
```

The nodeName is required. If parentName is undefined, this node is treated as a root node. If value is a number, this node is a leaf, otherwise this node is a parent node. Example:

```
data: [
   ['leaf_1', undefined, 5],
   ['child_1', undefined, undefined],
   ['child_Leaf_1', 'child_1', 15]
]
```

treemapProperties can be used to defined the format of header cells and outer cell borders in treemap charts. Cells in a treemap can be colored by series.color or by the colors defined in yaxis.colorScale.colors.

- chartType: **'waterfall'**: Waterfall charts graphically illustrate the cumulative effect of sequentially introducing positive or negative values to an initial value. The initial and final (or total) values are represented by whole columns drawn from the axis baseline. Intermediate positive and negative values are drawn as floating columns. Use the properties in the waterfallProperties group to define the general layout of a waterfall chart.

Chart-Wide Properties

These control the general appearance of a chart. This code segment shows the default settings defined in properties.js:

```
catchErrors: true,
backend: 'js',
allowBackendFallback: false,
width: 400,
height: 250,
fill: { color: 'white'},
border: { width: 0, color: 'transparent', dash: '' },
chartFrame: {
   fill: {color: 'transparent'},
   border: {width: 0, color: 'transparent', dash: ''},
   shadow: false
},
data: [
         [0.6, 0.8, 0.9],[1.0, 0.2, 0.3],
         [0.65, 0.7, 0.25],[0.3, 0.3, 1.0]],
dataSubset: {
   startGroup: undefined,stopGroup: undefined
},
swapData: false,
swapDataAndLabels: false,
riserCycleEndLightness: 0.8,
labelPadding: {
     frame: {left: 5, right: 5, top: 5, bottom: 5},
     label:{left: 5, right: 5, top: 5, bottom: 5}
},
chartsPerRow: undefined,
depth: undefined,
riserDepthGap: 0.2,
riserShadow: false,
riserBevel: undefined,
axisAutoLayout: {
   rotate45: true,rotate90: true,truncate: true,stagger: true,
   skip: true
},
dataLabels: {
   visible: true,
   displayMode: 'x',
   position: 'top',
   font: '7.5pt Sans-Serif',
   color: 'black',
   numberFormat: 'auto',
   formatCallback: undefined
   },
groupLabels: "ABCDEFGHIJKLMNOPQRSTUVWXYZ".split('')
```

allowBackendFallback / backend

These properties define a rendering technology (Java Script or Flash) and enable/disable generation of an alternate output format based on browser capabilities.

```
backend: 'string'
allowBackendFallback: boolean
```

backend a string that specifies the desired output format: 'js' (default) or 'flash'.

allowBackendFallback true/false enables/disables the ability to change backends based on browser capabilities. If false, the chosen *backend* will ***always*** be generated. If true, and the chosen *backend* is 'js', but the browser does not support SVG, and it does support Flash, Flash will be generated. If true, and the chosen backend is 'flash', but the browser does not have a suitable Flash player, and it does support SVG, SVG will be generated.

NOTES:

These properties MUST be passed to the tdgchart() constructor. Example:

```
var props = {
   backend: 'flash',
   allowBackendFallback: false
};
var chart = new tdgchart(props);
```

axisAutoLayout

These properties control automatic layout of ordinal axis labels.

```
axisAutoLayout: {
   rotate45: boolean,
   rotate90: boolean,
   truncate: boolean,
   stagger: boolean,
   skip: boolean
},
```

PARAMETERS:

rotate45: true/false = allow/do not allow auto-layout to rotate labels 45 degrees to achieve the best automatic layout of ordinal axis labels. The default value is true.

rotate90: true/false = allow/do not allow auto-layout to rotate labels 90 degrees to achieve the best automatic layout of ordinal axis labels. The default value is true.

truncate: true/false = allow/do not allow auto-layout to truncate labels to achieve the best automatic layout of ordinal axis labels. The default value is true.

stagger: true/false = allow/do not allow auto-layout to stagger labels to achieve the best automatic layout of ordinal axis labels. The default value is true.

skip: true/false = allow/do not allow auto-layout to skip labels to achieve the best automatic layout of ordinal axis labels. The default value is true.

border

This property defines the border of the drawing area.

```
border: {
   width: Number,
   color: 'string' | JSON Object,
   dash: 'string'
}
```

PARAMETERS:

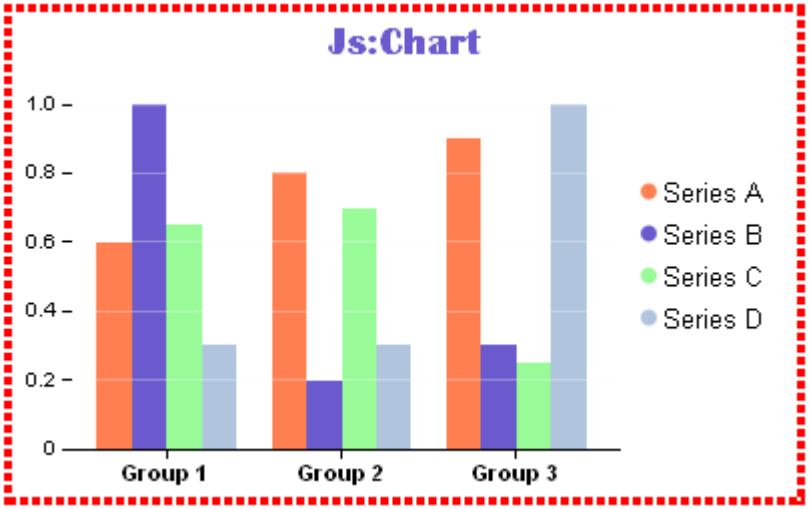
width: a number that defines the width of the border around the drawing area. The default value is zero (no border).

color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is transparent. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

dash: a string that defines the border's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

EXAMPLE:

```
border: {
         width: 4,
         color: 'red',
         dash: '4 2'
}
```



catchErrors

This property defines how Java Script errors are handled. Typical errors can include invalid property settings or misplacement or missing commas and brackets that do not allow the library to parse the Java Script. When catchErrors is false, you can use the Java Script console (e.g., in Chrome Control+Shift+j) to determine the exact location of the error.

```
catchErrors: boolean
```

PARAMETERS:

true: errors are caught inside the chart library and displayed as HTML text where the chart would otherwise be drawn. Subsequent JavaScript code in the calling application continues to run.

false: errors are not caught inside the library, but are passed on to the calling application. If the calling application does not catch the error, all subsequent JavaScript execution terminates.

The default value is true.

chartFrame

This property defines the fill color and border of the chart frame.

```
chartFrame: {
   fill: {
         color: 'string' | JSON Object
   },
   border: {
         width: Number,
         color: 'string' | JSON Object,
         dash: 'string'
   },
   shadow: boolean | JSON Object
},
```

PARAMETERS:

fill/color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is transparent. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

border/width: Defines the width of the border around the chart frame. The default value is zero (no border).

border/color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is transparent. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

border/dash: A string that defines the border's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

shadow: true/false enables/disables a default shadow on the chart frame. The default value is false. You may also use a JSON object in the following format to define the shadow:

```
shadow:
{
   angle: Number,
   distance: Number,
   blur: Number,
}
```

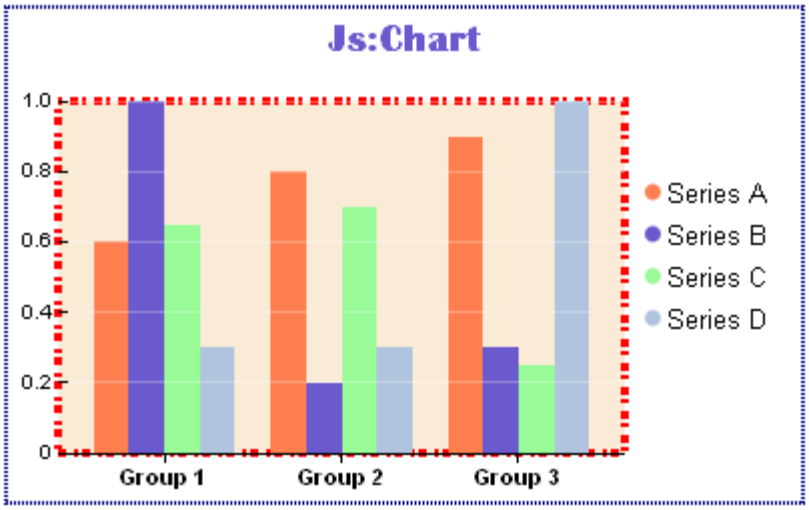
shadow/angle: a number in the range 0...360 defines the position of the light source. 0 = 12 o'clock, 90 = 3 o'clock. The default value is 315.

shadow/distance: a number defines the distance to push the shadow away from parent shape, in the global coordinate space (pixels by default). The default value is 10.

shadow/blur: a number in the range 0...1 defines the hardness of the shadow. 0 = perfectly hard edge, 1 = perfectly soft edge. The default value is zero.

EXAMPLE:

```
chartFrame: {
   fill: {color: 'antiquewhite'},
   border: {
         width: 4,
         color: 'red',
         dash: '2 4 4 2'
   }
},
```



chartsPerRow

For pies, gauges, and funnels that can have multiple charts in a single draw area, this property defines how many charts to draw in a horizontal row.

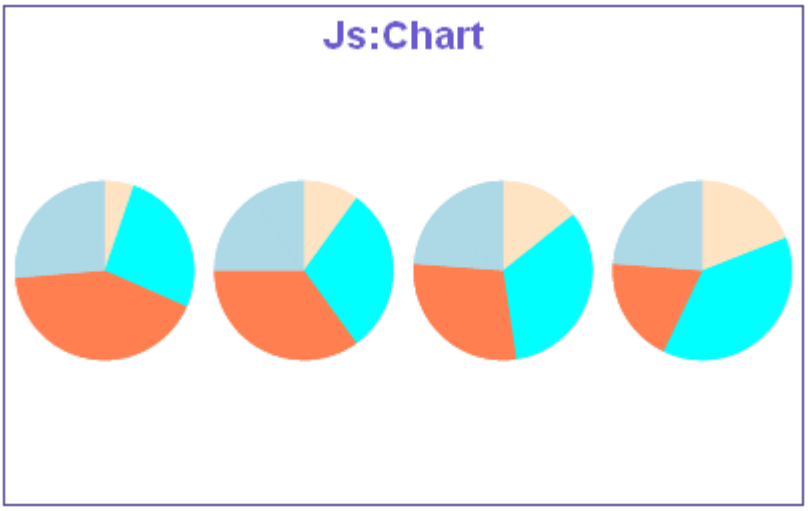
```
chartsPerRow: Number
```

PARAMETERS:

chartsPerRow: number of charts to draw in a horizontal row. The default value is undefined.

EXAMPLE:

```
chartType: 'pie',
chartsPerRow: 4
```



data

This property defines the values that draw a chart.

```
data: [[value, value, ... value]]
```

data: one or more arrays of values. The number of values and the number of arrays depend on the chart type. See the chartType property for details. The default value is:

```
data: [
        [0.6, 0.8, 0.9],
        [1.0, 0.2, 0.3],
        [0.65, 0.7, 0.25],
        [0.3, 0.3, 1.0]
],
```

These default values will draw a simple bar, line, or area chart. Each value represents a data point in the chart.

NOTES:

- For some chart types such as treemaps, a JSON object may be used to define data points.
- For bubble/scatter charts, X/Y position values can be defined as strings (e.g., data: [[['-A-', '-A-', 10]],[['-B-','-B-',20]],[['-C-','-C-',30]],[['-D-','-D-',100]]]).
- This property is affected by the swapData and swapDataAndLabels properties. When these properties are false (the default), each value in an array is presented as riser/marker in a series. When these properties are set to true, each value in an array is presented as a riser/marker in a group.
- In all cases, you can use null or undefined to identify a missing data point. See Data & Property Definitions for more information.

dataLabels

These properties control the visibility and format of data text labels for all series. Use the Series-Specific properties to enable/disable data text labels for individual series.

```
dataLabels: {
   visible: true,
   displayMode: 'string',
   position: 'string',
   font: 'string',
   color: 'string',
   numberFormat: JSON Object | 'string' | function(),
   formatCallback: function()
}
```

PARAMETERS:

visible; true/false controls the visibility of the data text labels. The default value is true. When visible is true, use the series showDataValues property to hide data labels for individual series.

displayMode: controls the data text to show: 'x', 'y', 'z', '%', '%+', 'cumulative', 'groupLabel', or 'seriesLabel'. The default value is 'x'.

- For a bubble chart, use 'x', 'y', or 'z' to identify which value to show (X-Position, Y-Position, or Size (z)).
- For a line chart where the data set includes marker size values, use 'x' or 'y' to show the Y-axis value. Use 'z' to show the marker size values.
- For a scatter chart, use 'x' or 'y' to identify which value to show (X-Position, Y-Position).
- For a pie chart, use '%' to show the percentage value of a pie slice.
- For stacked bar, line, or area charts (blaProperties.seriesLayout='stacked'), use 'cumulative' to show the stack total, '%' to show the stack value as a percentage, or '%+' to show percentage and cumulative values.

For % mode in pie charts, you must set the numberFormat property to display the value as a percentage (e.g.,numberFormat: '[>9]#,#%;[<0]-#.#%;[<=9]#.##%;[=0]0').

You can use the formatCallback property and define a function to display multiple values. See the example below under the formatCallback property.

position: a string defines the position of data labels relative to the risers: 'bottom', 'center', 'insideBottom', 'insideTop', 'left', 'outside', 'right', 'top'. The default value is 'top'.

- The 'insideBottom' and 'insideTop' settings are for bar risers only and draw the data labels inside the bottom or top of the riser.
- For pie charts, the only valid settings are 'center' (on pie slices) and 'outside' (outside pie slices with feeler lines).
- For bar risers, positions are orientation aware (e.g., 'top' = 'right' in a horizontal bar).
- For bullet charts, also see the series-specific marker position property that defines the position of the marker relative to data text.

font; a string that defines the size and type face of data text labels. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is font: '7.5pt Sans-Serif'.

color; a string that defines the color of data text labels. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

numberFormat: Use a string, JSON object, or function to define the format of data labels. See Properties Overview/Formatting Numbers in Section 1 for details and examples. The default value is 'auto'.

formatCallback: Identify a function to call for each data label. The default value is undefined. This function can use up to three arguments (data value, series ID, and group ID) and should return a string to be displayed.

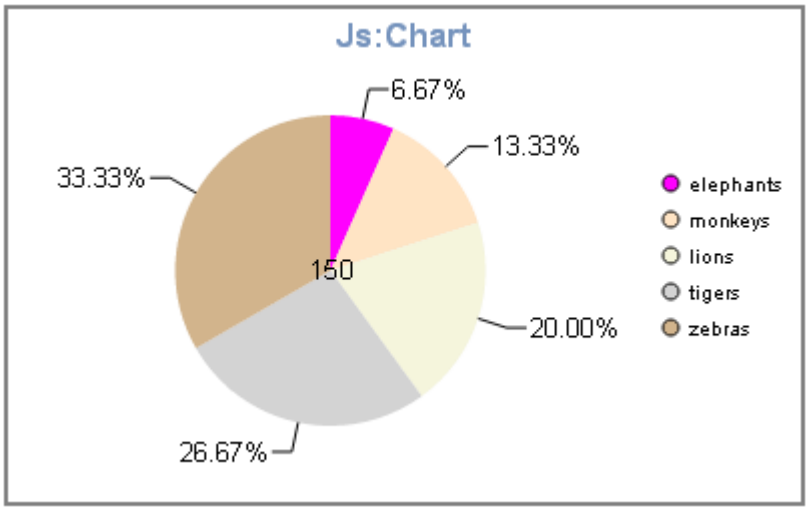
```
function myCallback(dataValue, seriesID, groupID);
```

As an example, you can use a user-defined function to show x, y, and z values in a bubble chart:

```
chart.dataLabels.formatCallback = function(d){
    return d[0] + ', ' + d[1] + ', ' + d[2];
}
```

EXAMPLES:

```
data: [[10,20,30,40,50]],
chartType: 'pie',
legend: {visible: true},
pieProperties: {totalLabel: {visible: true}},
dataLabels: {
   displayMode: '%',
   position: 'outside',
   font: '10pt Sans-Serif',
   numberFormat: '[>9]#,#%;[<0]-#.#%;[<=9]#.##%;[=0]0'
   },
series: [
   {series: 0, color: 'magenta',label: 'elephants'},
   {series: 1, color: 'bisque', label: 'monkeys'},
   {series: 2, color: 'beige', label: 'lions'},
   {series: 3, color: 'lightgrey', label: 'tigers'},
   {series: 4, color: 'tan', label: 'zebras'}
]
```



dataSubset

These properties define the range of data to draw in the chart. It can be used with the interaction properties (e.g., mouseDrag: pan) to define the range of data that is visible following a mouseDrag interaction.

```
dataSubset: {
   startGroup: number,
   stopGroup: number,
}
```

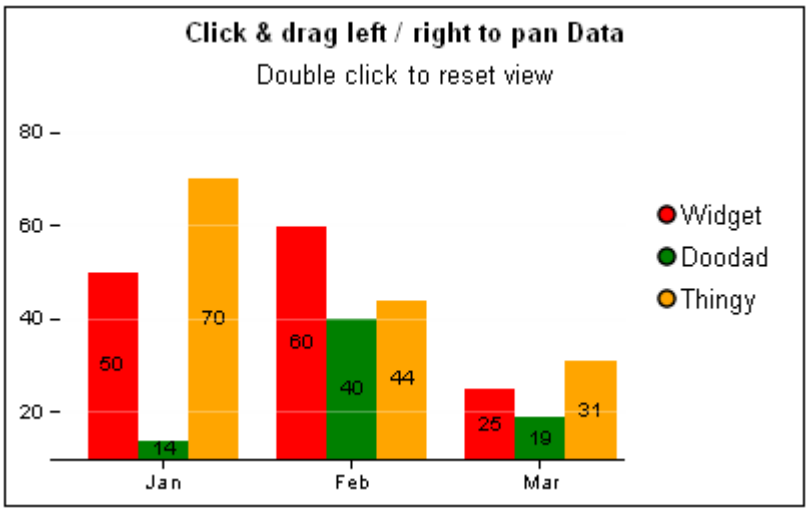
PARAMETERS:

startGroup: zero-based starting group number to show. The default value is undefined.

stopGroup: zero-based number of groups to show. The default value is undefined.

EXAMPLE:

```
chart.interaction.mousedrag = 'pan';
chart.data = [
   [50, 60, 25, 20, 30, 17, 44, 31, 19, 78, 50, 11],
   [14, 40, 19, 60, 38, 44, 23, 21, 30, 55, 77, 47],
   [70, 44, 31, 19, 33, 53, 10, 30, 40, 50, 22, 11]
];
chart.dataSubset.startGroup = 0;
chart.dataSubset.stopGroup = 3;
chart.groupLabels = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
```



depth

This property applies a 2.5D depth effect to bar charts, line charts, area charts, pie charts, and any 'bar-like' chart (gantt, histogram, etc.).

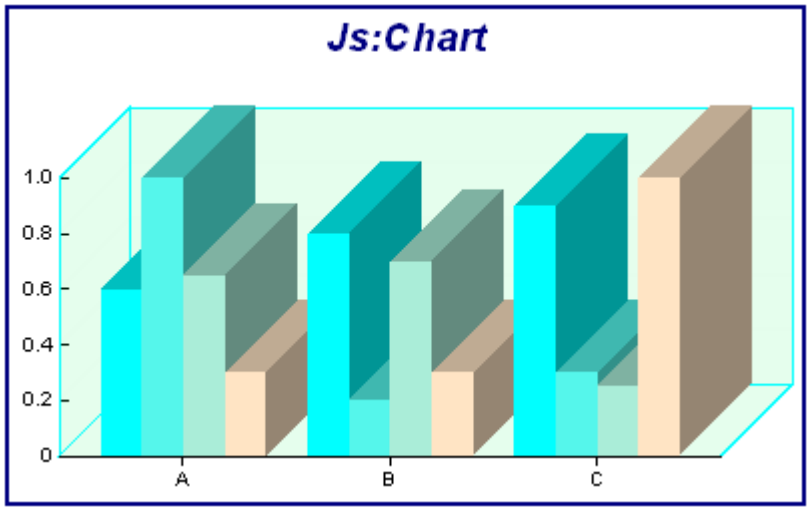
```
depth: Number
```

PARAMETERS:

depth: a number 0...100 that specifies the amount of depth to apply to risers and the chart frame. Use zero or undefined for no 2.5D depth effect. The default value is undefined.

EXAMPLE:

```
chartFrame: {border: {width: 1, color: 'cyan'}, fill: {color:
'hsla(140, 100%, 50%, 0.1)'}},
colorMode: {
   mode: 'byInterpolation',
   colorList: ['cyan', 'bisque']
},
blaProperties: {orientation: 'vertical'}
depth: 50,
```



fill

This property defines a color or gradient to be applied to the draw area.

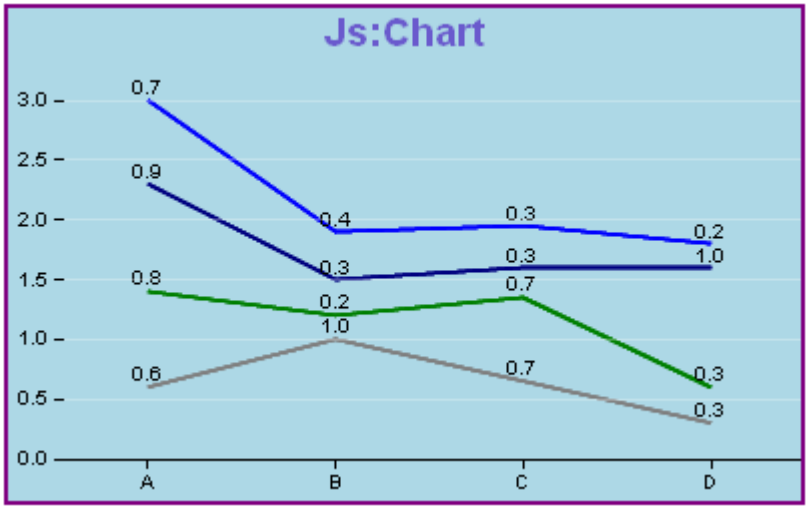
```
fill: {
   color: 'string' | JSON Object
}
```

PARAMETERS:

color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is white. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

EXAMPLE:

```
fill: {
         color: 'LightBlue'
         },
border: {
         width: 4,
         color: 'purple'
}
```



groupLabels

For all chart types except histogram and tagcloud, this property defines the chart's group labels that will appear on the ordinal (xaxis) axis. For a tagcloud chart, this property defines the labels that appear in the chart according to the data values. For all chart types except tagcloud, group labels can be defined as a string, an array of strings, or as a JSON object to produce nested labels. To define nested group labels, simply provide group labels in a hierarchical format and the nested labels will be automatically used. The chart frame will be automatically adjusted to leave the proper space for the axis. Nested labels are implemented only for left and bottom axis. The nodes are nested in the JSON structure with the leaves stored in an array. For a tagcloud chart, group labels must be defined as an array of strings.

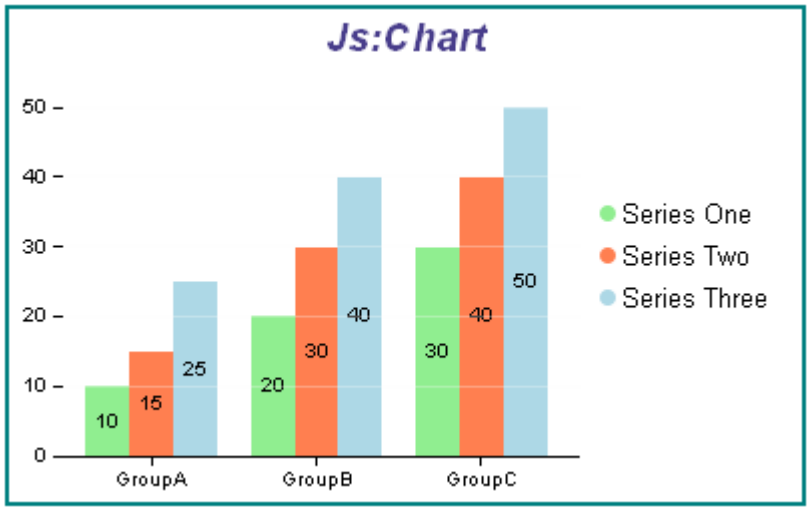
```
groupLabels: 'string' | ['string'...'string'] | JSON object
```

PARAMETERS:

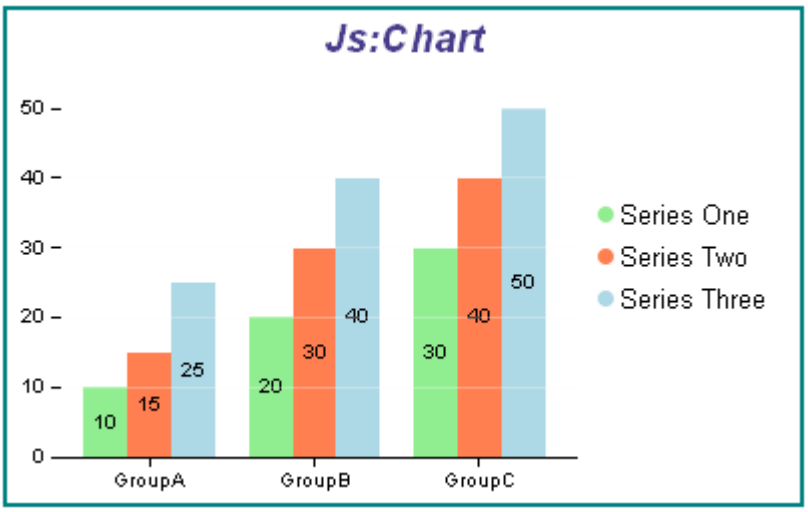
groupLabels: a string, an array of strings, or a JSON object. A string will be split into an array along any space character. The default value is "ABCDEFGHIJKLMNOPQRSTUVWXYZ".split('').

EXAMPLES:

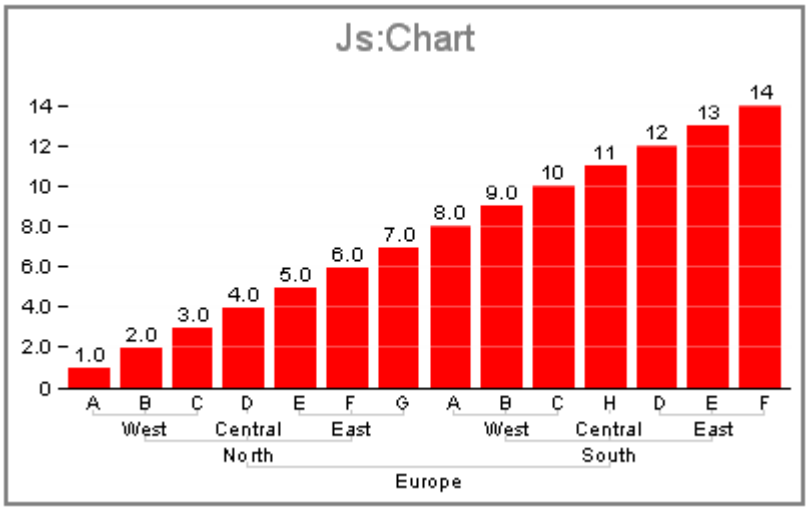
```
legend: {visible: true},
blaProperties: {orientation: 'vertical'},
groupLabels: "GroupA GroupB GroupC",
series: [
   {series: 0, label: 'Series One', color: 'lightgreen'},
   {series: 1, label: 'Series Two', color: 'coral'},
   {series: 2, label: 'Series Three', color: 'lightblue'},
   {series: 3, label: 'Series Four', color: 'burlywood'},
]
```




```
legend: {visible: true},
blaProperties: {orientation: 'vertical'},
groupLabels: ['Group A', 'Group B', 'Group C'],
series: [
   {series: 0, label: 'Series One', color: 'lightgreen'},
   {series: 1, label: 'Series Two', color: 'coral'},
   {series: 2, label: 'Series Three', color: 'lightblue'},
   {series: 3, label: 'Series Four', color: 'burlywood'},
   ]
```

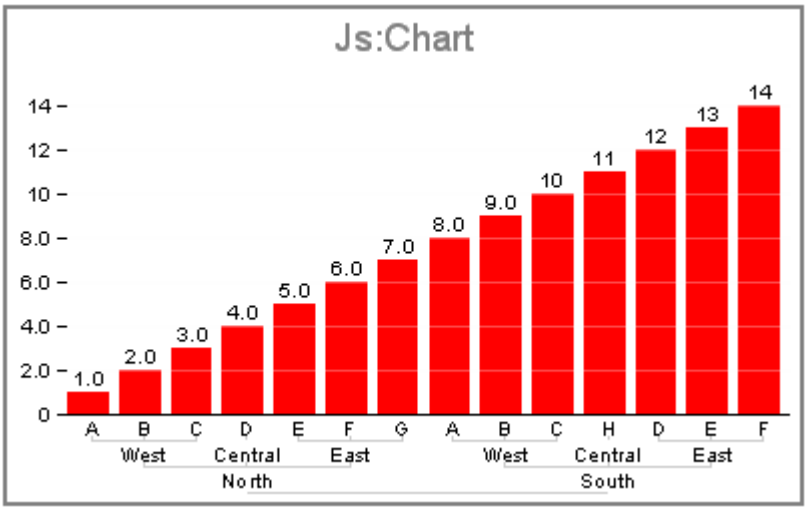



```
chart.groupLabels = {
   Europe: {
    North: {
      West: [ "A", "B", "C" ],
      Central: [ "D" ],
      East: [ "E", "F", "G" ]
    },
    South: {
      West: [ "A", "B", "C" ],
      Central: [ "H" ],
      East: [ "D", "E", "F" ],
    }
   }
 };
```




```
chart.groupLabels = {
    North: {
      West: [ "A", "B", "C" ],
      Central: [ "D" ],
      East: [ "E", "F", "G" ]
    },
    South: {
      West: [ "A", "B", "C" ],
      Central: [ "H" ],
      East: [ "D", "E", "F" ],
```

```
    }
};
```



NOTES:

- For all chart types except tagcloud, use xaxis:labels to format group labels.
- You can define date/time labels in an array and enable xaxis:timeAxis to create a time-based axis. See the timeAxis property for more details and examples. The library does not support date/time labels defined in a string or nested labels defined in a JSON object.

height / width

These properties define the width and height (in pixels) of the chart draw area.

```
width: number
height: number
```

PARAMETERS:

width: defines the width of the chart draw area (in pixels). The default value is 400.

height: defines the height of the chart draw area (in pixels). The default value is 250.

labelPadding

These properties control the amount of space to leave around the chart frame and labels (chart title, subtitle, footnote, and axis titles).

```
labelPadding: {
 frame: {left: Number, right: Number, top: Number, bottom: Number},
 label:{left: Number, right: Number, top: Number, bottom: Number}
}
```

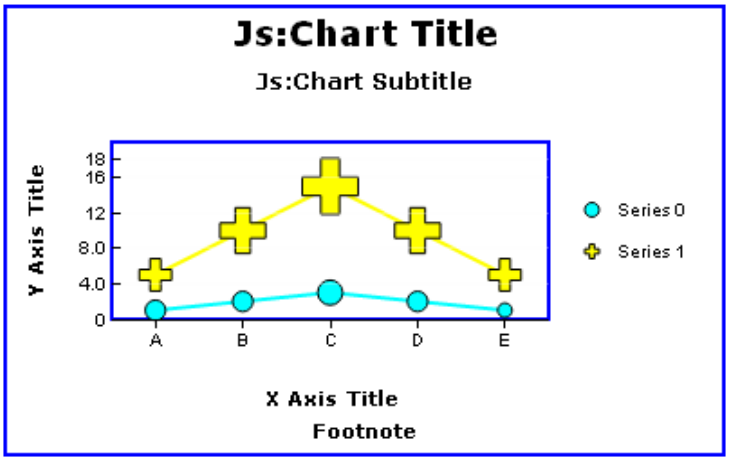
PARAMETERS:

frame: left/right/top/bottom; number of blank pixels around the chart frame. The default value is 5.

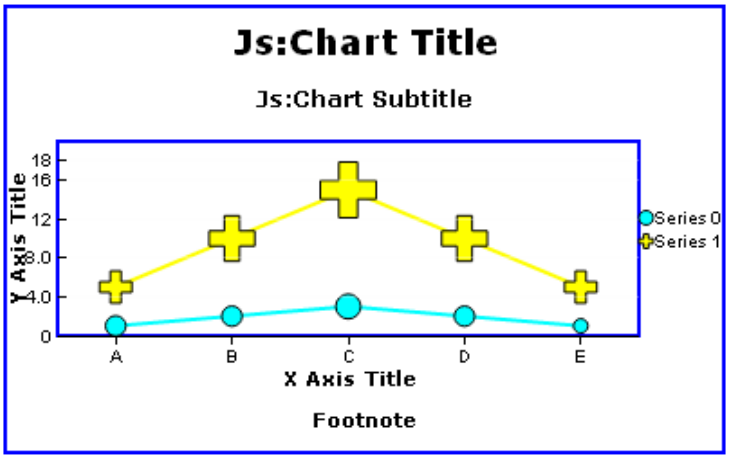
labels: left/right/top/bottom; number of blank pixels around chart labels ((chart title, subtitle, footnote, and axis titles). The default value is 5.

EXAMPLES:

```
labelPadding: {
   frame:{left: 0, right: 0, top: 0, bottom: 0},
   label:{left: 10, right: 10, top: 10, bottom: 10}
   },
```




```
labelPadding: {
   frame:{left: 10, right: 10, top: 10, bottom: 10},
   label:{left: 0, right: 0, top: 0, bottom: 0}
   },
```



riserBevel

This property applies a bevel to risers in a bar chart, markers in bubble/scatter charts, and slices in a pie chart.

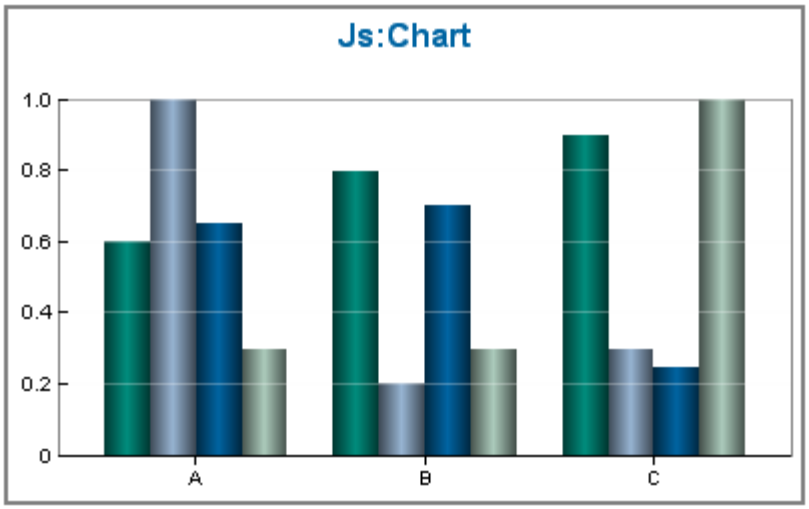
```
riserBevel: bevel: 'string'
```

PARAMETERS:

bevel: a string that defines the bevel type. For bar charts, use one of: bevel, cylinder, darken, darkenInverted, lighten, or lightenInverted. For bubble/scatter charts, use darken or lighten. For pie charts, use one of: bevel, cylinder, or donut. Bevel types darken and lighten will also be applied to legend markers (if visible). The default value is undefined.

EXAMPLES:

```
riserBevel: 'cylinder'
```



riserCycleEndLightness

If there are more series than series colors defined, the charting library cycles through the defined colors. This property defines the lightness / darkness of the final riser color cycle.

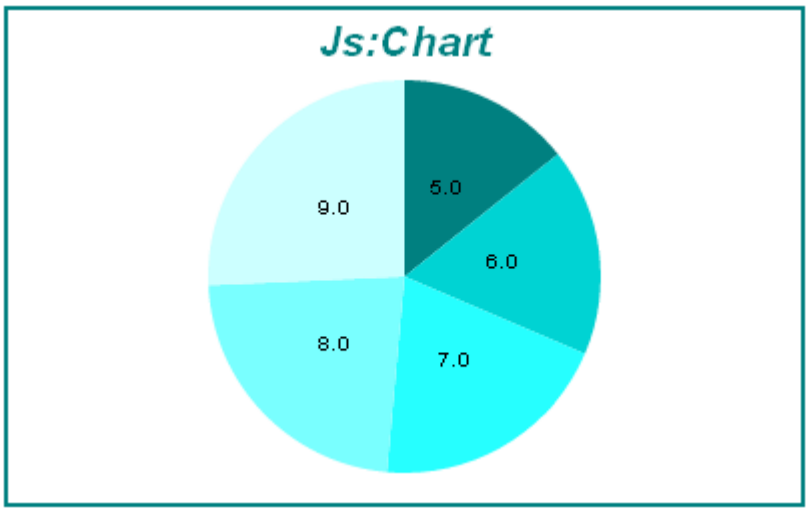
```
riserCycleEndLightness: Number
```

PARAMETERS:

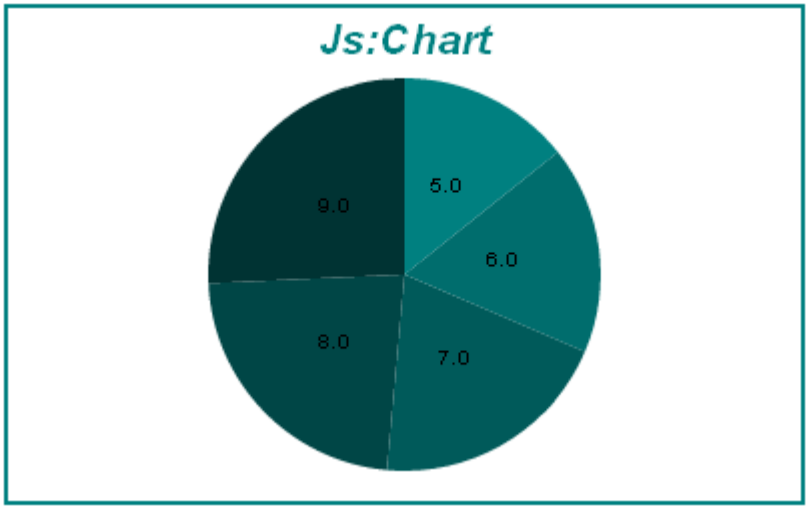
riserCycleEndLightness; a number in the range 0...1 controls the brightness of colors assigned to undefined series risers. Values less 0.5 darkens each series cycle. Values greater than 0.5 lightens each series cycle. The default value is 0.8.

EXAMPLES:

```
data: [[5,6,7,8,9]],
riserCycleEndLightness: 0.9,
chartType: 'pie',
series: [{series: 0, color: 'teal'}]
```




```
riserCycleEndLightness: 0.1,
```



riserDepthGap

In 3D chart and charts where 2.5D depth is applied to a chart with the depth property, this property adds space between risers that draw behind other risers.

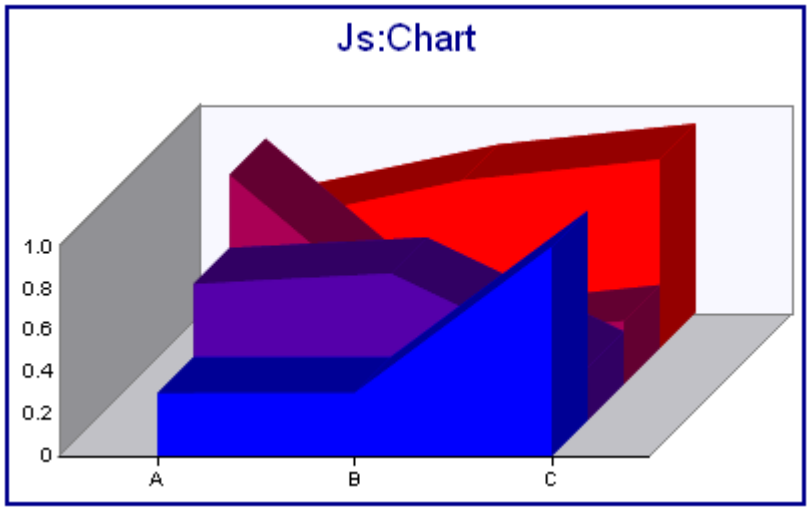
```
riserDepthGap: number
```

PARAMETERS:

riserDepthGap: a number in the range 0...1 (as a factor of the depth) that defines the margin between the risers. The default value is 0.2.

EXAMPLE:

```
chartType: 'area'
depth: 25,
riserDepthGap: 0,
chartFrame: {border: {width: 1, color: 'grey'}, fill: {color:
'ghostwhite'}},
dataLabels: {visible: false},
colorMode: {
   mode: 'byInterpolation',
   colorList: ['red', 'blue']
},
blaProperties: {orientation: 'vertical'},
```



riserShadow

This property applies a shadow to chart risers/markers.

```
riserShadow: boolean | JSON object
```

PARAMETERS:

riserShadow: true/false enables/disables a default shadow on chart risers. The default value is false. You can also use a JSON object in the following format to define the shadow:

```
riserShadow:
{
   angle: Number,
   distance: Number,
   blur: Number,
}
```

riserShadow/angle: a number in the range 0...360 defines the position of the light source. 0 = 12 o'clock, 90 = 3 o'clock. The default value is 315.

riserShadow/distance: a number defines the distance to push the shadow away from parent shape, in the global coordinate space (pixels by default). The default value is 10.

riserShadow/blur: a number in the range 0...1 defines the hardness of the shadow. 0 = perfectly hard edge, 1 = perfectly soft edge. The default value is zero.

NOTES:

Other chart objects can be assigned shadows. See chartFrame: shadow and legend: shadow.

swapData

This property can be used to swap series and group orientation. When false (the default), values in a row in a data set are represented as series. When true, values in a column are represented as series.

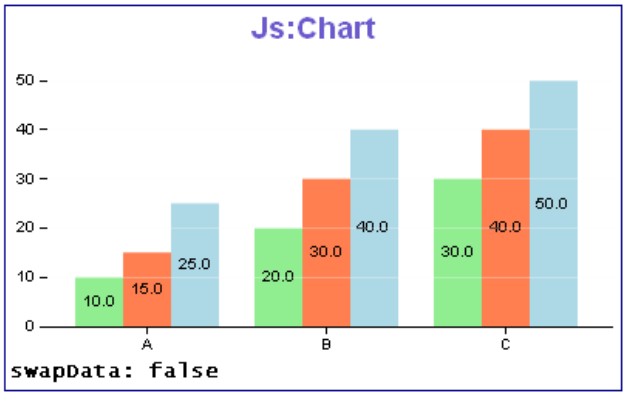
```
swapData: boolean
```

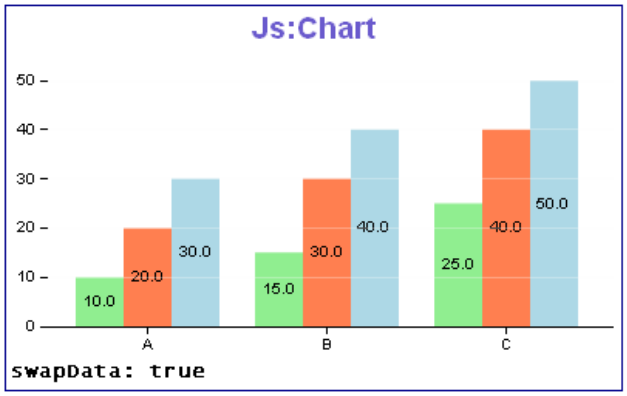
PARAMETERS:

swapData: true = swap series/group orientation, false (default) = do not swap series/group orientation

EXAMPLE:

```
data: [
          [10, 20, 30],
          [15, 30, 40],
          [25, 40, 50]
]
```





swapDataAndLabels

This property can be used to swap series and group orientation. It is the same as the swapData property except it also swaps the series and group labels. When false (the default), values in a row in a data set are represented as series. When true, values in a column are represented as series.

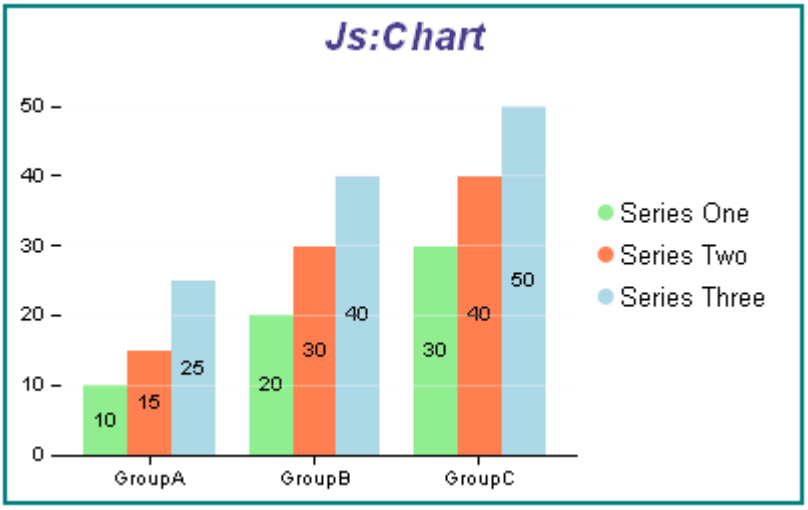
```
swapDataAndLabels: boolean
```

PARAMETERS:

swapData: true = swap series/group/ label orientation, false (default) = do not swap series/group/label orientation

EXAMPLES:

```
data: [  [10, 20, 30],
         [15, 30, 40],
         [25, 40, 50]],
swapDataAndLabels: false
```




```
data: [  [10, 20, 30],
         [15, 30, 40],
         [25, 40, 50]],
swapDataAndLabels: true
```

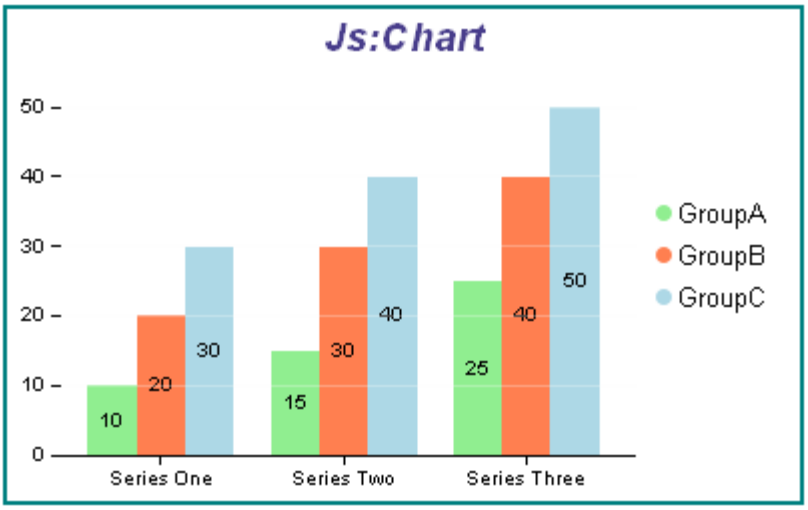


Chart Titles Properties (title, subtitle, footnote)

These properties control the format and visibility of the chart title, subtitle, and footnote. The following code segment shows the default values from the properties.js file:

```
title: {
      text: 'Chart Title',
      visible: true,
      align: 'center',  // One of 'left', 'center', 'right'
      font: 'bold 10pt Sans-Serif',
      color: 'black'
      tooltip: undefined
},
subtitle: {
      text: 'Chart Subtitle',
      visible: false,
      align: 'center',
      font: '10pt Sans-Serif',
      color: 'black'
      tooltip: undefined
},
footnote: {
      text: 'Chart Footnote',
      visible: false,
      align: 'right',
      font: '10pt Sans-Serif',
      color: 'black'
      tooltip: undefined
},
```

ALSO SEE:

legend: title

xaxis: title

xaxis: title

yaxis: title

y2axis: title

zaxis: title

title

These properties control the content, visibility, and format of the Chart Title.

```
title: {
   text: 'string',
   visible: boolean,
   align: 'string',
   font: 'string',
   color: 'string'
   tooltip: 'string' | function()
}
```

PARAMETERS:

text; a string that defines the Chart Title text. The default value is 'Chart Title'.

visible; true/false controls the visibility the Chart Title. The default value is true (visible).

align; a string that defines the alignment of the Chart Title: 'center', 'chartFrame' (center aligned with the chart frame), 'left', or 'right'. The default value is 'center'.

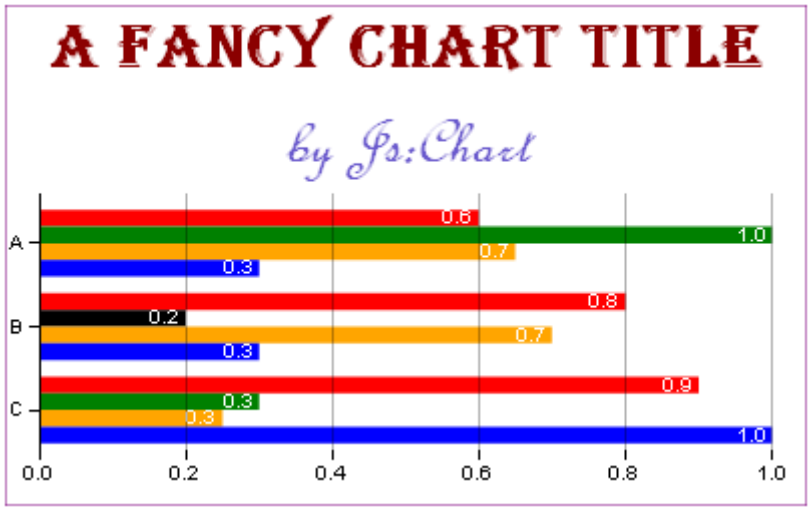
font; a string that defines the size and type face of the Chart Title. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is 'bold 10pt Sans-Serif'.

color; a string that defines the color of the Chart Title. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

tooltip: a string or a function that returns a string to show when the mouse hovers over the title. The default value is undefined. A callback function is invoked with three arguments: value, series ID, and group ID. For non-riser objects, set the arguments to undefined. 'this' inside the callback function is set to the parent chart and provides full access to the property tree and API.

EXAMPLE:

```
title: {
   text: 'A FANCY CHART TITLE',align: 'center',
   font: 'Bold 24pt Algerian',color: 'darkred'
}
```



subtitle

These properties control the content, visibility, and format of the Chart Subtitle.

```
subtitle: {
   text: 'string',
   visible: boolean,
   align: 'string',
   font: 'string',
   color: 'string'
   tooltip: 'string' | function()
}
```

PARAMETERS:

text; a string that defines the Chart Subtitle text. The default value is 'Chart Subtitle'.

visible; true/false controls the visibility the Chart Subtitle. The default value is false.

align; a string that defines the alignment of the Chart Subtitle: 'center', 'chartFrame' (center aligned with the chart frame), 'left', or 'right'. The default value is 'center'.

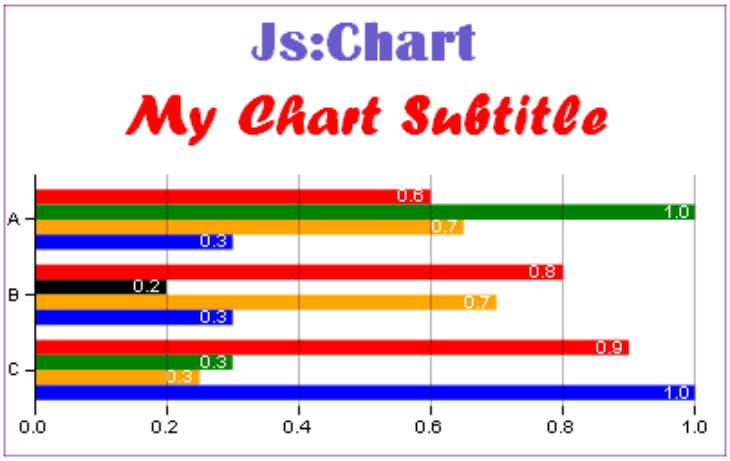
font; a string that defines the size and type face of the Chart Subtitle. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color; a string that defines the color of the Chart Subtitle. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is black.

tooltip: a string or a function that returns a string to show when the mouse hovers over the subtitle. The default value is undefined. A callback function is invoked with three arguments: value, series ID, and group ID. For non-riser objects, set the arguments to undefined. 'this' inside the callback function is set to the parent chart and provides full access to the property tree and API.

EXAMPLE:

```
subtitle: {
   text: 'My Chart Subtitle',visible: true,align: 'center',
   font: 'Bold 24pt Forte',color: 'red'
}
```



footnote

These properties control the content, visibility, and format of the Chart Footnote.

```
footnote: {
   text: 'string',
   visible: boolean,
   align: 'string',
   font: 'string',
   color: 'string'
   tooltip: 'string' | function()
}
```

PARAMETERS:

text; a string that defines the Chart Footnote text. The default value is 'Chart Footnote'.

visible; true/false controls the visibility the Chart Footnote. The default value is false.

align; a string that defines the alignment of the Chart Footnote: 'center', 'chartFrame' (center aligned with the chart frame), 'left', or 'right'. The default value is 'center'.

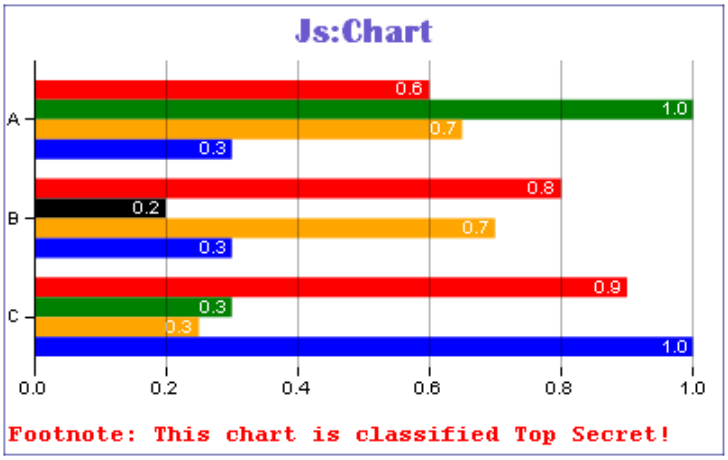
font; a string that defines the size and type face of the Chart Footnote. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color; a string that defines the color of the Chart Footnote. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

tooltip: a string or a function that returns a string to show when the mouse hovers over the footnote. A callback function is invoked with three arguments: value, series ID, and group ID. For non-riser objects, set the arguments to undefined. 'this' inside the callback function is set to the parent chart and provides full access to the property tree and API. The default value is undefined.

EXAMPLE:

```
footnote: {
   text: 'Footnote: This chart is classified Top Secret!',
   font: 'Bold 10pt Courier',visible: true,align: 'left',color: 'red'
}
```



Legend Properties

These properties control the visibility and format of the legend. This code segment shows the default values from properties.js:

```
legend: {
   visible: false,
   position: 'right',  // One of 'free', 'left', 'right', 'bottom'
   xy: {x:330, y:80},
   markerSize: 8,
   markerPosition: 'left',  // One of 'left', 'right', 'bottom'
   maxEntries: undefined,
   title: {
         visible: false,
         text: 'Legend Title',
         font: '10pt Sans-Serif',
         color: 'black'
   },
   labels: {
         font: '7.5pt Sans-Serif',
         color: 'black'
   },
   lineStyle: {
         width: 0,
         color: 'black',
         dash: ''
   },
   backgroundcolor: 'transparent',
   shadow: false
}
```

backgroundcolor

This property controls the background color of the legend area.

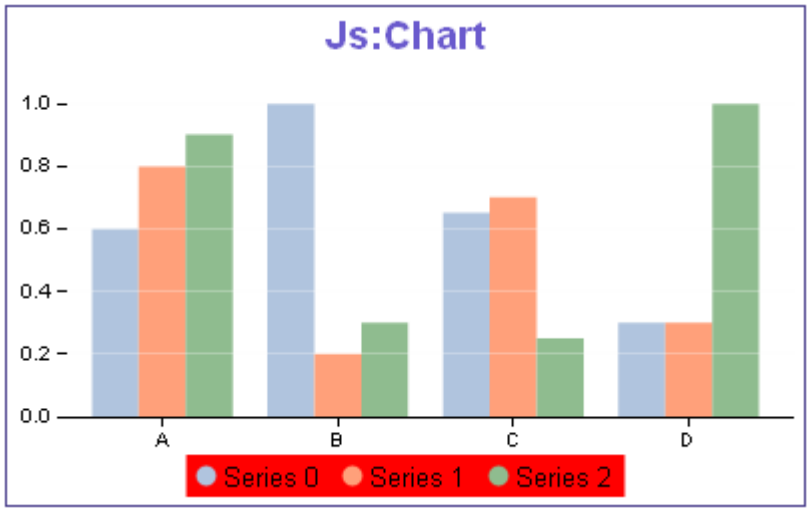
```
legend: {
   backgroundcolor: 'string' | JSON Object
}
```

PARAMETERS:

color; a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is transparent. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

EXAMPLE:

```
legend: {
   visible: true,
   position: 'bottom',
   backgroundcolor: 'red'
}
```



labels

This property controls the format of legend labels.

```
legend: {
   labels: {
         font: 'string',
         color: 'string'
         }
}
```

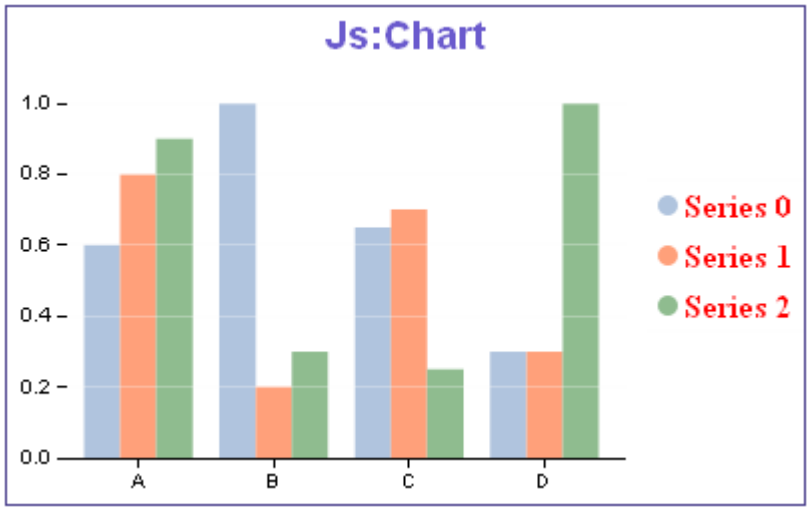
PARAMETERS:

font; a string that defines the size and type face of the Legend Labels. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '7.5pt Sans-Serif'.

color; a string that defines the color of the Legend Labels. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

EXAMPLE:

```
legend: {
   visible: true,
   labels: {
         font: 'Bold 12pt Times Roman',
         color: 'red'
   }
}
```



lineStyle

This property controls the width and color of a line that can be drawn around the legend area.

```
lineStyle: {
   width: number,
   color: 'string' | JSON Object,
   dash: 'string'
}
```

PARAMETERS:

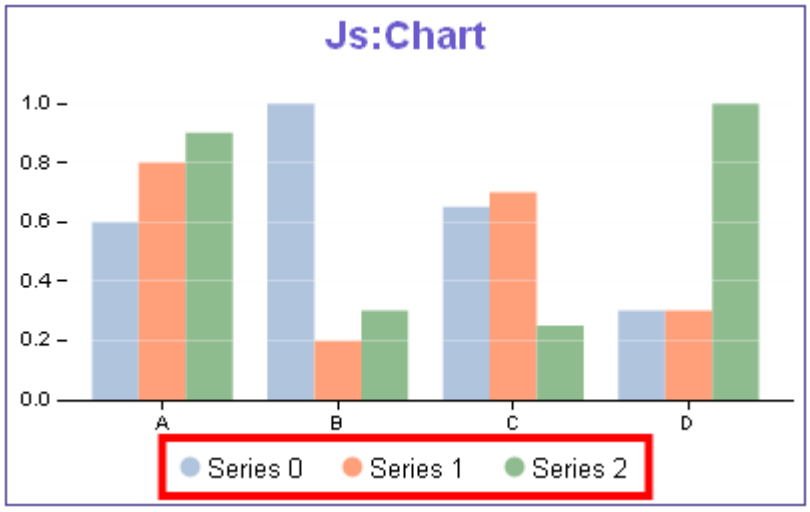
width; a number that defines the width of the line (Default = 0, no line)

color; a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is black. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

dash: A string that defines the border's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes.

EXAMPLE:

```
legend: {
   visible: true,
   position: 'bottom',
   lineStyle": {'width': 4, 'color': 'red'}
}
```



markerPosition

This property controls the location of the legend markers relative to the legend labels.

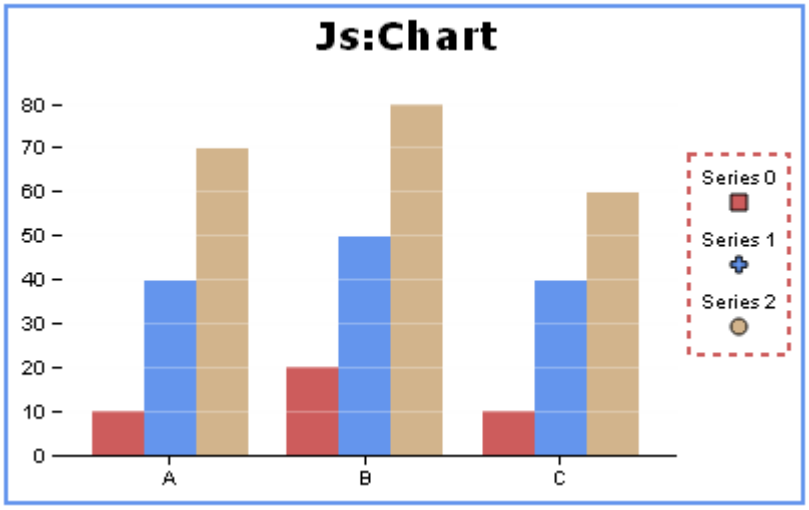
```
legend: {markerPosition: 'string'}
```

PARAMETERS:

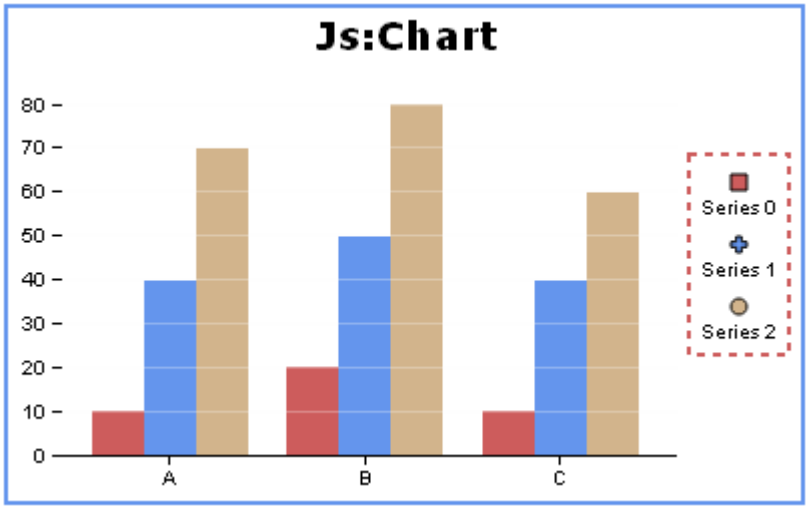
markerPosition: a string that defines the location of legend markers relative to the legend label: 'bottom', 'left', 'right', or 'top'. The default value is 'left'.

EXAMPLE:

```
legend: {visible: true, markerPosition: 'bottom'}
```




```
legend: {visible: true, markerPosition: 'top'}
```



markerSize

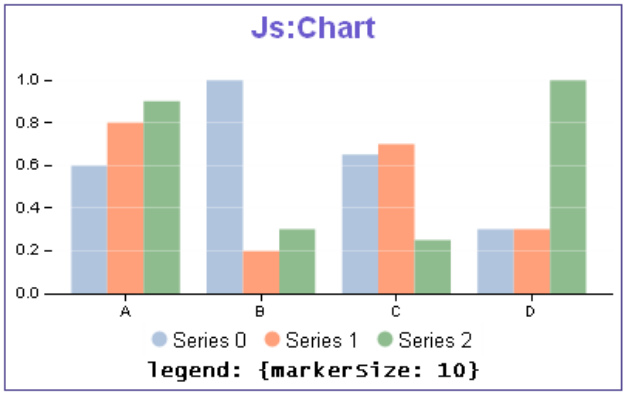
This property controls the size of legend markers.

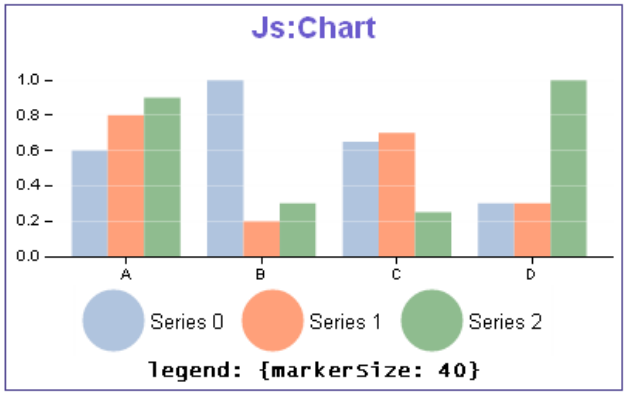
```
legend: {markerSize: Number}
```

PARAMETERS:

markerSize; defines the size of legend markers. Default value=8.

EXAMPLE:





maxEntries

This property controls the maximum number of series labels to draw in the legend area. When *maxEntries* is set to a number greater than zero and less than the number of series defined in the data set, a "more entries" indicator will be shown to indicate missing data in the legend area.

```
legend: {maxEntries: number}
```

PARAMETERS:

maxEntries: maximum number of series labels to draw in the legend. The default value is undefined.

position

This property controls the location of the legend.

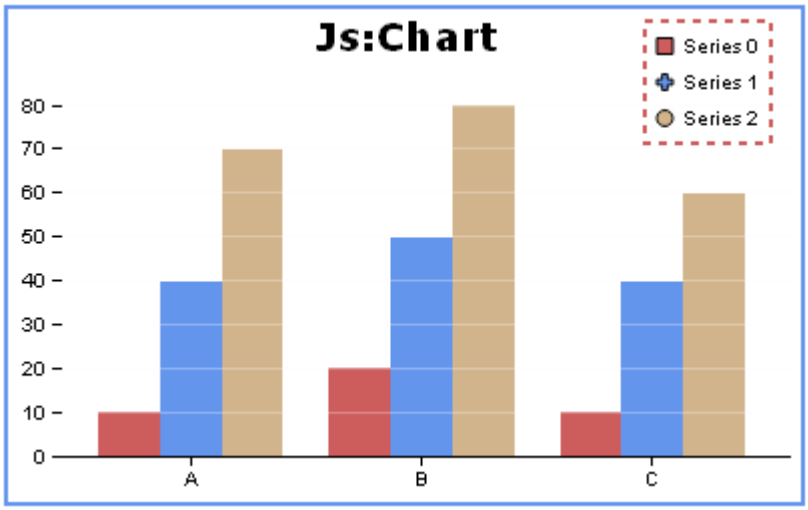
```
legend: {position: 'string'}
```

PARAMETERS:

position: a string that defines the location of the legend: 'bottom', 'free' (floating), 'left', 'right' (default), or 'top'.

EXAMPLE:

```
legend: {
   visible: true,
   position: 'free',
   lineStyle: {width: 2, color: 'indianred', dash: '4 4'},
   xy: {x:320, y:8}
   },
```



shadow

This property applies a shadow to the legend area.

```
legend: {
   shadow: 'boolean' | JSON Object
}
```

PARAMETERS:

shadow: true/false enables/disables a default shadow on the legend area. The default value is false. You can use a JSON object in the following format to define the shadow:

```
shadow:
{
   angle: Number,
   distance: Number,
   blur: Number,
}
```

shadow/angle: a number in the range 0...360 defines the position of the light source. 0 = 12 o'clock, 90 = 3 o'clock. The default value is 315.

shadow/distance: a number defines the distance to push the shadow away from parent shape, in the global coordinate space (pixels by default). The default value is 10.

shadow/blur: a number in the range 0...1 defines the hardness of the shadow. 0 = perfectly hard edge, 1 = perfectly soft edge. The default value is zero.

title

This property controls the visibility and format of the legend title.

```
legend: (
   title: {
         visible: boolean,
         text: 'string',
         font: 'string',
         color: 'string'
   }
}
```

PARAMETERS:

visible; true/false controls the visibility of the Legend Title. The default value is false.

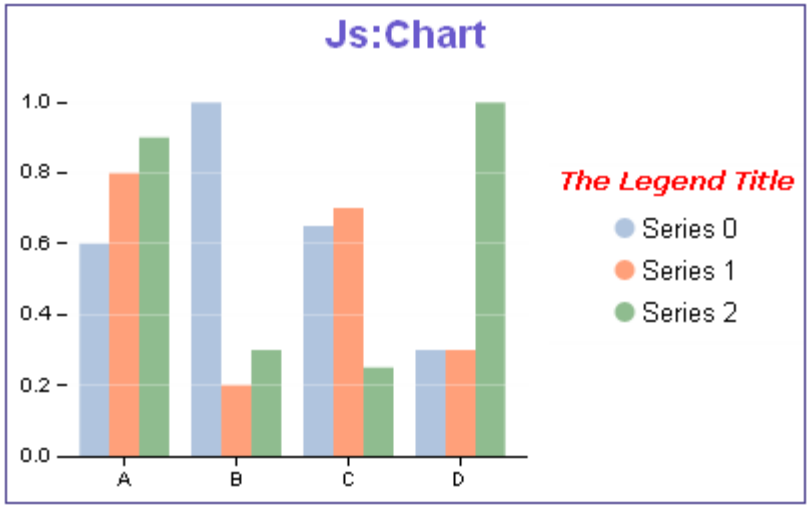
text; a string that defines the Legend Title text. The default value is 'Legend Title'.

font; a string that defines the size and type face of the Legend Title. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color; a string that defines the color of the Legend Title. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

EXAMPLE:

```
legend: {
   visible: true,
   title: {
         visible: true,
         text: 'The Legend Title',
         font: 'Bold Italic 10pt Verdana',
         color: 'red'
   }
}
```



visible

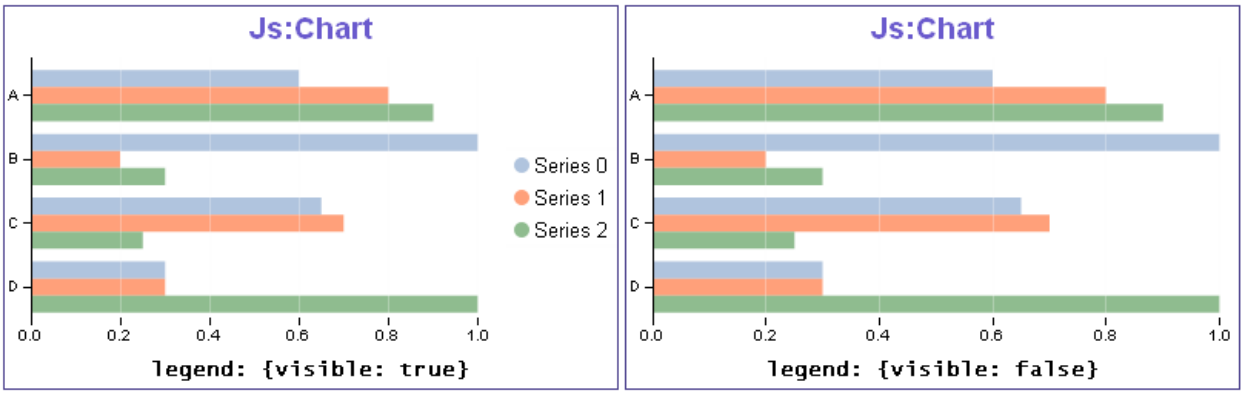
This property controls the visibility of the chart's legend area.

```
legend: {visible: boolean}
```

PARAMETERS:

visible: true = draw legend, false (default) = do not draw legend

EXAMPLE:



NOTES:

When legend position is set to "free", the legend may draw on top of other chart objects. Use the legend:xy property to define the location of the legend.

xy

When legend position is set to "free", this property controls the location of the legend.

```
legend: {xy: {x:Number, y:Number}}
```

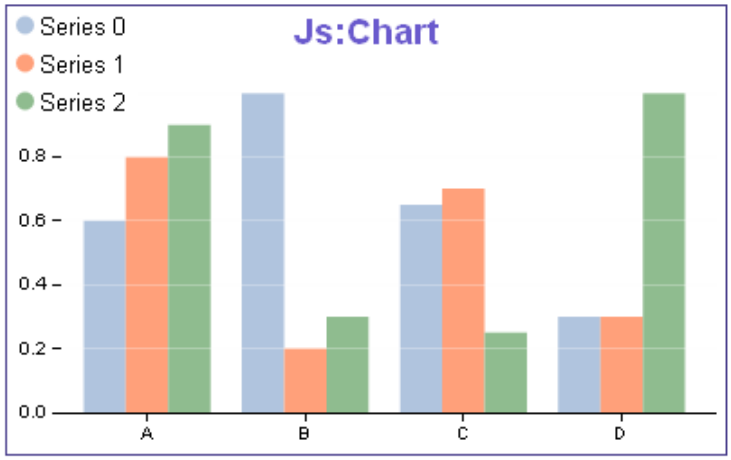
PARAMETERS:

x; Set to a value that defines the X/Horizontal location of the legend (default = 320)

y; Set to a value that defines the Y/Vertical location of the legend (default = 80)

EXAMPLE:

```
legend: {position: 'free',xy: {x:1, y:1}}
```



Axis Properties (xaxis, yaxis, y2axis, zaxis)

These properties control the appearance of chart axes. An axis can be numeric or ordinal (text strings) depending on the data that is supplied to draw the chart. The y2axis only appears in pareto charts and Bar/Line/Area charts where the series.yAxisAssignment property assigns a series to the y2axis. The zaxis only appears on 3D charts and heatmap charts.

altFrameColor

This property fills every other segment of axis grid lines with a specified color.

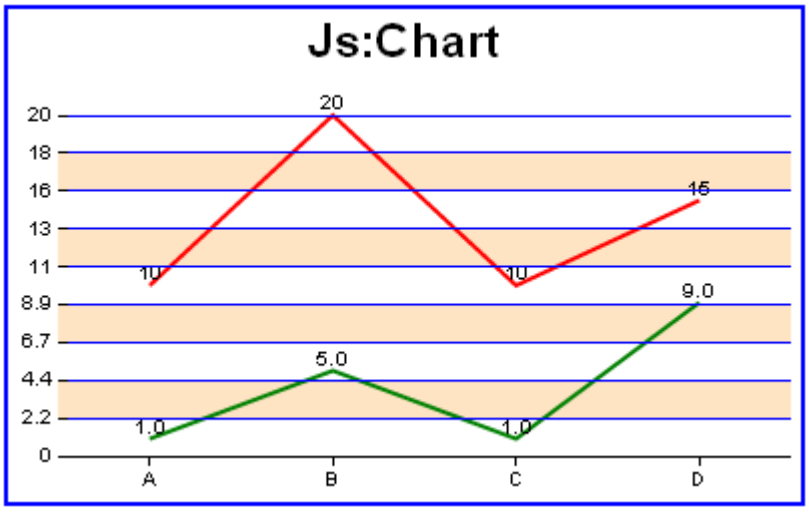
```
xaxis | yaxis | y2axis | zaxis: {
   altFrameColor: 'string' | JSON Object
},
```

PARAMETERS:

color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient. The default value is undefined.

EXAMPLES:

```
yaxis: {
   intervalMode: 'count',
   intervalValue: 10,
   majorGrid: {lineStyle: {width: 1,color: 'blue'}},
   altFrameColor: 'bisque'
},
```



baseLineStyle

On a numeric axis, these properties control the appearance of an axis base line. The base line is parallel to the grid lines and normally passes through the zero value.

```
xaxis: | yaxis | y2axis | zaxis: {
   baseLineStyle: {
         width: Number,
         color: 'string'
         dash: 'string'
   }
}
```

PARAMETERS:

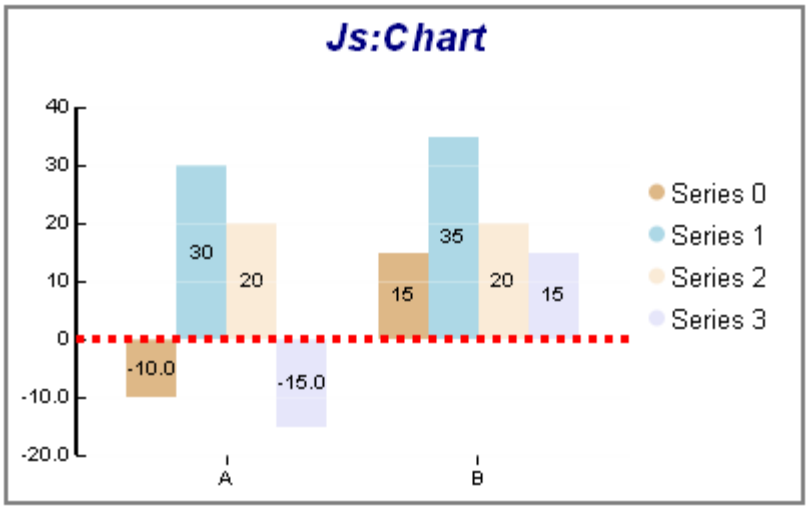
width; a number that defines the width of the line. The default value is 1.

color; a string that defines the color of the line. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

dash: a string that defines the dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

EXAMPLES:

```
blaProperties: {orientation: 'vertical'},
yaxis: {
   bodyLineStyle: {width: 2, color: 'black'},
   baseLineStyle: {width: 4, color: 'red', dash: '4 4'},
}
```



bIsLog

On a numeric axis, this property enables/disables logarithmic scale.

```
xaxis: | yaxis | y2axis | zaxis: {
   bIsLog: boolean
}
```

PARAMETERS:

bIsLog; true/false enables/disables logarithmic scale on the numeric axis. The default value is false.

bodyLineStyle

These properties control the appearance of an axis body line. The axis body line is perpendicular to the grid lines. It is drawn through all of the tick marks and on top of the chart frame (if visible, see the chartFrame property).

```
xaxis: | yaxis | y2axis | zaxis: {
   bodyLineStyle: {
         width: Number,
         color: 'string',
         dash: 'string'
   }
}
```

PARAMETERS:

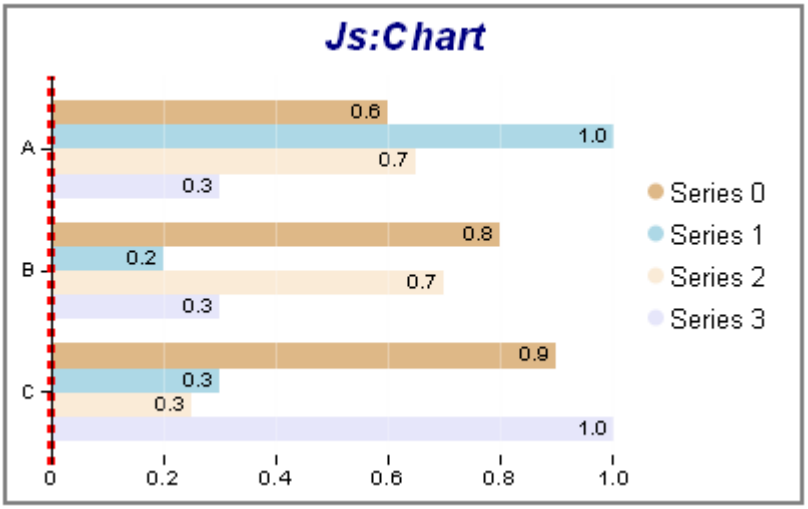
width; a number that defines the width of the line. The default value is 1.

color; a string that defines the color of the line. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'transparent'.

dash: a string that defines the dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

EXAMPLE:

```
xaxis: {bodyLineStyle: {width: 4,color: 'red',dash: '4 4'}}
```



colorBands

These properties create blocks of color that span the overall chart area

```
xaxis: | yaxis | y2axis | zaxis: {
   colorBands: [
         {
               start: 'string' | number,
               stop: 'string' | number,
               color: 'string' | JSON Object
         }
   ],
},
```

PARAMETERS:

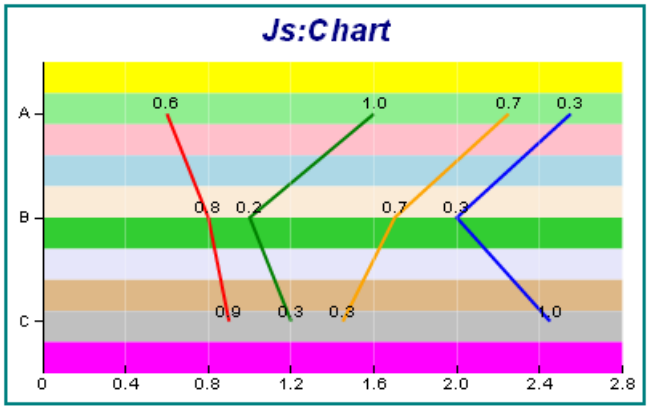
start: a number or string that defines where on the axis to start the color band. For a numeric axis, a number must be a value that is visible on the axis. For an ordinal axis, a number must be a value in the range 0...1 (e.g., .5 will start the color band in the center of the chart). For an ordinal axis, a string must be a group label that is visible on the axis. For a numeric axis, a string must a percentage string (e.g., '50%').

stop: a number or string that defines where on the axis to end the color band. A number can be a value that is visible on the axis or a value in the range 0...1 (e.g., .5 will end the color band in the center of the chart). A string must be a group label that is visible on the axis or a percentage string (e.g., '50%').

color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

EXAMPLES:

```
xaxis: {
   colorBands: [{start: 0,stop: .1,color: 'yellow'},
         {start: .1,stop: .2,color: 'lightgreen'},
         {start: .2,stop: .3,color: 'pink'},
         {start: .3,stop: .4,color: 'lightblue'},
         {start: .4,stop: .5,color: 'antiquewhite'},
         {start: .5,stop: .6,color: 'limegreen'},
         {start: .6,stop: .7,color: 'lavender'},
         {start: .7,stop: .8,color: 'burlywood'},
         {start: .8,stop: .9,color: 'silver'},
         {start: .9,stop: 1.0,color: 'fuchsia'}]},
```



colorScale

When yaxis.mode='color', this property defines the colors to use for drawing the color scale in treemap, heatmap, and tagcloud charts.

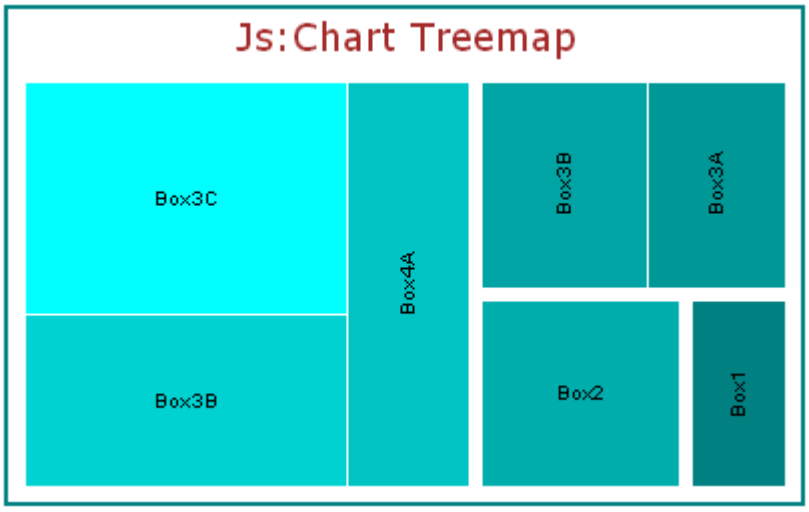
```
yaxis: {
   colorScale: {
         colors: ['color', ... 'color']
   }
}
```

PARAMETERS:

colors: an array of colors. The default value is: ['#253494', '#2C7FB8', '#41B6C4', '#A1DAB4']. Colors can defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

EXAMPLES:

```
chartType: 'treemap',
data: {
   Box1: 10,
   Box2: 20,
   Box3: {Box3A: 15,Box3B: 18},
   Box4: {Box4A: 25,Box3B: 28, Box3C: 38}
},
treemapProperties: {header: {height: 0}},
yaxis: {mode: 'color',colorScale: {colors: ['teal','cyan']}}
```



intervalMode / Value

On a numeric axis, these properties control the number of major grid lines, ticks, and labels.

```
xaxis: | yaxis | y2axis | zaxis: {
   intervalMode: 'string' | undefined
   intervalValue: Number | undefined
}
```

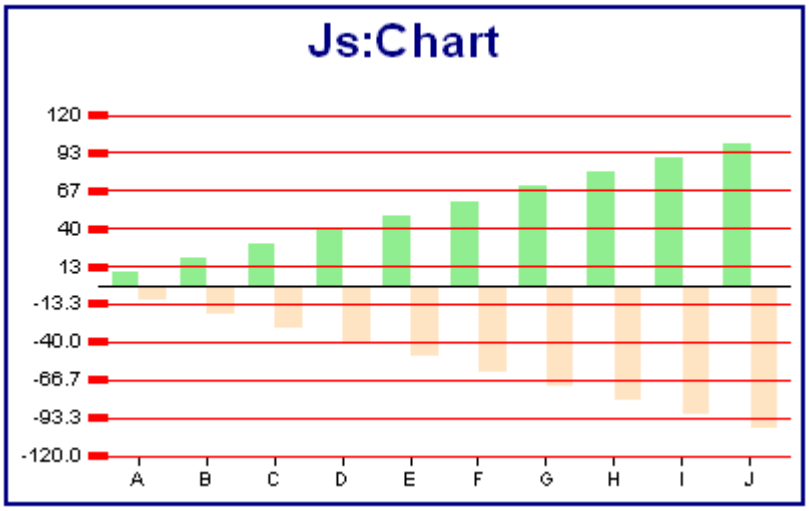
PARAMETERS:

intervalMode; a string that defines the interval mode: 'count', 'interval', or 'skip'. Set to undefined to automatically calculate the number and frequency of major grid lines. The default value is undefined.

intervalValue; if intervalMode is 'count', set this property to the number of major grid lines to draw. If intervalMode is 'interval', set this property to the interval at which major grid lines are drawn. If intervalMode is 'skip', set this property to the number of grid lines to skip. Set to undefined to automatically calculate the number and frequency of major grid lines. The default value is undefined.

EXAMPLE:

```
yaxis: {
   intervalMode: 'count', intervalValue: 10,
},
```



NOTES:

The first and last grid lines are always drawn regardless of the intervalMode or intervalValue.

invert

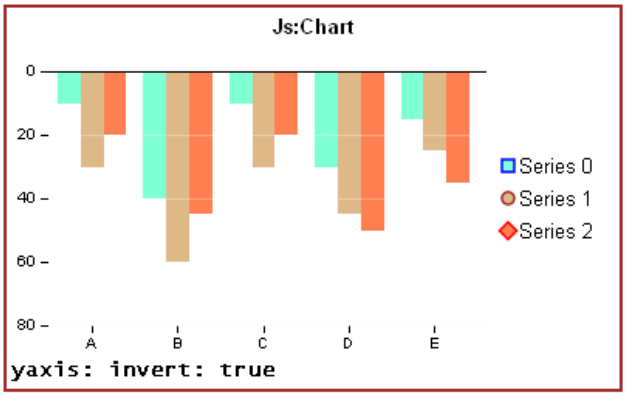
This property controls the direction of the axis: ascending or descending.

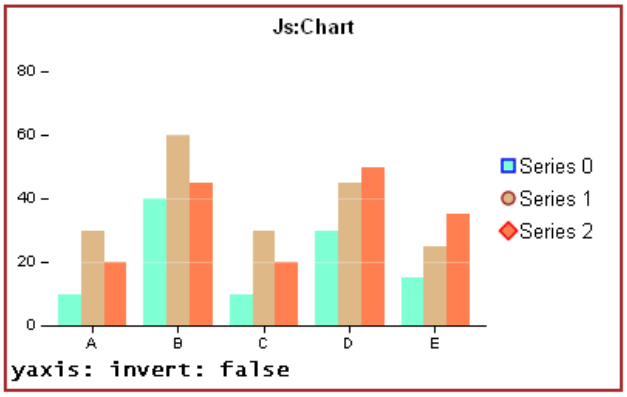
```
xaxis | yaxis | y2axis | zaxis: {
   invert: boolean
}
```

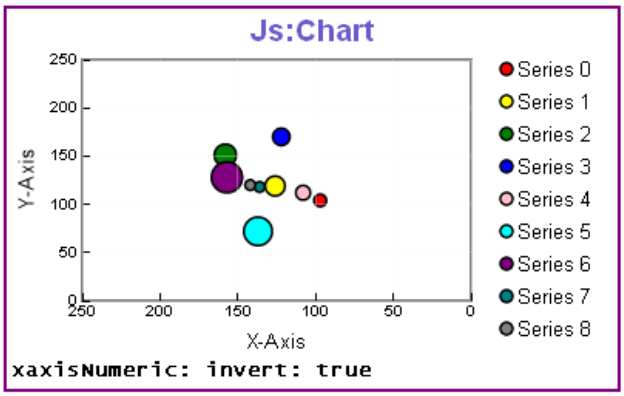
PARAMETERS:

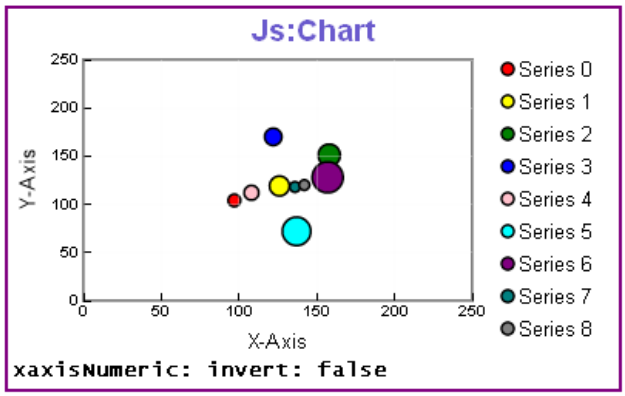
invert: true/false. true = draw descending axis, false = draw ascending axis. The default value is false.

EXAMPLES:









labels

These properties control the visibility and format of axis labels.

```
xaxis | yaxis | y2axis | zaxis: {
   labels: {
         visible: boolean,
         font: 'string',
         color: 'string'
         rotation: number
   }
}
```

PARAMETERS:

visible; true/false controls the visibility of the Axis Labels. The default value is true.

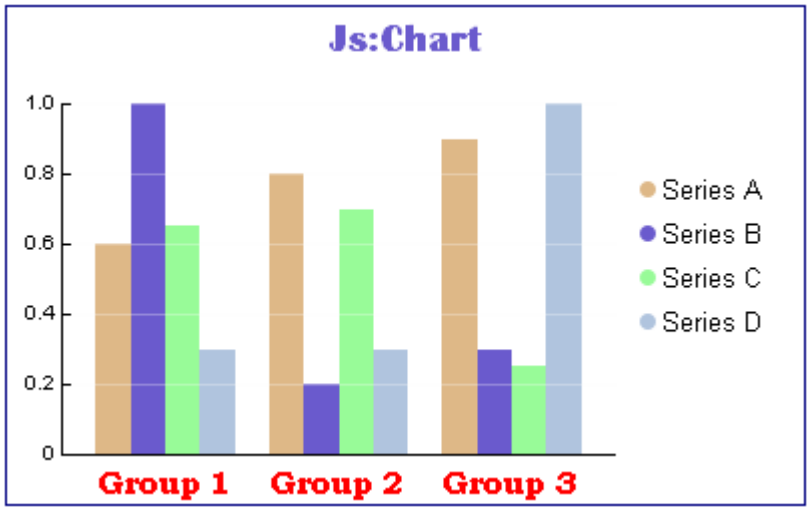
font; a string that defines the size and type face of Axis Labels. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '7.5pt Sans-Serif'.

color; a string that defines the color of Axis Labels. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

rotation: Defines the rotation of axis labels. 0 (none), 45 degrees, 90 degrees, 135 degrees, 180 degrees, 270 degrees, or undefined. Any value other than undefined will disable automatic layout (see chart.axisAutoLayout) of axis labels. The default value is undefined.

EXAMPLE:

```
blaProperties: {orientation: 'vertical'},
xaxis: {
   labels: {visible: true,font: 'bold 12pt Bookman Old Style',
         color: 'red'}}
```



majorGrid

These properties control the visibility and appearance of major grid lines and tick marks.

```
xaxis: | yaxis | y2axis | zaxis: {
   majorGrid: {
         visible: boolean,
         aboveRisers: boolean,
         lineStyle: {
               width: number,
               color: 'string',
               dash: 'string'
         },
         ticks: {
               length: number,
               visible: boolean,
               style: 'string',
               lineStyle: {
                     width: number,
                     color: 'string'
               }
         }
   },
}
```

PARAMETERS:

visible: true = draw major grid lines/false = hide major grid lines. The default value is false.

aboveRisers: true = draw major grid lines above/on top of risers. false = draw major grid lines behind risers. The default value is true.

lineStyle: These properties control the width and color of major grid lines.

lineStyle/width: a number that defines the width of the line. The default value is 1.

lineStyle/color: a string that defines the color of the line. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'rgba(255, 255, 255, 0.3)'.

lineStyle/dash: a string that defines the dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

ticks: These properties control the visibility and format of tick marks.

ticks/length: a number that defines the length of tick marks. The default value is 5.

ticks/visible: true = draw tick marks/false = hide tick marks. The default value is true.

ticks/style: a string that defines the tick mark style: 'inner', 'outer', or 'span'. The default value is 'outer'.

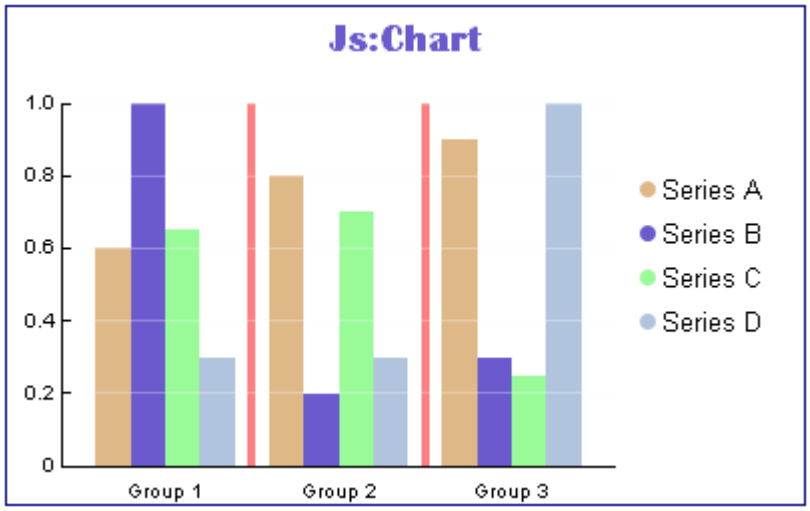
ticks/lineStyle: These properties control the width and color of tick marks.

ticks/lineStyle/width: a number that defines the width of the tick marks. The default value is 1.

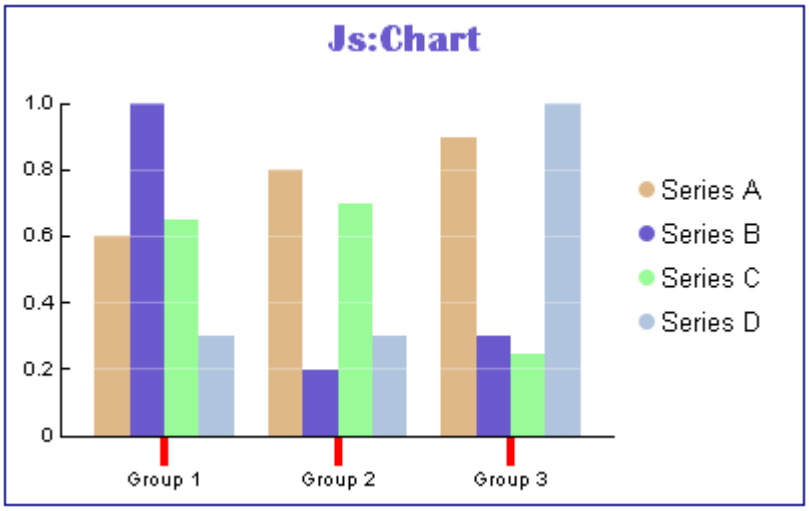
ticks/lineStyle/color: a string that defines the tick mark color. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

EXAMPLES:

```
xaxis: {
   majorGrid: {visible: true, aboveRisers: true,
         lineStyle: {width: 4,color: 'red'}}
}
```




```
xaxis: {
   majorGrid: {
         ticks: {length: 15,visible: true,style: 'outer',
                  lineStyle: {width: 4, color: 'red'}
               }
         }
}
```



min / max

On a numeric axis, these properties define the minimum and maximum values to draw on the axis.

```
xaxis: | yaxis | y2axis | zaxis: {
   min: Number
   max: Number
}
```

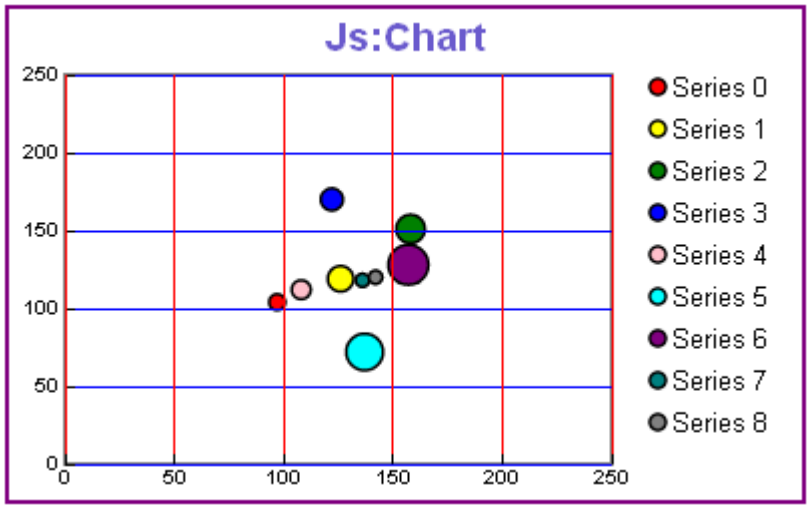
PARAMETERS:

min; minimum value to draw on a Numeric Axis. Set to undefined to automatically calculate the minimum value based on values in the data set. The default value is undefined.

max; maximum value to draw on a Numeric Axis. Set to undefined to automatically calculate the maximum value based on values in the data set. The default value is undefined.

EXAMPLE:

```
chartType: 'bubble',
yaxis: {
   min: 0,max: 250,
   majorGrid: {lineStyle: {width: 1, color: 'blue'}},
   },
xaxis: {
   min: 0,max: 250,
   majorGrid: {lineStyle: {width: 1, color: 'red'}},
   },
```



minorGrid

These properties control the visibility and appearance of minor grid lines and tick marks.

```
xaxis: | yaxis | y2axis | zaxis: {
   minorGrid: {
         visible: boolean,
         count: Number,
         lineStyle: {
               width: Number,
               color: 'string',
               dash: 'string'
         }
         ticks: {
               length: Number,
               visible: boolean,
               style: 'string',
               lineStyle: {width: Number,color: 'string'}
         }
   }
}
```

PARAMETERS:

visible: true = draw minor grid lines/false = hide minor grid lines. The default value is false.

count: number of minor grid lines to draw between each major grid line. The default value is undefined.

lineStyle: These properties define the width, color, and dash style of minor grid lines.

lineStyle/width: a number that defines the width of the line. The default value is 1.

lineStyle/color: a string that defines the color of the line. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

lineStyle/dash: a string that defines the dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

ticks: These properties control the visibility and format of minor tick marks.

ticks/length: a number that defines the length of tick marks. The default value is 5.

ticks/visible: true = draw tick marks/false = hide tick marks. The default value is false.

ticks/style: a string that defines the tick mark style: 'inner', 'outer', or 'span'. The default value is 'inner'.

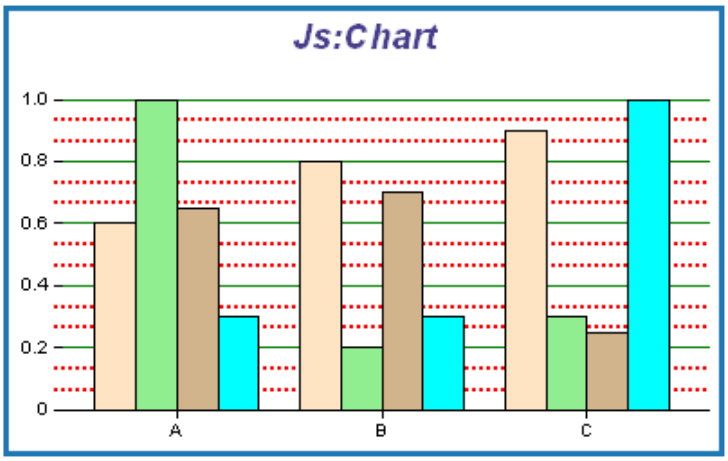
ticks/lineStyle: These properties control the width and color of tick marks.

ticks/lineStyle/width: a number that defines the width of the tick marks. The default value is 1.

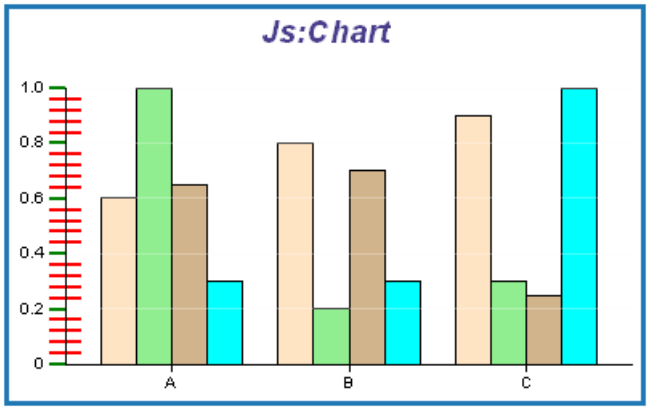
ticks/lineStyle/color: a string that defines the tick mark color. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

EXAMPLES:

```
yaxis: {
   minorGrid: {visible: true,count: 2,
         lineStyle: {width: 2,color: 'red',dash: '2 2'}
   }
},
```




```
yaxis: {
   majorGrid: {
         ticks: {length: 10,visible: true,style: 'outer',
               lineStyle: {width: 2,color: 'green'}}
   },
   minorGrid: {
         ticks: {length: 10,visible: true,style: 'span',
               lineStyle: {width: 2,color: 'red'}}
   }
},
```



mode

This property sets the axis mode as ordinal, numeric, time, or color. The axis mode is typically set to undefined which allows the charting engine to automatically set the mode based on the data.

```
xaxis | yaxis | zaxis: {
   mode: 'string'
}
```

PARAMETERS:

mode: One of 'ordinal', 'numeric', 'time', 'color' or undefined (mode will automatically be chosen based on the data).

NOTES:

For heatmap, tagcloud, and treemap charts, you can set mode to 'color' and define yaxis.colorScale colors to define the coloring of cells in heatmaps and treemaps and the coloring of labels in tagclouds.

numberFormat

On a numeric axis, this property defines the format of numeric axis labels.

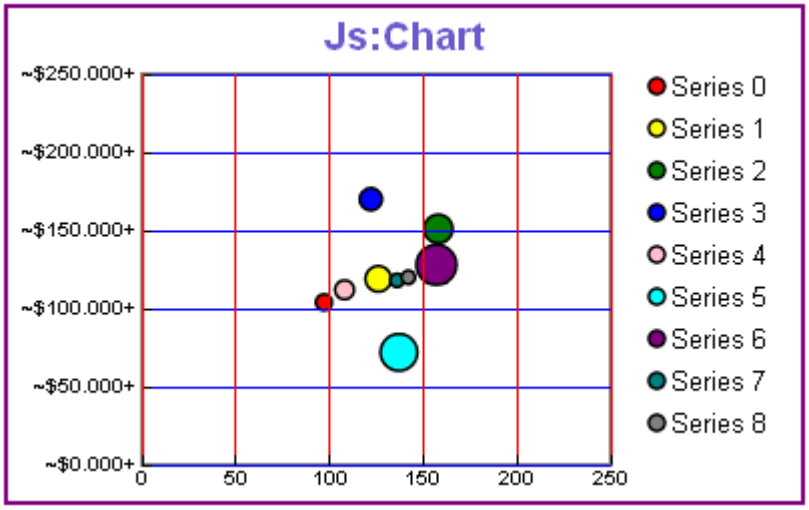
```
xaxis | yaxis | y2axis | zaxis: {
   numberFormat: JSON Object | 'string' | function()
}
```

PARAMETERS:

numberFormat: Use a string, JSON object, or function to define the format of numeric axis labels. See Properties Overview/Formatting Numbers in Section 1 for details and examples. The default value is 'auto'.

EXAMPLES:

```
yaxis: {
   min: 0,
   max: 250,
   numberFormat: {
         mode: 'currency',
         thousandSep: ',',
         decimalSep: '.',
         decimalPlaces: 3,
         prefix: '~',
         suffix: '+'
         },
   majorGrid: {lineStyle: {width: 1, color: 'blue'}},
   },
```



swapChartSide

This property controls the location of the axis body line and labels. In the default configuration, the yaxis body line and labels appear on the bottom of a horizontal chart and the left side of a vertical chart. The Y2-Axis body line and labels appear on the top of a horizontal chart and the right side of a vertical chart. In Bubble and Scatter charts, the X-Axis normally appears on the bottom of the chart and the Y-Axis on the left side of the chart. This property can be used to reverse these default locations..

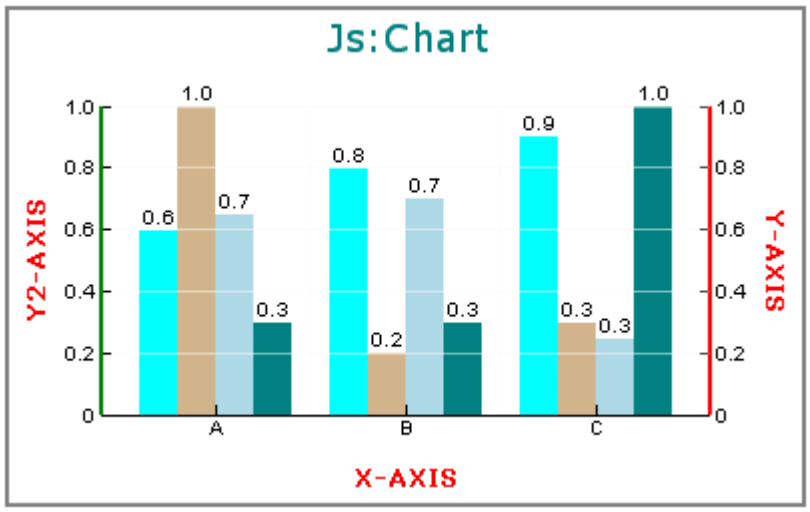
```
xaxis: | yaxis | zaxis: {
   swapChartSide: boolean
}
```

PARAMETERS:

swapChartSide: true = swap labels/body line locations. false = use the default axis locations. The default value is false.

EXAMPLES:

```
blaProperties: {orientation: 'vertical'},
xaxis: {title: {visible: true, color: 'red', font: 'bold 10pt
Verdana',text: 'X-AXIS'}},
yaxis: {title: {visible: true, color: 'red', font: 'bold 10pt
Verdana',text: 'Y-AXIS'},
   bodyLineStyle: {width: 2, color: 'red'},swapChartSide: true},
y2axis: {title: {visible: true, color: 'red', font: 'bold 10pt
Verdana',text: 'Y2-AXIS'},
   bodyLineStyle: {width: 2, color: 'green'}},
series: [
   {series: 0, color: 'cyan'},{series: 1, color: 'tan'},
   {series: 2, color: 'lightblue'},{series: 3, color:
'teal',yAxisAssignment: 2}]
```



timeAxis

These properties create data/time labels on an axis. The primary purpose of a time axis is to auto-generate group labels in units of time based on start/stop times and an optional interval. The library also supports 'sparse' data (i.e., the number of data values that define the chart do not match the number of group labels defined by the time axis). For example, the start/stop times are January/December (12 months) but the data set only includes six values. You can configure the time axis properties to show all 12 month/group labels with the six risers at given months.

```
xaxis: {
timeAxis: {
   enabled: boolean,
   startTime: 'string',
   stopTime: 'string',
   interval: 'string',
   stepSize: Number,
   labelFormat: 'string'
   }
}
```

PARAMETERS:

enabled: true/false enables/disables the time axis. The default value is false (disabled).

startTime: a string identifying the start time. It can be almost any kind of string denoting time, mixing hours (xx:xx), days of the week, dates, months and years. Examples: "Mon", "1 July 20", "June 2001", "8:00", "6/10/01", "Thu, 4 Apr 1976 14:30", "2 0:00" (February 1), "1 1 0:00" (January 1). The default value is undefined.

stopTime: a string identifying the start time. As with startTime, the string can be almost any date/time format. The default value is undefined.

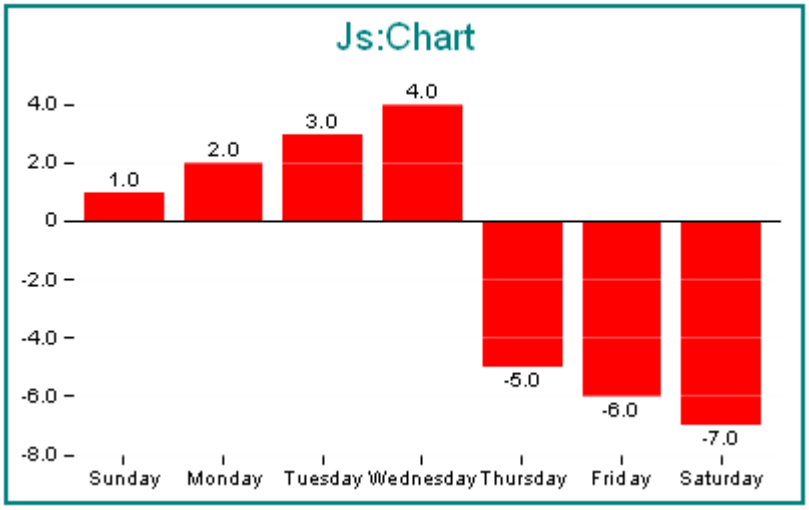
interval: a string that specifies the time interval: 'seconds', 'minutes', 'hours', 'days', 'weeks', 'months', 'quarters', or 'years'. This property defines the time gap between each group label. If undefined, the library will assign an interval based on the startTime, stopTime, and the number of groups. The default value is undefined.

stepSize: a number identifying how many group labels to skip between each label. If undefined (or 0), it's ignored and all labels are drawn. The default value is undefined.

labelFormat: a string that defines how a time/date number is converted into a label. If undefined, the library will we use a default formats based on the interval. See http://code.google.com/p/datejs/wiki/FormatSpecifiers for available format options. The default value is undefined.

EXAMPLES:

```
data: [[1,2,3,4,-5,-6,-7]],
xaxis: {
   timeAxis: {
   enabled: true,
   startTime: 'Sunday',
   stopTime: 'Saturday',
   interval: 'days',
   labelFormat: 'dddd'
   }
},
```



NOTES:

You can also create a time axis by setting xaxis: timeAxis: enabled: true and defining start and stop times in group labels. Example:

```
groupLabels: ["Monday", "Friday"]
```

If timeAxis: startTime is undefined and the first groupLabel is a date/time, it will be used as the start time. If timeAxis: stopTime is undefined and the last groupLabel is a date/time, it will be used as the stop time. However, timeAxis: startTime and stopTime properties take precedence over group labels.

If timeAxis: enabled: true and all of group labels are dates, the library will plot the corresponding data points according to each group label date -- not according to their passed in order. This allows you to specify both sparse and unordered data.

title

These properties control the content, visibility, and format of an axis title.

```
xaxis: | yaxis | y2axis | zaxis: {
   title: {
         text: 'string',
         visible: boolean,
         font: 'string',
         color: 'string'
   }
}
```

PARAMETERS:

text; a string that defines Axis Title text. The default value for xaxis is 'X Axis Title'.

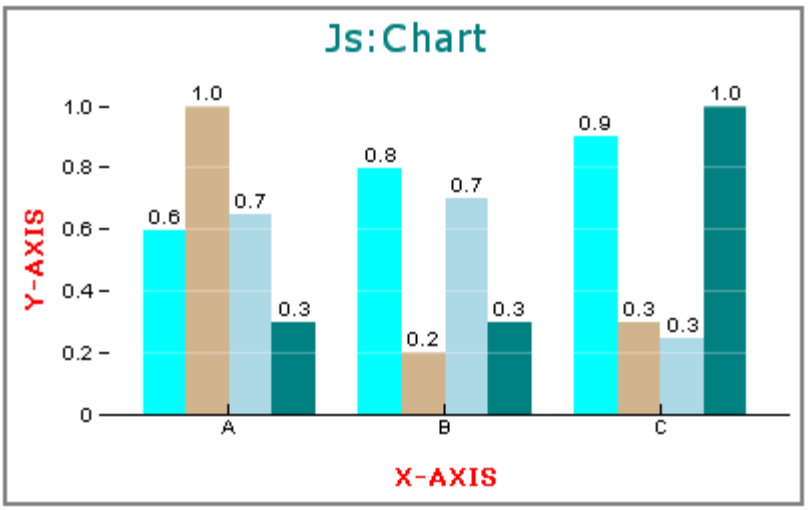
visible; true/false controls the visibility of the Axis Title. The default value is false.

font; a string that defines the size and type face of the Axis Title. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value for xaxis is '7.5pt Sans-Serif'.

color; a string that defines the color of the Axis Title. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

EXAMPLE:

```
blaProperties: {orientation: 'vertical'},
xaxis: {title: {visible: true, color: 'red', font: 'bold 10pt
Verdana',text: 'X-AXIS'}},
yaxis: {title: {visible: true, color: 'red', font: 'bold 10pt
Verdana',text: 'Y-AXIS'}},
series: [{series: 0, color: 'cyan'},{series: 1, color:
'tan'},{series: 2, color: 'lightblue'},{series: 3, color: 'teal'}]
```



Series-Specific Properties

These properties can be used to control the appearance of individual series:

```
series: [
   series: Number,
   group: Number,
   visible: true,
   label: 'string',
   color: 'string' | JSON Object,
   riserShape: 'string'
   border: {
         width: Number,
         color: 'string',
         dash: 'string'
   },
   marker: {
         visible: boolean,  // Line Charts Only
         color: 'string',
         size: Number,
         shape: 'string',
         rotation: Number,
         position: 'string',
         fillMode: 'string',
         border: {width: Number,color: 'string',dash: 'string'}
   },
   // Per-series trendline
   trendline: {
         enabled: boolean,
         mode: 'string',
         line: {width: Number,color: 'string',dash: 'string'},
         equationLabel: {
               visible: false,
               font: 'string',
               color: 'string',
               mode: ''string''
         }
   },
   showDataValues: boolean,
   explodeSlice: Number,
   deleteSlice: boolean,
   yAxisAssignment: Number,
   tooltip: 'string' | function()
]
```

The following code segment shows the default settings from properties.js:

```
series: [
   {series: 'all', color: 'blue', showDataValues: true, border:
{width: 2}, marker: {size: 8, border: {width: 1, color: 'black'}}},
   {series: 0, color: 'red'},
   {series: 1, color: 'green'},
   {series: 2, color: 'orange'}
]
```

border

These properties define a border for all risers, for all risers in an individual series, or for an individual riser identified by a series and group number. It can be used for area risers, bar risers, funnel segments, gauge needles, and pie slices.

```
series: [
```

```
   {
         series: Number,
         group: Number, // optional
         border: {
               width: Number,
               color: 'string',
               dash: 'string'
         },
   }
]
```

PARAMETERS:

series: zero-based series number. If the series does not exist in the chart, the property is ignored.

group: (optional) zero-based group number. If the group does not exist in the chart, the property is ignored. If a group number is not specified, the border is applied to all risers in the series.

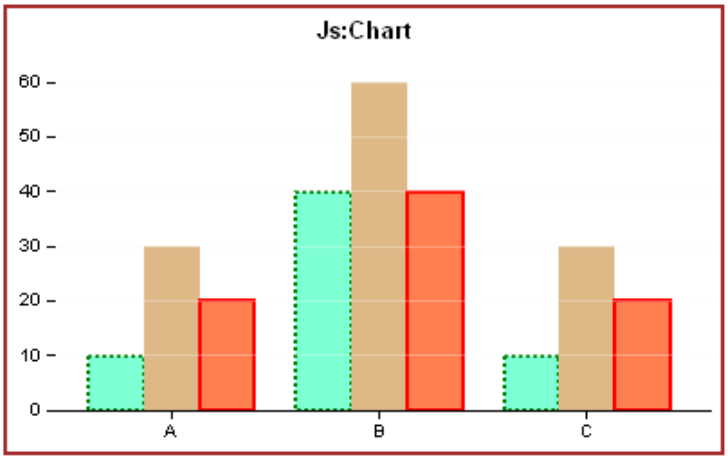
width: a number that defines the width of the border.

color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

dash: a string that defines the border dash style. Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes.

EXAMPLES:

```
blaProperties: {orientation: 'vertical'}
series: [
   {series: 0, color: 'aquamarine', border: {width: 2, color:
'green', dash: '2 2'}},
   {series: 1, color: 'burlywood'},
   {series: 2, color: 'coral', border: {width: 2, color: 'red'}},
   ]
```



color

These properties define a color for all risers, for all risers in an individual series, or for an individual riser identified by a series and group number. If a group number is not specified, the color is also applied to the series icon in the legend area.

```
series: [
   {
         series: Number,
         group: Number, // optional
         color: 'string' | JSON Object
   },
]
```

PARAMETERS:

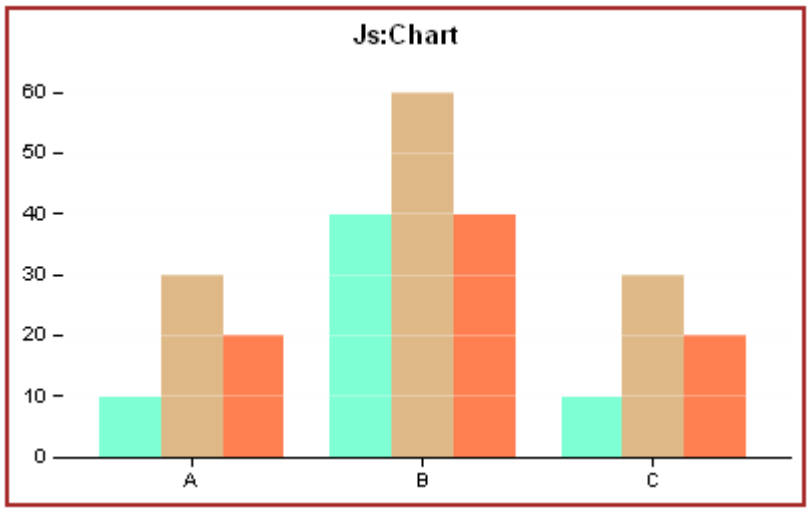
series: zero-based series number. If the series does not exist in the chart, the property is ignored.

group: optional zero-based group number. If the group does not exist in the chart, the property is ignored. If a group number is not specified, the color is applied to all risers in the series.

color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

EXAMPLES:

```
blaProperties: {orientation: 'vertical'}
series: [
   {series: 0, color: 'aquamarine'},
   {series: 1, color: 'burlywood'},
   {series: 2, color: 'coral'},
   ]
```



deleteSlice

This property deletes a slice from a pie chart (effectively, assigns the slice a transparent color).

```
series: [
   {
         series: Number | 'string,
         group: Number, // optional
         deleteSlice: boolean
   }
]
```

PARAMETERS:

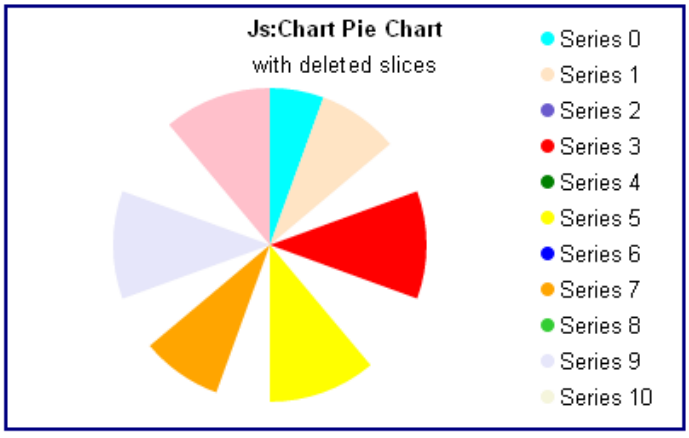
series: zero-based series number or 'all' to delete all slices. If the series does not exist in the chart, the property is ignored.

group: For multi-pie charts, optional zero-based group/pie number. If the group does not exist in the chart, the property is ignored. If a group number is not specified, the slice is deleted from all pies.

deleteSlice: true/false. true: delete the slice, false: restore the slice.

EXAMPLE:

```
border: {width: 2, color: 'navy'},
chartType: 'pie',
title: {text: 'Js:Chart Pie Chart'},
subtitle: {visible: true, text: 'with deleted slices'},
series: [
   {series: 0, color: 'cyan'},
   {series: 1, color: 'bisque'},
   {series: 2, color: 'slateblue', deleteSlice: true},
   {series: 3, color: 'red'},
   {series: 4, color: 'green', deleteSlice: true},
   {series: 5, color: 'yellow'},
   {series: 6, color: 'blue', deleteSlice: true},
   {series: 7, color: 'orange'},
   {series: 8, color: 'limegreen', deleteSlice: true},
   {series: 9, color: 'lavender'},
   {series: 10, color: 'beige', deleteSlice: true},
   {series: 11, color: 'pink'},
]
```



explodeSlice

This property defines the distance (in pixels) to push a slice away from the pie chart.

```
series: [
   {
         series: number,
         group: number, // optional
         explodeSlice: number | string
   }
]
```

PARAMETERS:

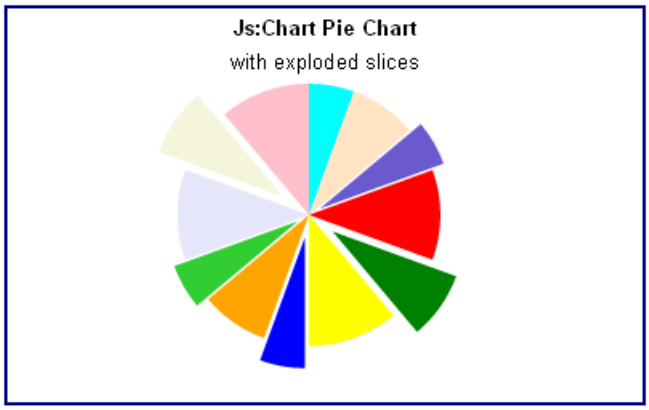
series: zero-based series number. If the series does not exist in the chart, the property is ignored.

group: For multi-pie charts, optional zero-based group/pie number. If the group does not exist in the chart, the property is ignored. If a group number is not specified, the property is applied to all pies.

explodeSlice: a number that defines the distance (in pixels) to push a slice away from the pie or a percentage string (e.g., '25%') to explode a slice a percentage of the radius of the pie. Example: if the pie is 200 pixels in diameter (100 pixels radius), explodeSlice: '50%' will push the slice out 50 pixels from the center of the pie.

EXAMPLE:

```
chartType: 'pie',
subtitle: {visible: true, text: 'with exploded slices'},
series: [
   {series: 0, color: 'cyan'},
   {series: 1, color: 'bisque'},
   {series: 2, color: 'slateblue', explodeSlice: 8},
   {series: 3, color: 'red'},
   {series: 4, color: 'green', explodeSlice: 18},
   {series: 5, color: 'yellow'},
   {series: 6, color: 'blue', explodeSlice: 14},
   {series: 7, color: 'orange'},
   {series: 8, color: 'limegreen', explodeSlice: 8},
   {series: 9, color: 'lavender'},
   {series: 10, color: 'beige', explodeSlice: 20},
   {series: 11, color: 'pink'},
]
```



label

This property assigns a label to an individual series that will be shown in the chart legend area. You can use the legend:labels property to define the format and color of the series label.

```
series: [
   {
         series: Number,
         label: 'string',
   }
]
```

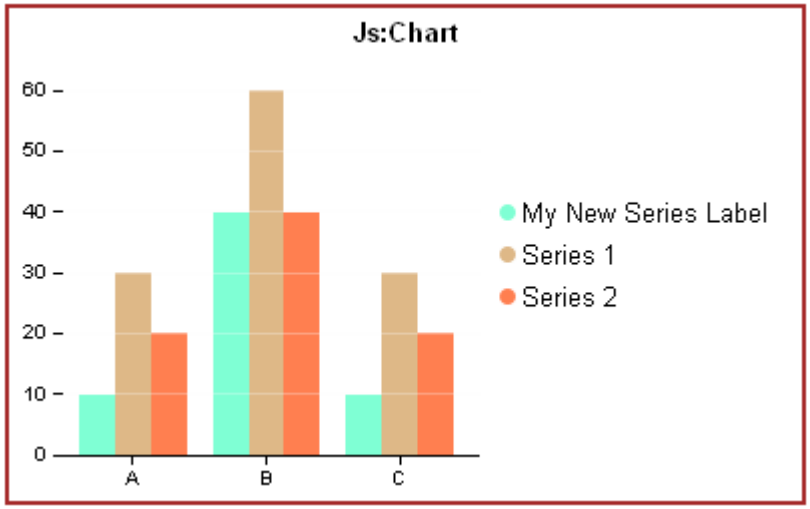
PARAMETERS:

series: zero-based series number. If the series does not exist in the chart, the property is ignored.

label: a string that defines the series label.

EXAMPLES:

```
blaProperties: {orientation: 'vertical'},
legend: {visible: true},
series: [
   {series: 0, color: 'aquamarine', label: 'My New Series Label'},
   {series: 1, color: 'burlywood'},
   {series: 2, color: 'coral'},
   ]
```



marker

These properties define the border color, size, shape and rotation of series markers.

```
series: [
   {
         series: Number,
         group: Number, // optional
         marker: {
               visible: boolean,  // Line Charts Only
               color: 'string',
               size: Number,
               shape: 'string',
               rotation: Number,
               position: 'string',
               fillMode: 'string',
               border: {
                     width: Number,
                     color: 'string',
                     dash: 'string'
               }
         },
   }
]
```

PARAMETERS:

series: zero-based series number. If the series does not exist in the chart, the property is ignored.

group: (optional) zero-based group number. If the group does not exist in the chart, the property is ignored. If a group number is not specified, the marker definition is applied to all risers in the series.

visible: true/false controls the visibility of markers (in line charts only).

color: a color defined by a keyword or numerical specification string or a gradient defined by a string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

size: a number that defines the size of the marker.

shape: a string that defines the shape of markers for a series: arrow, bar, circle, cross, diamond, fiveStar, hexagon, hourglass, house, pie (for bubble and scatter charts only), pin, pirateCross, plus, rectangle, sixStar, square, thinPlus, tick, or triangle.

rotation: a number 0...360 that defines the angle (in degrees) of the marker.

position: for bullet charts only, defines the position of the marker relative to data text: 'bottom', 'middle', 'top'. 'top' draws the data text label below the marker. 'bottom' draws the data text label above the marker. 'middle' draws the data text label according to the chart.dataLabels.position.

fillMode: undefined or a string that defines the marker fill mode: 'seriesHollow', 'seriesFill', 'seriesWhite', or 'seriesAuto'. For 'seriesHollow', the marker fill area is hollow (transparent) and the marker border is the color set by the series.color property. For 'seriesFill', the marker draws with no border and the fill is the same as the series.color property. For 'seriesWhite', the marker fill area is white and the marker border is the color set by the series.color property. For 'seriesAuto', the marker draws with the border 20% darker than the series color and the fill

draws 20% lighter than the series color. Use undefined (the default) to use the existing marker border color and fill color.

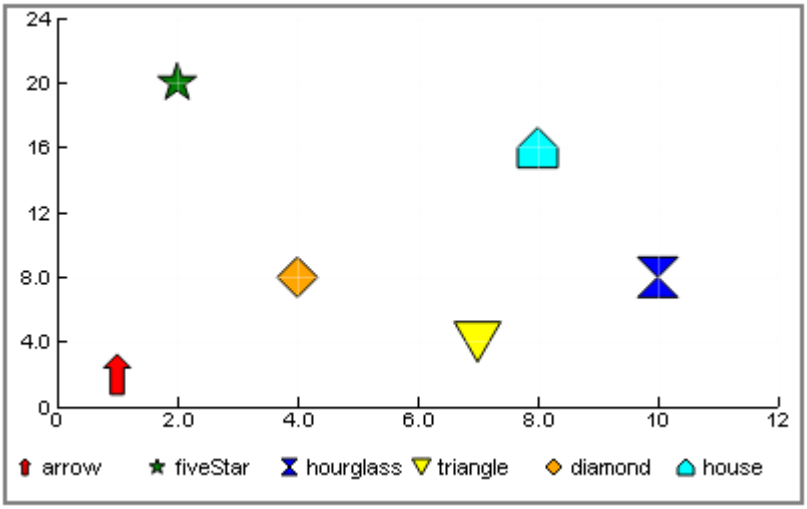
border/width: a number defines the width of the border in pixels.

border/color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

border/dash: a string that defines the border dash style. Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes.

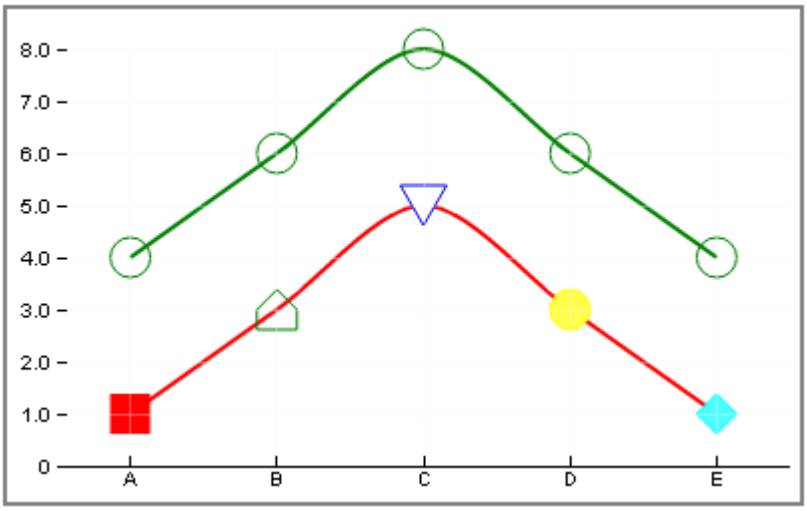
EXAMPLES:

```
data: [[[1, 2]], [[2, 20]],[[10, 8]],[[7,4]],[[4, 8]],[[8, 16]]],
chartType: 'scatter',
legend: {visible: true,labels: {font: '8pt Sans-Serif'},position:
'bottom'},
yaxis: {max: 24},
xaxis: {min: 0, max: 12},
dataLabels: {visible: false},
series: [
{series: 0, color: 'red', label: 'arrow',marker: {shape:'arrow',
size: 20}},
{series: 1, color: 'green', label: 'fiveStar',marker:
{shape:'fiveStar', size: 20}},
{series: 2, color: 'blue', label: 'hourglass',marker:
{shape:'hourglass', size: 20}},
{series: 3, color: 'yellow', label: 'triangle',marker:
{shape:'triangle', size: 20}},
{series: 4, color: 'orange', label: 'diamond',marker:
{shape:'diamond', size: 20}},
{series: 5, color: 'cyan', label: 'house',marker: {shape:'house',
size: 20}},
]
```

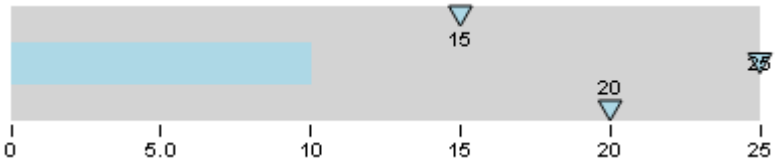



```
border:{width:2, color:'grey'},
data: [[1,3,5,3,1],[4,6,8,6,4]],
chartType: 'line',
blaProperties: {orientation: 'vertical', lineConnection: 'curved'},
dataLabels: {visible: false},
series: [
{series: 'all', marker: {visible: true}},
{series: 0, group: 0, color: 'red', marker: {shape:'square', size:
20,fillMode: 'seriesFill'}},
{series: 0, group: 1, color: 'green', marker: {shape:'house', size:
20,fillMode: 'seriesHollow'}},
```

```
{series: 0, group: 2, color: 'blue', marker: {shape:'triangle', size:
20,fillMode: 'seriesWhite'}},
{series: 0, group: 3, color: 'yellow', marker: {shape:'circle', size:
20,fillMode: 'seriesAuto'}},
{series: 0, group: 4, color: 'cyan', marker: {shape:'diamond', size:
20,fillMode: 'seriesAuto'}},
{series: 1, color: 'green', marker: {shape:'circle', size:
20,fillMode: 'seriesHollow'}},
]
```




```
data: [[10,15,20,25]],
height: 85,
chartType: 'bullet',
series: [
{series: 0, group: 0, color: 'lightblue'},
{series: 0, group: 1, color: 'lightblue',marker: {position: 'top'}},
{series: 0, group: 2, color: 'lightblue',marker: {position:
'bottom'}},
{series: 0, group: 3, color: 'lightblue',marker: {position:
'middle'}}
],
dataLabels: {position: 'center'}
```



NOTES:

- The bar, cross, and tick marker shapes require a border width and color.
- The special pie marker shape is for bubble and scatter charts only and requires the use of series: 'all' {marker: {shape: 'pie'}}. See the bubble and scatter charts for details and examples.

riserShape

This property is used in conjunction with blaProperties.comboCharts to define a combination chart (i.e., a chart that can consist of a combination of bar risers, area risers, and line risers). When the chartType property is set to 'bar', 'line', or 'area', this property can be used to define the shape of risers (bar, line, or area) for each series. The comboCharts properties control the layout (stacked, absolute, percent, or sideBySide) of the risers for each series.

PARAMETERS:

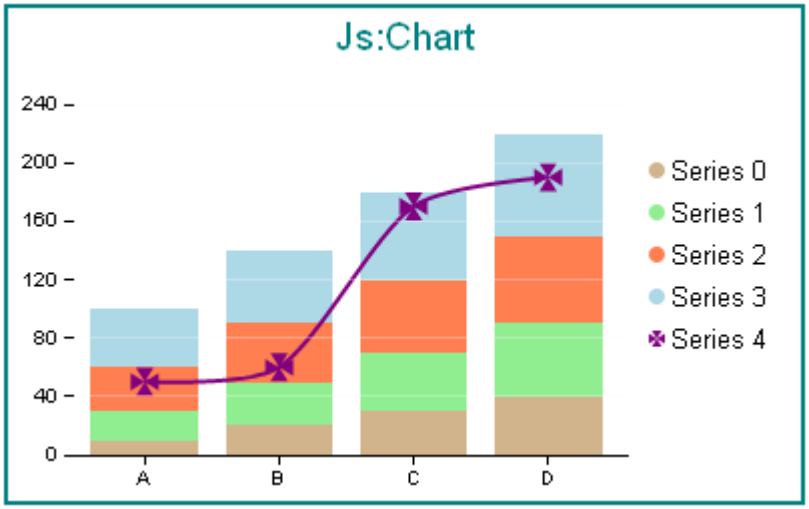
```
series: [
   {
         series: Number,
         riserShape: 'string'
   },
]
```

series: zero-based series number. If the series does not exist in the chart, the property is ignored.

riserShape: a string that defines the shape of the riser for the specified series: 'bar', 'line', or 'area'.

EXAMPLE:

```
legend: {visible: true},
blaProperties: {orientation: 'vertical',lineConnection:'curved',
   comboCharts: {barSeriesLayout: 'stacked', lineSeriesLayout:
'absolute',areaSeriesLayout: undefined}
},
series: [
{series: 0, color: 'tan', riserShape: 'bar'},
{series: 1, color: 'lightgreen', riserShape: 'bar'},
{series: 2, color: 'coral', riserShape: 'bar'},
{series: 3, color: 'lightblue', riserShape: 'bar'},
{series: 4, color: 'purple', riserShape: 'line', border: {width:
2},marker: {shape: 'pirateCross', size: 14, visible: true}}
]
```



showDataValues

This property enables/disables data text labels for individual series and groups.

```
series:
[
    (
         series: Number,
         group: Number,
         showDataValues: boolean,
   }
]
```

PARAMETERS:

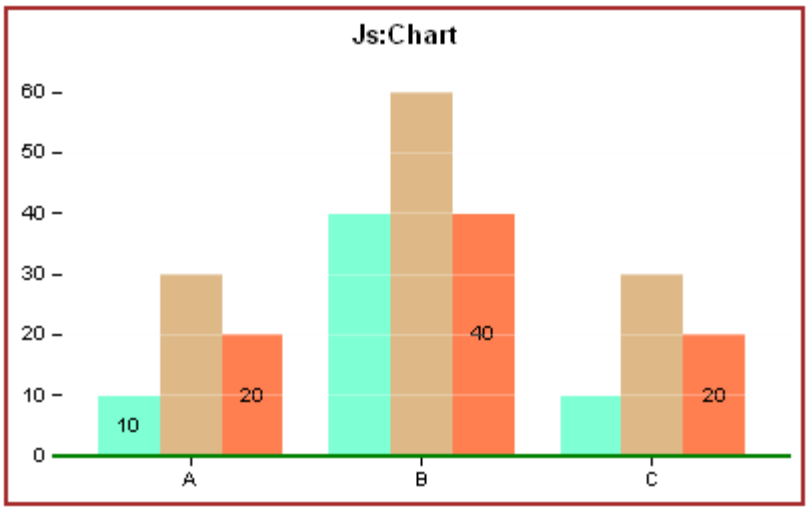
series: zero-based series number. If the series does not exist in the chart, the property is ignored.

group: (optional) zero-based group number. If the group does not exist in the chart, the property is ignored. If a group number is not specified, the property definition is applied to all risers in the series.

showDataValues: When the dataLabels: visible property is true, use showDataValues false to turn off data labels for individual series/groups.

EXAMPLE:

```
blaProperties: {orientation: 'vertical'},
dataLabels: {visible: true},
series: [
   {series: 0, color: 'aquamarine'},
   {series: 0, group: 1, showDataValues: false},
   {series: 0, group: 2, showDataValues: false},
   {series: 1, color: 'burlywood', showDataValues: false},
   {series: 2, color: 'coral'}
]
```



tooltip

This property defines a tooltip for one or more risers. The tooltip string is shown when the mouse hovers over a riser. A tooltip can be assigned to: all risers/markers, a single riser/marker (identified by a series and group), or all risers/markers in a series.

```
series: [
   {
         series: Number | 'string',
         group: Number, // optional
         tooltip: 'string' | function
   }
]
```

PARAMETERS:

series: zero-based series number. If the series does not exist in the chart, the property is ignored. Use 'all' to define the tooltip for all risers.

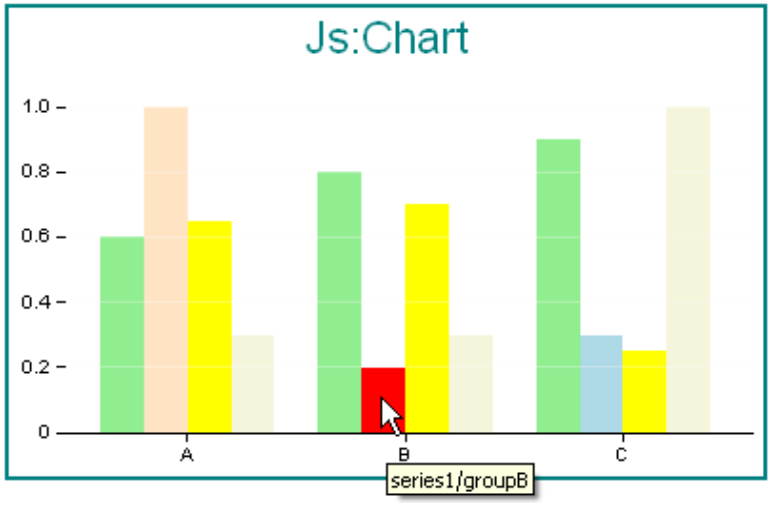
group: (optional) zero-based group number. If the group does not exist in the chart, the property is ignored. If a group number is not specified, the property is applied to all groups.

tooltip: a string or a function that returns a string to show when the mouse hovers over the riser. If htmlTooltip.enabled = true, you can use the special string 'auto' to automatically populate the tooltip with series labels, group labels, and values. The content will be different for different chart types. See htmlTooltip for examples. A callback function is invoked with three arguments: value, series ID, and group ID. Example:

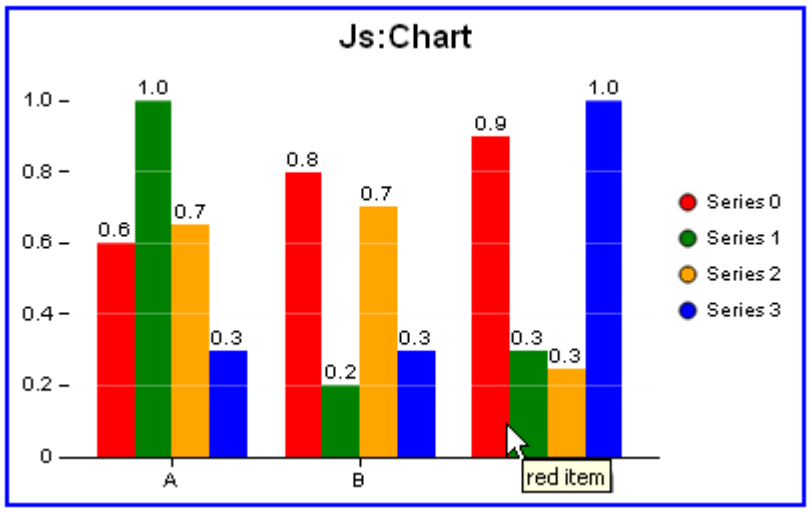
```
chart.series[0].tooltip = function(value, series, group) {
    return this.title.text + ", value: " + value + ", s: " + series +
", g: " + g;
}
```

EXAMPLES:

```
series: [
{series: 0, color: 'lightgreen'},
{series: 1, group: 0, color: 'bisque'},
{series: 1, group: 1, color: 'red', tooltip: 'series1/groupB'},
{series: 1, group: 2, color: 'lightblue'},
{series: 2, color: 'yellow'},
{series: 3, color: 'beige'},
]
```



```
blaProperties: {orientation: 'vertical'},
legend: {visible: true},
series: [
{series: 0, color: 'red',  tooltip: 'red item'},
]
```



NOTES:

When a tooltip is defined for one or all series, the htmlToolTip property can be used to define HTML-based (div) style tooltips for any chart tooltips.

trendline

These properties draw a trendline and equation label in bubble and scatter charts.

```
series: [
{
   series: Number
   trendline: {
         enabled: false,
         mode: undefined
         line: {
               width: 1,
               color: 'black',
               dash: ''
         },
         equationLabel: {
               visible: false,
               font: '8pt Sans-Serif',
               color: 'black'
               mode: 'equation'
               }
         },
   }
]
```

PARAMETERS:

series: zero-based series number. If the series does not exist in the chart, the property is ignored.

enabled: true/false enables/disables the trendline.

mode: a string that defines the trendline mode: exponential', 'geometric', 'hyperbolic', 'linear', 'logarithmic', 'logQuadratic', 'modExponential', 'modHyperbolic', 'polynomial', 'quadratic', 'rational' or undefined. The default value is undefined.

line/width: a number that defines the width of the trendline. The default value is 1.

line/color: a color defined by a keyword or numerical specification string that defines the color of the trendline. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

line/dash: a string that defines the line's dash style. Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line). Use '' for a solid line.

equationLabel/visible: true/false controls the visibility of the equation label

equationLabel/font; a string that defines the size, style, and, typeface of the equation label. The default value is '8pt Sans-Serif'. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string.

equationLabel/color: a color defined by a keyword or numerical specification string. The default value is 'black'. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

equationLabel/mode: a string that defines the equation label mode: 'equation' (equation in slope intercept form) or 'r' (correlation coefficient). The default value is 'equation'.

visible

This property controls the visibility of individual series.

```
series:
[
    (
         series: Number,
         visible: boolean,
   }
]
```

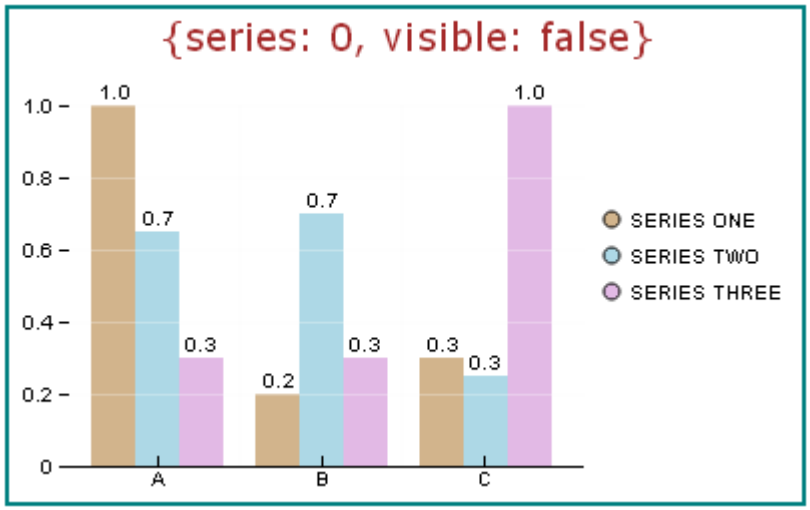
PARAMETERS:

series: zero-based series number. If the series does not exist in the chart, the property is ignored.

visible: true/false controls the visibility of a series.

EXAMPLES:

```
title: {text: "{series: 0, visible: false}",font: '14pt
Verdana',color: 'brown'},
border:{width:2, color:'teal'},
blaProperties: {orientation: 'vertical'},
legend: {visible: true},
series: [
{series: 0, color: 'rgb(204,255,255)', label: 'SERIES ZERO', visible:
false},
{series: 1, color: 'tan', label: 'SERIES ONE'},
{series: 2, color: 'lightblue', label: 'SERIES TWO'},
{series: 3, color: 'rgb(226,185,229)', label: 'SERIES THREE'},
]
```



yAxisAssignment

This property assigns a series to an axis. When *yAxisAssignment* is set to 2, Y2-Axis labels are added to the chart. See the Numeric Axes for properties that control other y2axis objects.

```
series: [
   {
         series: Number,
         yAxisAssignment: number
   }
]
```

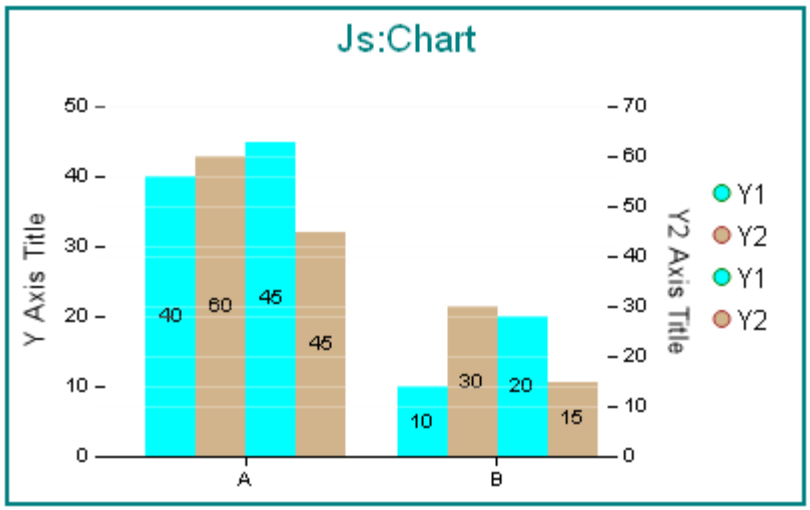
PARAMETERS:

series: zero-based series number. If the series does not exist in the chart, the property is ignored.

yAxisAssignment: 1 = assign series to Y-axis, 2 = assign series to Y2-Axis.

EXAMPLE:

```
blaProperties: {orientation: 'vertical'},
legend: {visible: true},
yaxis:{title: {visible: true}},
y2axis:{title: {visible: true}},
series:[
{series: 0, color: 'cyan', yAxisAssignment: 1, label: 'Y1', marker:
{border: {width: 1, color: 'green'}}},
{series: 1, color: 'tan', yAxisAssignment: 2, label: 'Y2', marker:
{border: {width: 1, color: 'brown'}}},
{series: 2, color: 'cyan', yAxisAssignment: 1, label: 'Y1', marker:
{border: {width: 1, color: 'green'}}},
{series: 3, color: 'tan', yAxisAssignment: 2, label: 'Y2', marker:
{border: {width: 1, color: 'brown'}}},
]
```




THREE D GRAPHICS

Section 4: Chart-Specific Properties

Bar / Line / Area Charts (blaProperties)

These properties control the basic layout of Bar / Line / Area Charts. The following code segment shows the default values from properties.js.

```
blaProperties: {
   seriesLayout: 'sideBySide',
   orientation: 'horizontal',
   lineConnection: 'linear',
   lineFillEffect: undefined,
   barGroupGapWidth: 0.2
   stackTotalLabel: {
         visible: false,
         font: '10pt Sans-Serif',
         color: 'black',
         numberFormat: 'auto'
   },
   comboCharts: {
         barSeriesLayout: undefined,
         lineSeriesLayout: undefined,
         areaSeriesLayout: undefined,
   }
}
```

barGroupGapWidth

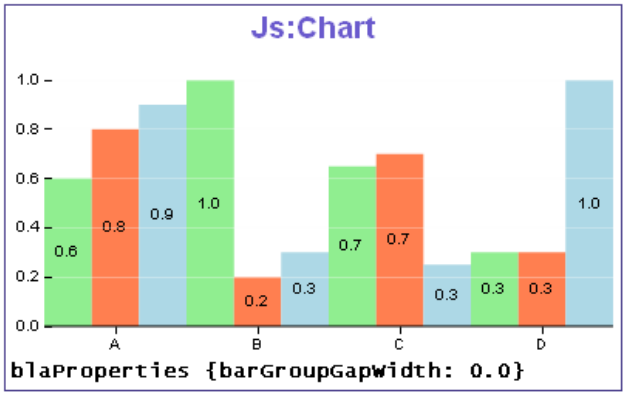
This property controls the width of the gap between groups of risers.

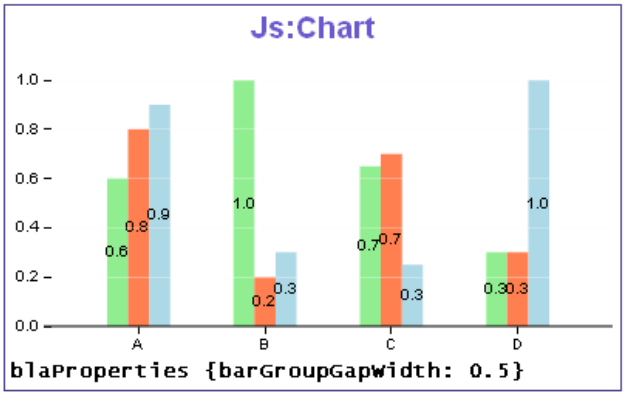
```
blaProperties: {
   barGroupGapWidth: Number
}
```

PARAMETERS:

barGroupGapWidth; 0.0 to 1.0 defines the width of the gap between groups of risers. The default value is 0.2.

EXAMPLE:





comboCharts

These properties are used in conjunction with the series#:riserShape property to define a combination chart (i.e., a chart that can consist of a combination of bar risers, area risers, and line risers). When the chartType property is set to 'bar', 'line', or 'area', the series#:riserShape property can be used to define the shape of risers (bar, line, or area) for each series. These properties control the layout (stacked, absolute, percent, or sideBySide) of the risers for each series.

```
blaProperties: {
   comboCharts: {
         barSeriesLayout: 'string' | undefined,
         lineSeriesLayout: 'string' | undefined,
         areaSeriesLayout: 'string' | undefined,
   }
}
```

PARAMETERS:

barSeriesLayout: a string or undefined. The default value is undefined. Use 'absolute', 'percent', 'sideBySide', or 'stacked' to define the layout of any series assigned to bar (series#:markerShape: 'bar'). undefined = use the series layout defined in blaProperties.seriesLayout.

lineSeriesLayout: a string or undefined. The default value is undefined. Use 'absolute', 'percent', or 'stacked' to define the layout of any series assigned to line (series#:markerShape: 'line'). undefined = use the series layout defined in blaProperties.seriesLayout.

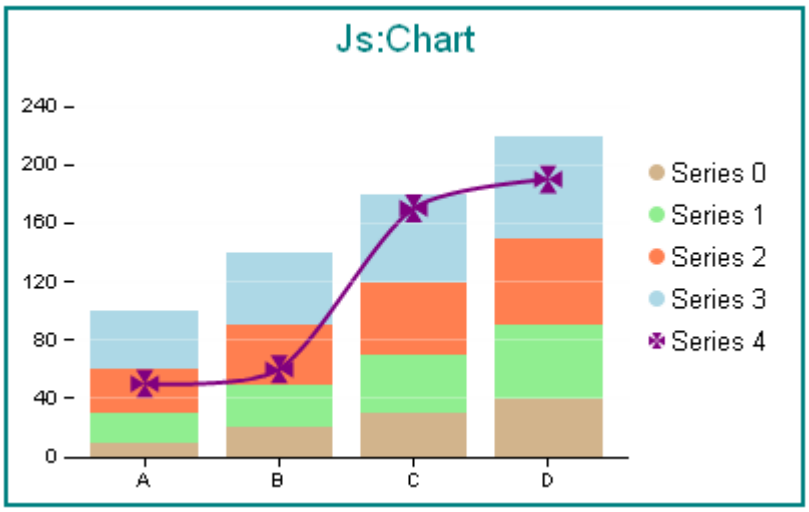
areaSeriesLayout: a string or undefined. The default value is undefined. Use 'absolute', 'percent', or 'stacked' to define the layout of any series assigned to area (series#:markerShape: 'area'). undefined = use the series layout defined in blaProperties.seriesLayout.

NOTES:

When a combination chart is defined with these properties and the series#:riserShape property, area risers are drawn first in the back, then bars, then lines in the front.

EXAMPLE:

```
data:
[[10,20,30,40],[20,30,40,50],[30,40,50,60],[40,50,60,70],[50,60,170,1
90]],
legend: {visible: true},
border: {width: 2, color: 'teal'},
title: {text: 'Js:Chart',font: '14pt Sans-Serif', color: 'teal'},
blaProperties: {orientation: 'vertical',lineConnection:'curved',
   comboCharts: {barSeriesLayout: 'stacked', lineSeriesLayout:
'absolute',areaSeriesLayout: undefined}
},
series: [
   {series: 0, color: 'tan', riserShape: 'bar'},
   {series: 1, color: 'lightgreen', riserShape: 'bar'},
   {series: 2, color: 'coral', riserShape: 'bar'},
   {series: 3, color: 'lightblue', riserShape: 'bar'},
   {series: 4, color: 'purple', riserShape: 'line',showDataValues:
true, border: {width: 2},marker: {shape: 'pirateCross', size: 14,
visible: true}}
]
```



lineConnection

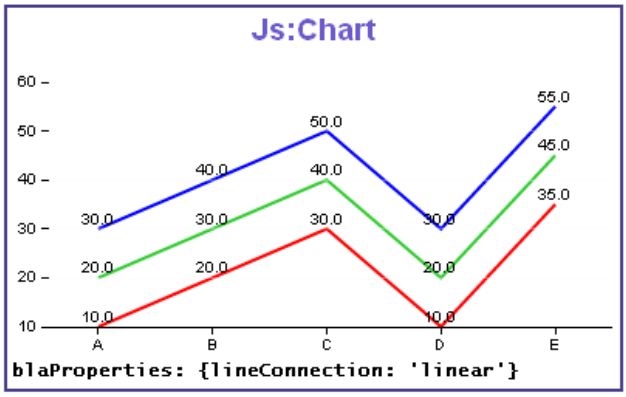
This property controls how data points are connected in line and area charts and in charts that use a combination of area and/or line risers using the comboCharts property and the series-specific riserShape property.

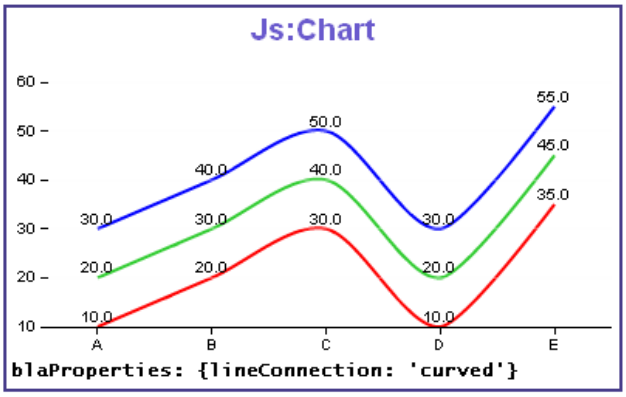
```
blaProperties: {
   lineConnection: 'string'
}
```

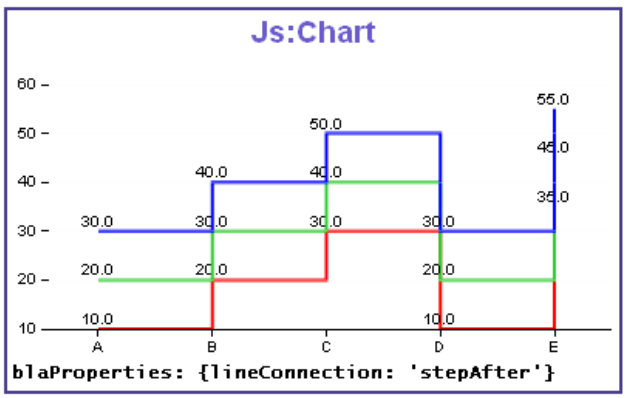
PARAMETERS:

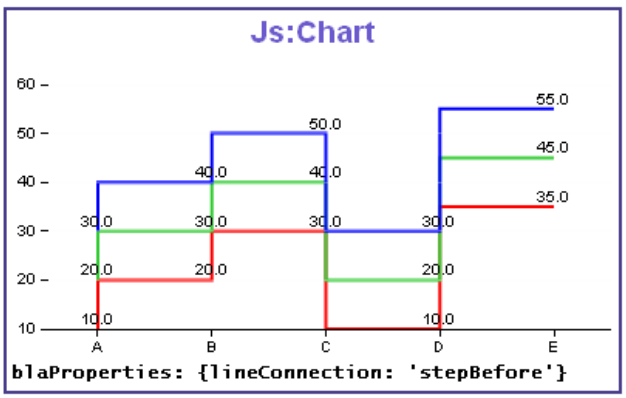
lineConnection; a string that defines the line connection: 'curved', 'linear', 'stepAfter', 'stepBefore', or 'stepBetween'. The default value is 'linear'.

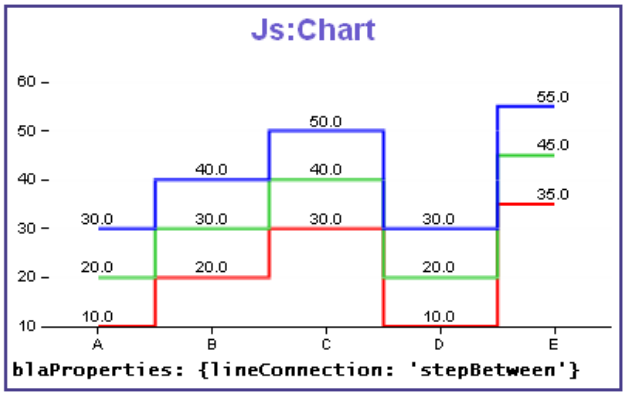
EXAMPLE:











lineFillEffect

This property will fill the space between line risers and the baseline (or sibling in a stacked / percent line) with the specified color.

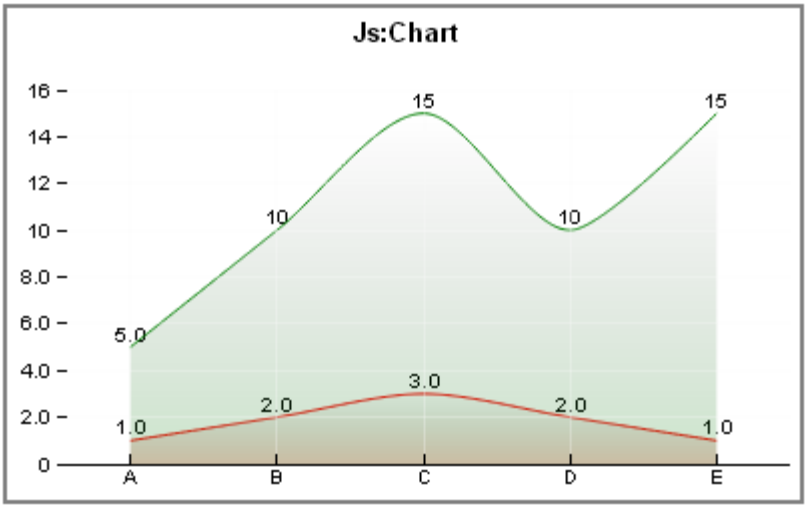
```
blaProperties: {
   lineFillEffect: 'string'
}
```

PARAMETERS:

lineFillEffect: specifies the fill effect color as one of: undefined (disabled), 'seriesAuto', 'seriesLighten', 'seriesLightenOpaque' a color defined by a keyword or numerical specification string, a gradient defined by a string, or a percent string ('-100%'...'100%'). A percent string uses the series color at the specified percentage of opacity. The default value is undefined (disabled). See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

EXAMPLE:

```
chartType: 'line',
title: {text: 'Js:Chart'},
blaProperties: {orientation: 'vertical', lineConnection:
'curved',lineFillEffect: 'seriesLighten'},
series: [
{series: 'all',border: {width:1}}
],
```



orientation

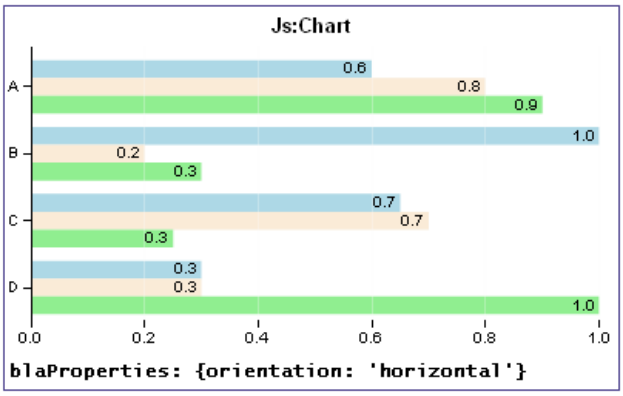
This property selects the orientation of a Bar, Line, or Area chart.

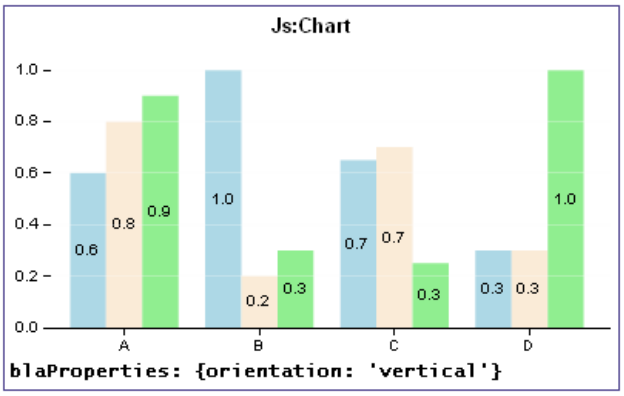
```
blaProperties: {
   orientation: 'string'
}
```

PARAMETERS:

orientation: 'horizontal' (default) or 'vertical'
'vertical' draws a vertical chart (X-axis on bottom, Y-Axis on left).
'horizontal' draws a horizontal chart (X-axis on left, Y-Axis on bottom)

EXAMPLE:





seriesLayout

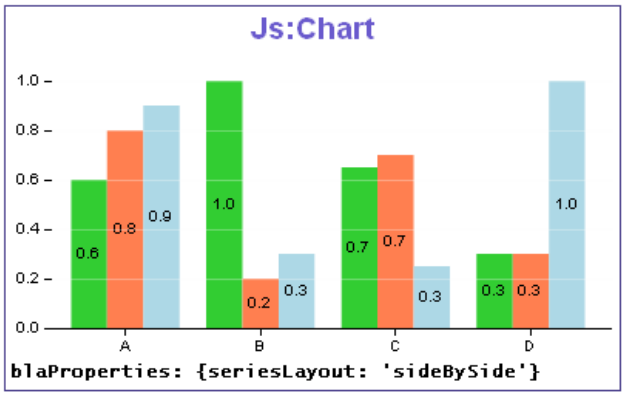
This property controls the arrangement of risers in a Bar, Line, or Area chart.

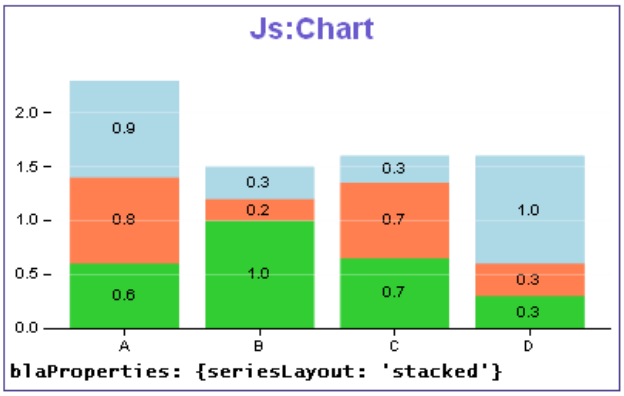
```
blaProperties: {
   seriesLayout: 'string'
}
```

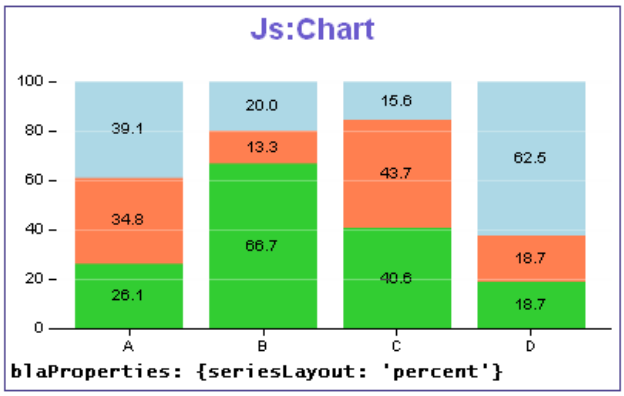
PARAMETERS:

seriesLayout; a string that defines the arrangement of risers: 'stacked', 'absolute', 'percent', or 'sideBySide' (bar charts only). The default value is 'sideBySide'.

EXAMPLES:







stackTotalLabel

These properties control the appearance of a total label to draw in stacked charts (i.e., blaProperties.seriesLayout='stacked').

```
stackTotalLabel: {
      visible: boolean,
      font: 'string',
      color: 'string',
      numberFormat: 'string' | JSON object | function()
},
```

PARAMETERS:

visible: true/false controls the visibility of the total label.

font; a string that defines the size and type face of the label. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is 'black'.

numberFormat: Use a string, JSON object, or function to define the format of the label. See Properties Overview/Formatting Numbers in Section 1 for details and examples. The default value is 'auto'.

Box Plots (boxPlotProperties)

These properties control the general appearance of a box plot. The following code segment shows the default values from properties.js:

```
boxPlotProperties: {
   hatWidth: '100%',
   drawHatAsBox: false,
   medianLine: {
         width: 1,color: 'black',dash: ''
   },
   connectorLine: {
         width: 1,color: 'black',dash: ''
   }
}
```

connectorLine

These properties control the appearance of the box plot connector lines and hats. The connector line is only visible when *drawHatAsBox* is false.

```
boxPlotProperties: {
   connectorLine: {
         width: Number,
         color: 'string',
         dash: 'string'
   }
},
```

PARAMETERS:

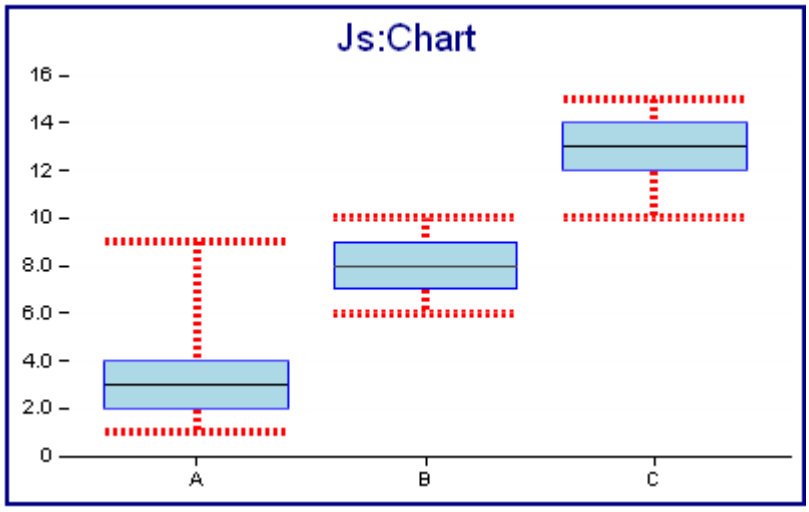
width: a number that defines the width of the line in pixels. The default value is 1.

color: a color defined by a keyword or numerical specification string. The default value is 'black'.

dash: a string that defines the line dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes.

EXAMPLE:

```
chartType: 'boxplot',
blaProperties: {orientation: 'vertical'},
boxPlotProperties: {
   connectorLine: {
         width: 4,
         color: 'red',
         dash: '2 2'
         }
   },
series: [
{series: 0, color: 'lightblue', border: {width: 2,color: 'blue'}},
],
```



drawHatAsBox

This property controls the appearance of the box plot hats (upper and lower).

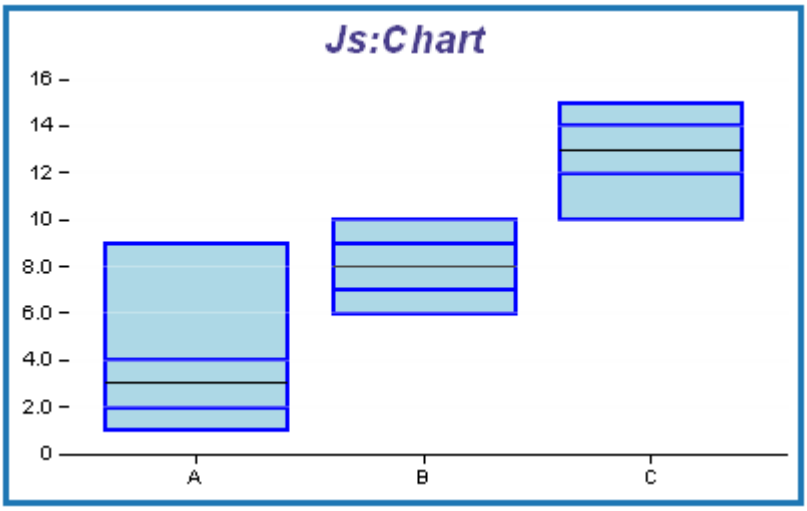
```
boxPlotProperties: {
   drawHatAsBox: boolean
},
```

PARAMETERS:

drawHatAsBox: true = draw box plot hats as boxes / false = draw normal hats. The default value is false.

EXAMPLE:

```
boxPlotProperties: {
   drawHatAsBox: true
},
```



hatWidth

This property controls the width of the hat that draws at the top and base of a box plot.

```
boxPlotProperties: {
   hatWidth: Number | 'string'
},
```

PARAMETERS:

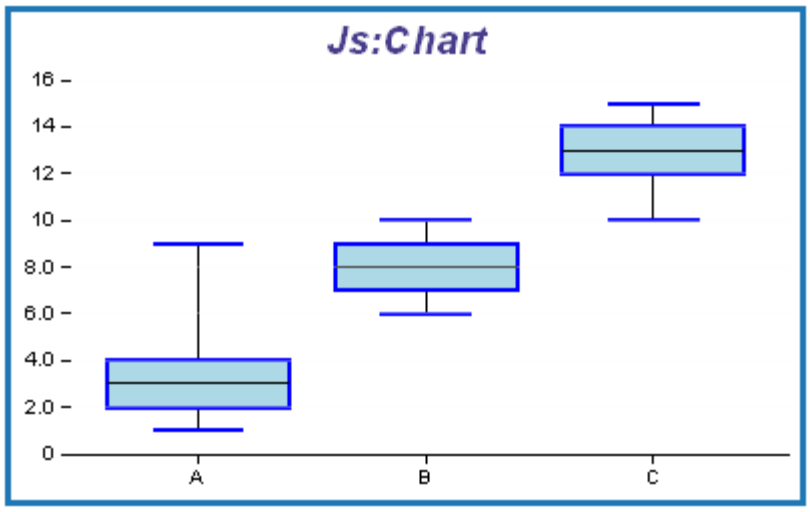
hatWidth: a string that expresses a percentage value ('0%'...'100%') or a number that defines the width of the hat in pixels. The default value is '100%' (the width of the box).

NOTES:

The series/group border properties control the color and format of the hats. You can set border:width to zero to remove the hats entirely (i.e., the same as hatWidth: '0%' or hatWidth: 0).

EXAMPLE:

```
data: [[[1,2,3,4,9],[6,7,8,9,10],[10,12,13,14,15]]],
chartType: 'boxplot',
blaProperties: {orientation: 'vertical'},
boxPlotProperties: {
   hatWidth: '50%',
   },
series: [
{series: 0, color: 'lightblue'},
],
```



medianLine

These properties control the appearance of the box plot median line.

```
boxPlotProperties: {
   medianLine: {
         width: Number,
         color: 'string',
         dash: 'string'
   }
},
```

PARAMETERS:

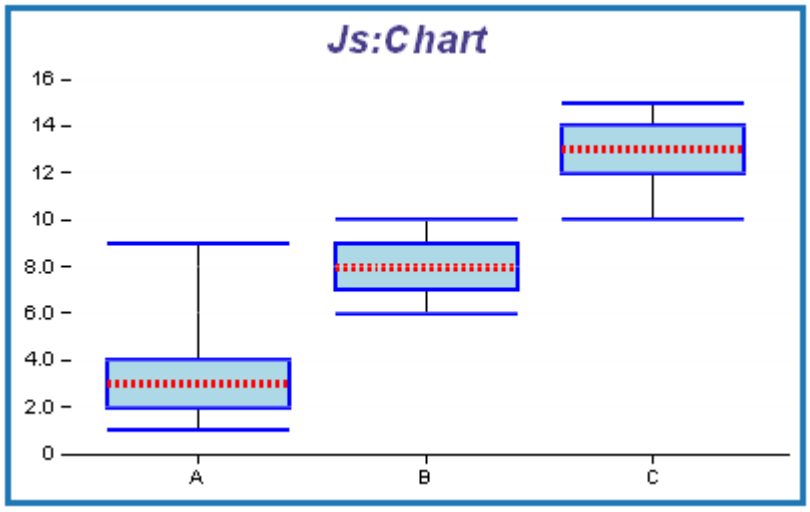
width: a number that defines the width of the line in pixels. The default value is 1.

color: a color defined by a keyword or numerical specification string. The default value is 'black'.

dash: a string that defines the line dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes.

EXAMPLE:

```
chartType: 'boxplot',
blaProperties: {orientation: 'vertical'},
boxPlotProperties: {
   medianLine: {
         width: 4,
         color: 'red',
         dash: '2 2'
         }
   },
series: [
{series: 0, color: 'lightblue', border: {width: 2,color: 'blue'}},
],
```



Bullet Charts (bulletProperties)

This property controls the general appearance of a bullet chart.

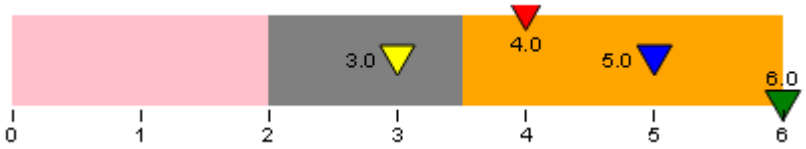
```
bulletProperties: {
   drawFirstValueAsBar: boolean
},
```

PARAMETERS:

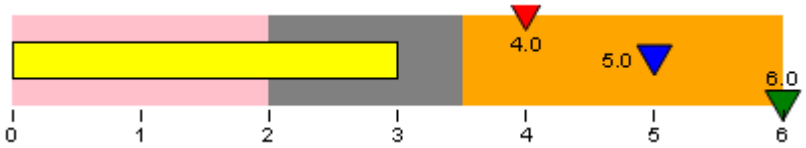
drawFirstValueAsBar: true = draw first data value as a bar, false = draw first value as a marker. The default value is true.

EXAMPLE:

```
data: [[3, 4, 5, 6]],
chartType: 'bullet',
bulletProperties: {drawFirstValueAsBar: false},
```




```
data: [[3, 4, 5, 6]],
chartType: 'bullet',
bulletProperties: {drawFirstValueAsBar: true},
```



Funnel Charts (funnelProperties)

These properties control the general layout of a funnel chart. The following code segment shows the default values.

```
funnelProperties: {
   topWidth: '100%',
   baseWidth: '20%',
   riserGap: 0,
   groupLabel: {
         visible: false,
         font: '10pt Sans-Serif',
         color: 'black'
   }
},
```

baseWidth

This property controls the width of the base of the funnel.

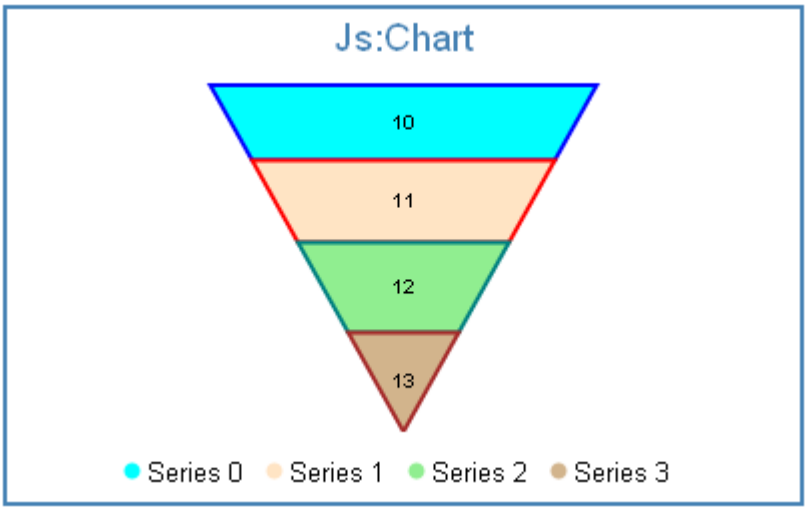
```
funnelProperties: {
   baseWidth: Number | 'string'
},
```

PARAMETERS:

baseWidth; a number or string that defines the width of the base of the funnel. A number defines the width in pixels. A string must represent a percentage value (i.e., '0%' ... '100%'). The default value is '20%'.

EXAMPLE:

```
data: [[10,11,12,13]],
chartType: 'funnel',
funnelProperties: {
   topWidth: '50%',
   baseWidth: 1
},
series: [
{series: 0, color: 'cyan', showDataValues: true, border: {width: 2,
color: 'blue'}},
{series: 1, color: 'bisque', showDataValues: true, border: {width: 2,
color: 'red'}},
{series: 2, color: 'lightgreen', showDataValues: true, border:
{width: 2, color: 'teal'}},
{series: 3, color: 'tan', showDataValues: true, border: {width: 2,
color: 'brown'}},
]
```



groupLabel

These properties control the visibility and format of the group label in a funnel chart.

```
funnelProperties: {
   groupLabel: {
         visible: boolean,
         font: 'string',
         color: 'string'
   }
},
```

PARAMETERS:

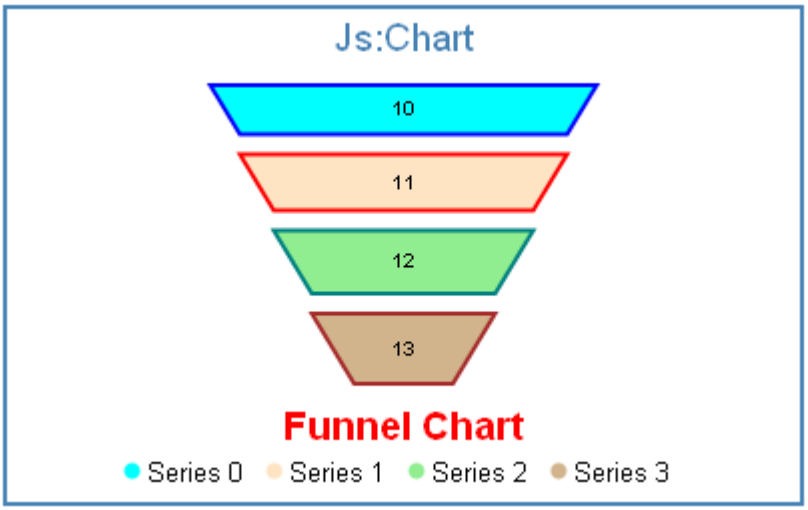
visible: true/false = show/hide group label. The default value is false.

font; a string that defines the size and type face of the label. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is 'black'.

EXAMPLE:

```
data: [[10,11,12,13]],
chartType: 'funnel',
funnelProperties: {
   topWidth: '50%', baseWidth: 1,riserGap: 10
   groupLabel: {
         visible: true,font: 'bold 14pt Sans-Serif',color: 'red'
   }
},
groupLabels: ["Funnel Chart"],
series: [
{series: 0, color: 'cyan', border: {width: 2, color: 'blue'}},
{series: 1, color: 'bisque', border: {width: 2, color: 'red'}},
{series: 2, color: 'lightgreen', border: {width: 2, color: 'teal'}},
{series: 3, color: 'tan', border: {width: 2, color: 'brown'}},
]
```



riserGap

This property controls the empty space between layers in the funnel.

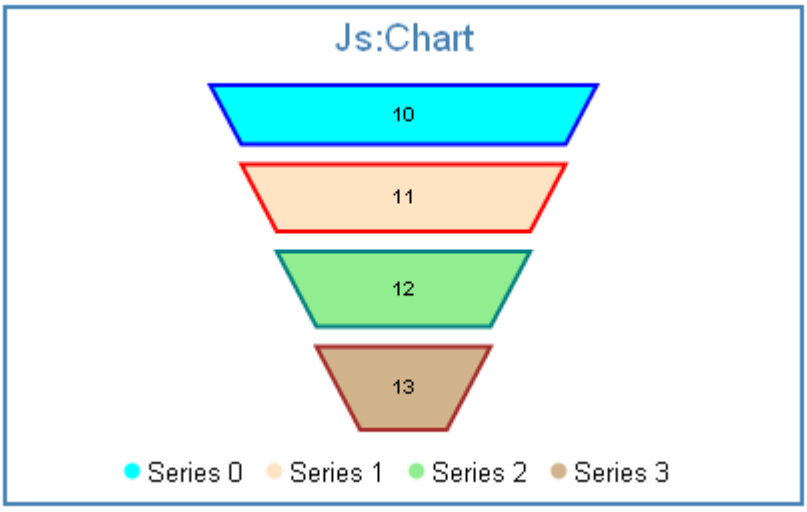
```
funnelProperties: {
   riserGap: Number | 'string'
},
```

PARAMETERS:

riserGap; a number or string that defines the width of the riser gap between funnel layers. A number defines the width of the gap in pixels. A string must represent a percentage value '0%' ... '100%' (of average riser height). For example, riserGap: '10%' will use 10% of each funnel layer as white space between each segment. The default value is zero.

EXAMPLE:

```
data: [[10,11,12,13]],
chartType: 'funnel',
funnelProperties: {
   topWidth: '50%',
   baseWidth: 1,
   riserGap: 10
},
series: [
{series: 0, color: 'cyan', border: {width: 2, color: 'blue'}},
{series: 1, color: 'bisque', border: {width: 2, color: 'red'}},
{series: 2, color: 'lightgreen', border: {width: 2, color: 'teal'}},
{series: 3, color: 'tan', border: {width: 2, color: 'brown'}},
]
```



topWidth

This property controls the width of the top of the funnel.

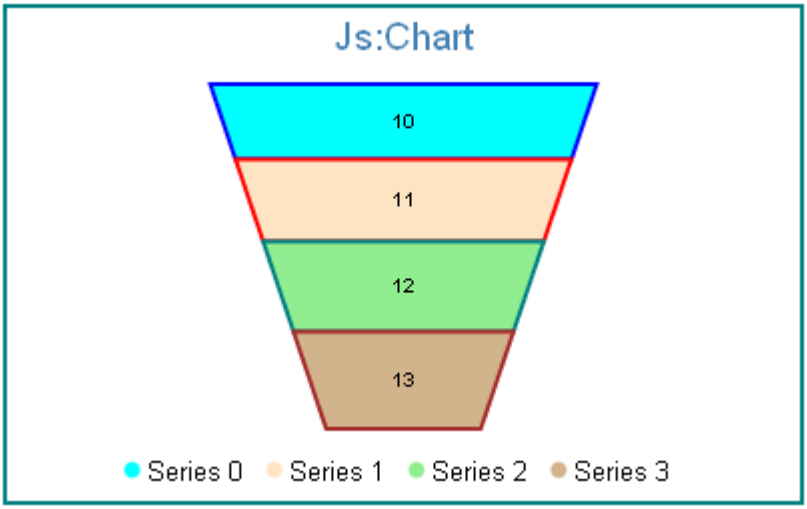
```
funnelProperties: {
   topWidth: Number | 'string'
},
```

PARAMETERS:

topWidth; a number or string that defines the width of the top of the funnel. A number defines the width in pixels. A string must represent a percentage value (i.e., '0%' ... '100%'). The default value is '100%'.

EXAMPLE:

```
data: [[10,11,12,13]],
chartType: 'funnel',
funnelProperties: {
   topWidth: '50%',
},
series: [
{series: 0, color: 'cyan', border: {width: 2, color: 'blue'}},
{series: 1, color: 'bisque', border: {width: 2, color: 'red'}},
{series: 2, color: 'lightgreen', border: {width: 2, color: 'teal'}},
{series: 3, color: 'tan', border: {width: 2, color: 'brown'}},
]
```



Gantt Charts (ganttProperties)

These properties define the general appearance and layout of a gantt chart.

```
ganttProperties: {
   startTime: 'string' | undefined,
   stopTime: 'string' | undefined,
   interval: 'string' | undefined,
   labelFormat: 'string' | undefined,
   durationValues: boolean,
   staggerRisers: boolean
},
```

PARAMETERS:

startTime: a string identifying the start time. It can be almost any kind of string denoting time, mixing hours (xx:xx), days of the week, dates, months and years. Examples: "Mon", "1 July 20", "June 2001", "8:00", "6/10/01", "Thu, 4 Apr 1976 14:30", "2 0:00" (February 1), "1 1 0:00" (January 1). The default value is undefined.

stopTime: a string identifying the start time. As with startTime, the string can be almost any date/time format. The default value is undefined.

interval: a string that specifies the time interval: 'seconds', 'minutes', 'hours', 'days', 'weeks', 'months', 'quarters', or 'years'. This property defines the time gap between each group label. If undefined, the library will assign an interval based on the startTime, stopTime, and the number of groups. The default value is undefined.

labelFormat: a string that defines how a time/date number is converted into a label. If undefined, the library will we use a default formats based on the interval. See http://code.google.com/p/datejs/wiki/FormatSpecifiers for available format options. The default value is undefined.

durationValues: true/false controls the use of the second value in the data set that defines each riser. If false, the second data point is treated as a stop time. If true, the second data point is treated as a duration. The default value is false.

staggerRisers: If false, risers within same group are drawn in one column, stacked above (or beside) each other. In this mode, consecutive risers will be forced to have consecutive start / stop times. If true, risers within same group are drawn beside each other. In this mode, risers can have any arbitrary start / stop times. The default value is true.

Gauges (gaugeProperties)

A gauge chart represents one or more values as needles or a single needle and markers on a circular surface. This chart type is typically used by executive dashboard applications.

The following properties control the basic layout of gauge charts. This code segment shows the default settings from properties.js.

```
gaugeProperties: {
   startAngle: 135,
   endAngle: 45,
   secondaryNeedlesAsMarkers: false,
   groupLabel: {
         visible: false,
         font: '10pt Sans-Serif',
         color: 'black'
   },
   totalLabel: {
         visible: false,
         font: '10pt Sans-Serif',
         color: 'black',
         numberFormat: 'auto'
   },
   fill: {
         color: 'transparent'
   },
   needleBase: {
         size: "6%",
         color: '#1f77b4',
         border: {
               width: 0,
               color: 'transparent',
               dash: ''
         }
   },
   axisWidth: "22%",
   axisTickLength: "30%",
   outerBorder: {
         width: '10%',
         fill: {
               color: '#1f77b4'
         },
         border: {
               width: 0,
               color: 'transparent',
               dash: ''
         }
   }
},
```

The numeric yaxis properties control the range and format of values, the format of grid lines, and color bands shown on the gauge axis.

axisTickLength

This property defines with length of axis tick marks.

```
gaugeProperties: {
   axisTickLength: Number | 'string'
},
```

PARAMETERS:

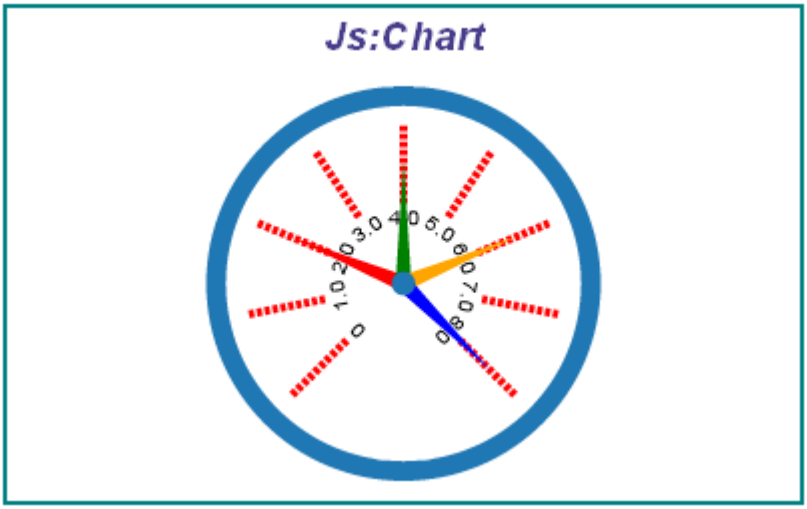
axisTickLength: a number or string that defines the length of tick marks. A number defines the length in pixels. A string must express a percent value that defines the width as a percentage of the overall gauge. The default value is '30%'.

NOTES:

Use yaxis:majorGrid to format the tick marks.

EXAMPLE:

```
gaugeProperties: {
   axisTickLength: '50%',
},
yaxis: {
   majorGrid: {
         lineStyle: {
         width: 4,
               color: 'red',
               dash: '2 2'
         },
   }
},
```



axisWidth

This property defines the width of the gauge axis.

```
gaugeProperties: {
   axisWidth: number | 'string'
},
```

PARAMETERS:

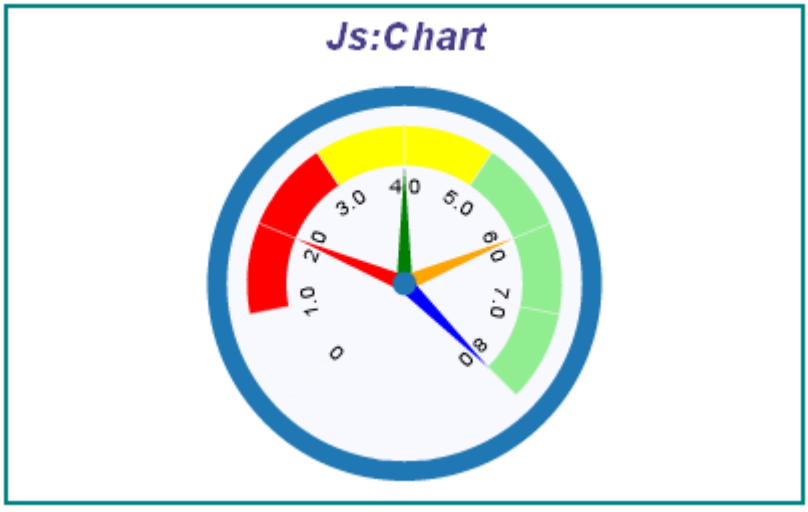
axisWidth: a number or string that defines the axis width. A number defines the width in pixels. A string must express a percent value that defines the width as a percentage of the overall gauge. The default value is '22%'.

NOTES:

Use yaxis:colorBands (as shown in the examples) to apply color to the gauge axis.

EXAMPLE:

```
chartType: 'gauge',
gaugeProperties: {
   axisWidth: '25%',
},
yaxis: {
   colorBands: [
         {start: 1,stop: 3,color: 'red'},
         {start: 3,stop: 5,color: 'yellow'},
         {start: 5,stop: 8,color: 'lightgreen'},
   ],
},
```



fill

This property controls the fill color of the interior of the gauge face.

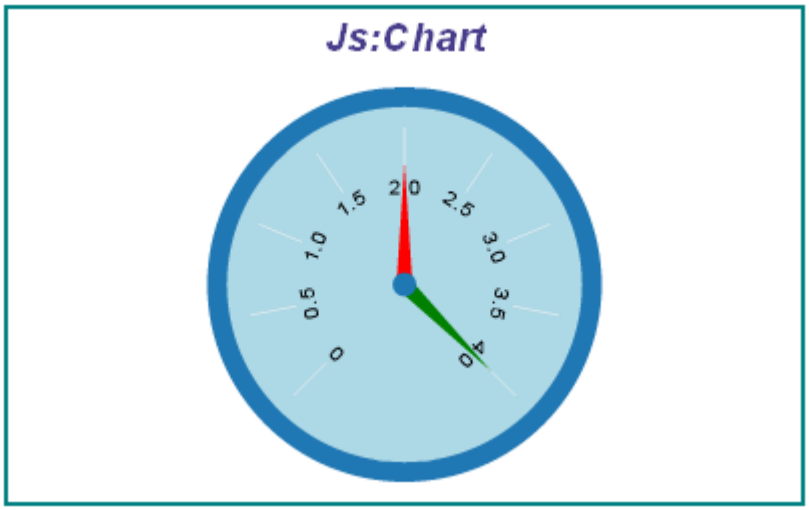
```
gaugeProperties: {
   fill: {
         color: 'string' | JSON Object
   },
},
```

PARAMETERS:

color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is transparent. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

EXAMPLE:

```
chartType: 'gauge',
gaugeProperties: {
   fill: {color: 'lightBlue'}
},
```



groupLabel

These properties control the visibility and format of the group label in a gauge chart.

```
gaugeProperties: {
   groupLabel: {
         visible: false,
         font: '10pt Sans-Serif',
         color: 'black'
   },
```

PARAMETERS:

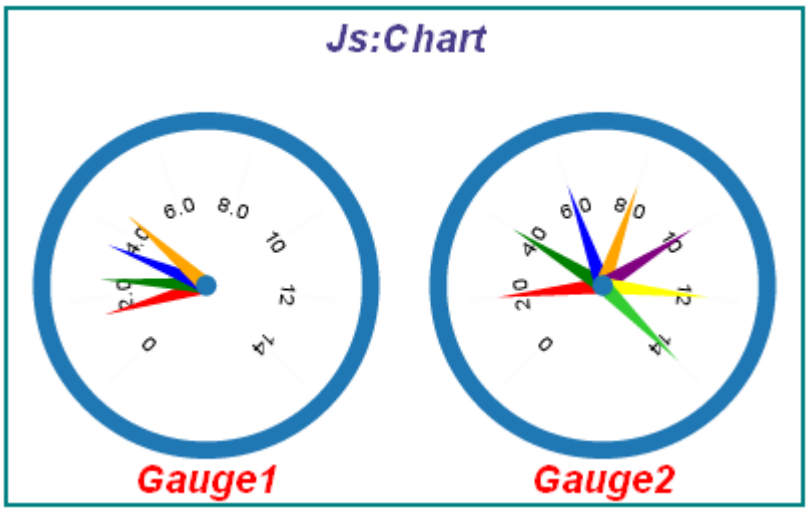
visible: true/false = show/hide group label. The default value is false.

font; a string that defines the size and type face of the label. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is 'black'.

EXAMPLE:

```
groupLabels: "Gauge1 Gauge2",
chartType: 'gauge',
gaugeProperties: {
   secondaryNeedlesAsMarkers: false,
   groupLabel: {
         visible: true,
         font: 'bold italic 14pt Sans-Serif',
         color: 'red'
   },
}
```



needleBase

These properties control the format of the base of the gauge needle.

```
gaugeProperties: {
   needleBase: {
         size: number | 'string',
         color: 'string' | JSON object,
         border: {
               width: number,
               color: 'string',
               dash: 'string'
         }
   },
},
```

PARAMETERS:

size: a number or string that defines the radius of the base of the gauge needle. A number defines the radius in pixels. A string must express a percent value that defines the radius as a percentage of the overall gauge. The default value is '6%'.

color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is '#1f77b4'. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

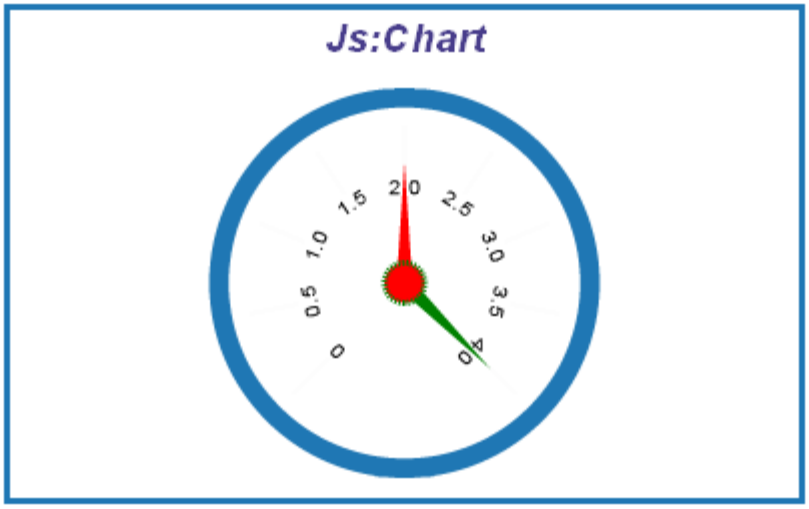
border/width: a number that defines the width of the border around the needle base. The default value is zero (no border).

border/color: a color defined by a keyword or numerical specification string. The default value is 'transparent' (no border).

border/dash: a string that defines the border dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes.

EXAMPLE:

```
gaugeProperties: {
   needleBase: {size: 10,color: 'red',
         border: {width: 3,color: 'green',dash: '1 1'}}
},
```



outerBorder

These properties control the format of a gauge's outer border.

```
gaugeProperties: {
   outerBorder: {
         width: number | 'string'
         fill: {
               color: 'string' | JSON Object
         },
         border: {
               width: number,
               color: 'string',
               dash: 'string'
         }
   }
},
```

PARAMETERS:

width: a number or a percent string. A number defines the width (in pixels) of the outer border ring. A percent string defines the width of the outer border as a percentage of the gauge size. The default value is '10%'.

fill/color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is '#1f77b4'. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

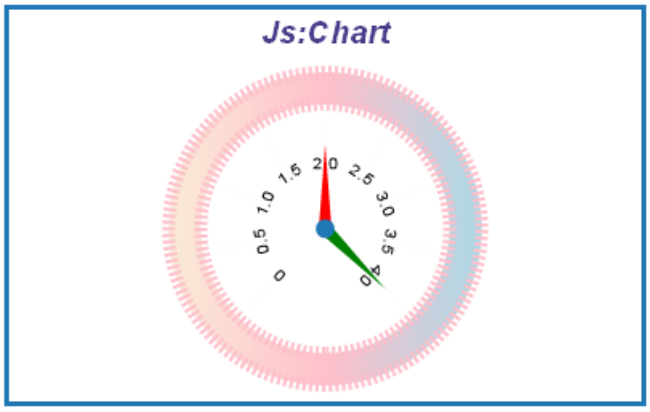
border/width: a number that defines the width (in pixels) of the border of the outer border. The default value is zero.

border/color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is transparent.

border/dash: a string that defines the border dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes.

EXAMPLE:

```
gaugeProperties: {
   outerBorder: {width: 20, fill: {color: 'linear-
gradient(0%,0,100%,0, 0 antiquewhite, 0.5 pink, 1 lightblue)'},
   border: {width: 8,color: 'pink',dash: '2 2'}}
},
```



secondaryNeedlesAsMarkers

When a gauge includes multiple needles, use the property to draw secondary needles as markers.

```
gaugeProperties: {
   secondaryNeedlesAsMarkers: boolean,
},
```

PARAMETERS:

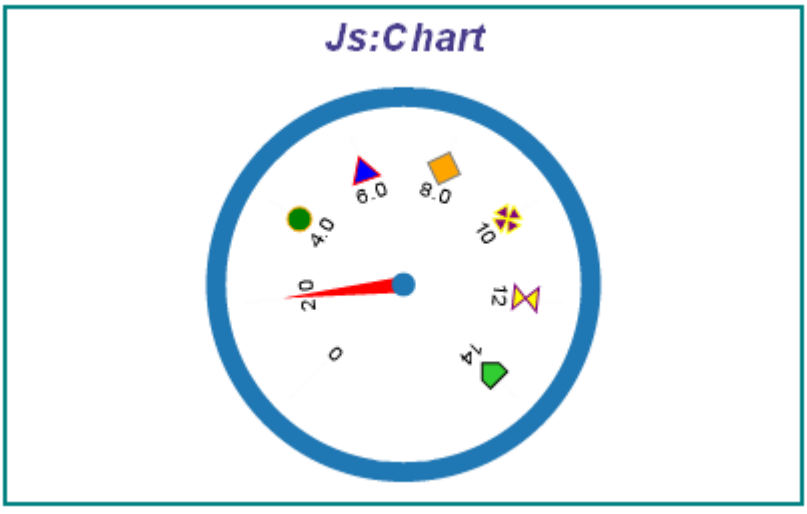
secondaryNeedlesAsMarkers: true = show secondary needles as markers, false = draw all values as multiple needles. The default value is false (multiple needles).

NOTES:

When secondaryNeedlesAsMarkers is false, use the series:marker property to control the size and format of markers.

EXAMPLE:

```
chartType: 'gauge',
gaugeProperties: {
   secondaryNeedlesAsMarkers: true
},
series: [
{series: 0, color: 'red',marker: {shape: 'square', size: 12, border:
{width: 1, color: 'black'}}},
{series: 1, color: 'green',marker: {shape: 'circle', size: 12,
border: {width: 1, color: 'orange'}}},
{series: 2, color: 'blue',marker: {shape: 'triangle', size: 12,
border: {width: 1, color: 'red'}}},
{series: 3, color: 'orange',marker: {shape: 'diamond', size: 12,
border: {width: 1, color: 'grey'}}},
{series: 4, color: 'purple',marker: {shape: 'pirateCross', size: 12,
border: {width: 1, color: 'yellow'}}},
{series: 5, color: 'yellow',marker: {shape: 'hourglass', size: 12,
border: {width: 1, color: 'purple'}}},
{series: 6, color: 'limegreen',marker: {shape: 'house', size: 12,
border: {width: 1, color: 'black'}}},
]
```



startAngle / endAngle

These properties control the start/end angle of the gauge value bands. Angles are defined in degrees. Positive angles draw clockwise, negative angles draw counter clockwise. Zero will start or end the band at the 3 o'clock position. 90 will start or end the band at the 6 o'clock position. -90 specifies the 12 o'clock position.

```
gaugeProperties: {
   startAngle: number,
   endAngle: number
},
```

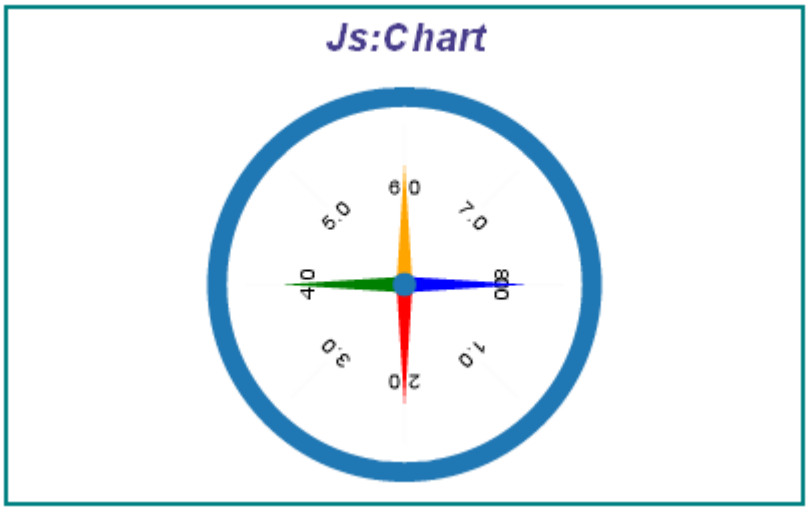
PARAMETERS:

startAngle: a number that defines the angle (in degrees) to start the gauge value band. The default value is 135.

endAngle: a number that defines the angle (in degrees) to end the gauge band value. The default value is 45.

EXAMPLE:

```
chartType: 'gauge',
gaugeProperties: {
   startAngle: 0,
   endAngle: 0
},
```



totalLabel

These properties control the visibility and format of the total label in a gauge chart.

```
gaugeProperties: {
   totalLabel: {
         visible: false,
         font: '10pt Sans-Serif',
         color: 'black'
   },
```

PARAMETERS:

visible: true/false = show/hide total label. The default value is false.

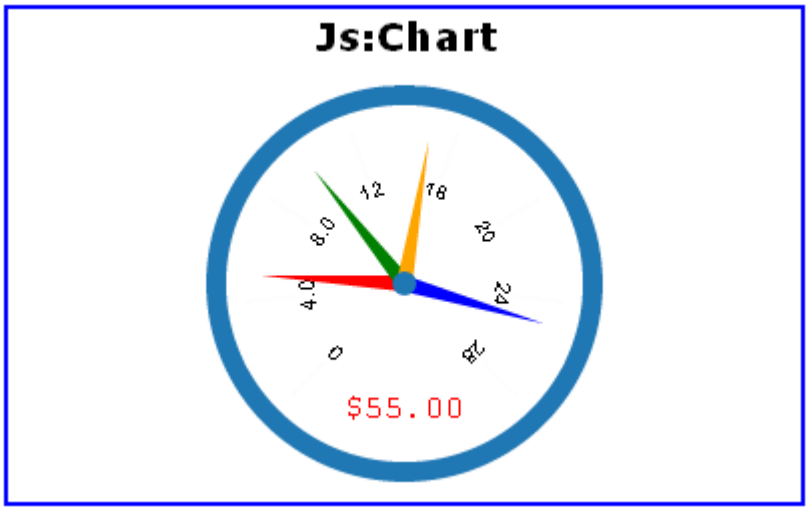
font; a string that defines the size and type face of the label. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is 'black'.

numberFormat; Use a string, JSON object, or function to define the format of the label. See Properties Overview/Formatting Numbers in Section 1 for details and examples. The default value is 'auto'.

EXAMPLE:

```
chartType: 'gauge',
gaugeProperties: {
   totalLabel: {visible: true,
   font: '12pt Courier',
   color: 'red',
   numberFormat: {mode: 'currency'}
   }
},
```



Histogram Charts (histogramProperties)

A histogram is a graphical representation that shows a visual impression of the distribution of data. These properties control the distribution of data.

```
histogramProperties: {
   binCount: Number | undefined,
   binSize: Number | Array | undefined
   startBinValue: Number | undefined
   },
```

PARAMETERS:

binCount: number of group labels to draw or undefined (automatic). The default value is undefined.

binSize: a number, an array of numbers (for variable bin sizes), or undefined (automatic). If *binSize* is an array, *binCount* is ignored and assumed to be the length of the *binSize* array. The default value is undefined.

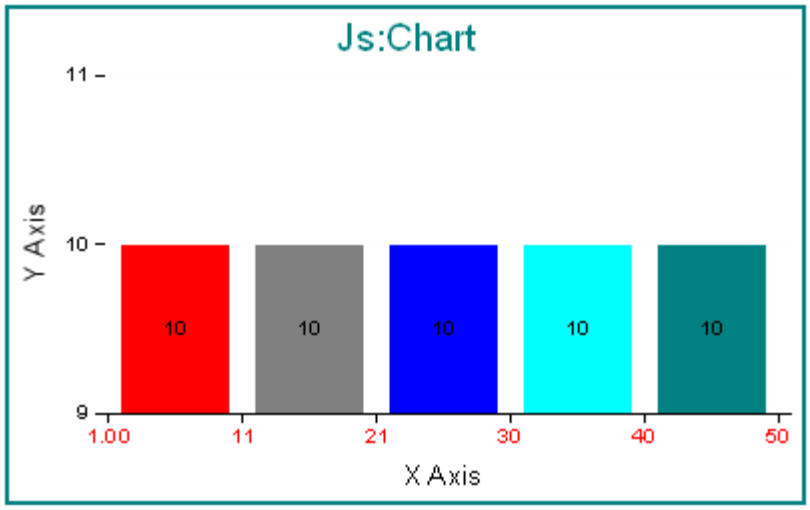
startBinValue: a number to use as the first bin or undefined (automatic). The default value is undefined.

NOTES:

- Use blaProperties.orientation to draw the histogram in vertical or horizontal format.
- blaProperties.barGroupGapWidth can be used to modify the width and space between histogram risers.
- The series color and border properties can be used to format histogram risers.
- Use yaxis properties to control the format of Y-axis title and labels.
- Use xaxis properties to control the format of X-axis title and labels. Axis labels are calculated. They are not defined by the groupLabels property.
- In a vertical histogram, the Y-axis is on the left side of the chart and the ordinal X-axis is drawn on the bottom of the chart. In a horizontal histogram, the X-axis is on the left side of the chart and the Y-axis axis is drawn on the bottom of the chart.

EXAMPLE:

```
data = [[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
   20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,
   36,37,38,39,40,41,42,43,44,45,46,47,48,49,50]],
chartType: 'histogram',
blaProperties: {orientation: 'vertical'},
histogramProperties: {
   binCount: 5,
},
series: [
{series: 0, group: 0, color: 'red'},
{series: 0, group: 1, color: 'grey'},
{series: 0, group: 2, color: 'blue'},
{series: 0, group: 3, color: 'cyan'},
{series: 0, group: 4, color: 'teal'},
]
```



Parabox Chart (paraboxProperties)

This property defines the active group in a parabox chart.

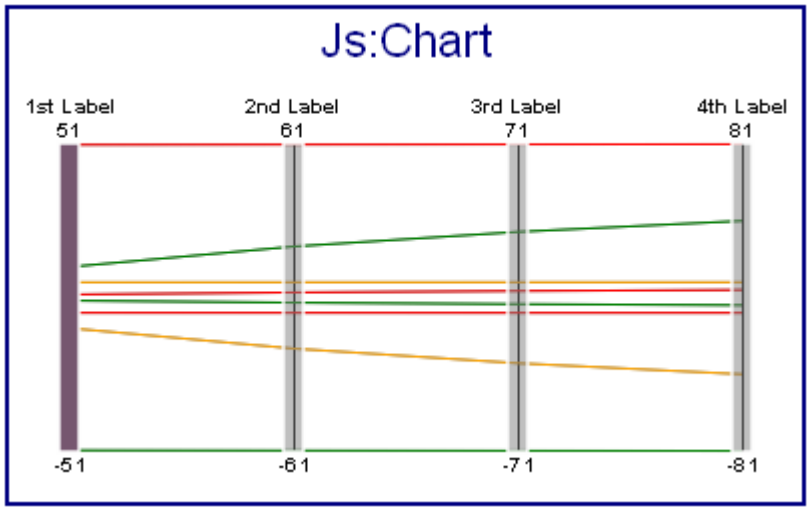
```
paraboxProperties: {
  activeGroup: undefined
},
```

PARAMETERS:

activeGroup: a string that defines the group (specified by its label) that is active. The active group defines coloring for the lines that comes through it.

EXAMPLE:

```
data: [[1,2,3,4],[-1,-2,-3,-4],[5,6,7,8],[-5,-6,-7,-
8],[10,20,30,40],[-10,-20,-30,-40],[50,60,70,80],[-50,-60,-70,-80]],
border: {width: 2, color: 'navy'},
chartType: 'parabox',
groupLabels: ['1st Label','2nd Label','3rd Label','4th Label'],
paraboxProperties: {activeGroup: '1st Label'},
```



Pie Charts (pieProperties)

These properties control the basic layout of pie charts. The following code segment shows the default values:

```
pieProperties: {
   groupPiesBySelection: false,
   holeSize: 0,
   rotation: 0,
   skew: 0,
   label: {visible: false,font: '10pt Sans-Serif',color: 'black'},
   totalLabel: {
         visible: false,
         font: '10pt Sans-Serif',
         color: 'black',
         numberFormat: 'auto'
   },
   feelerLine: {
         visible: true,
         width: 1,
         color: 'black',
         dash: ''
   },
   otherSlice: {
         threshold: undefined,
         legendLabel: 'Other',
         color: 'grey',
         border: {width: 1,color: 'transparent',dash: ''},
         showDataValues: true,
         marker: {
               shape: 'circle',
               border: {width: 0,color: 'transparent',dash: ''}
         }
   },
   explodeClick: {
         enabled: false,
         duration: 700,
         distance: 25,
         limitExplodeCount: false
   }
}
```

Also see the series-specific properties to:

- border: draw borders around pie slices
- explodeSlice: explode a slice from a pie
- deleteSlice: delete a slice from a pie
- For charts with multiple pies, use the chartsPerRow property to defined the number of pie charts to draw in a horizontal row.
- Set the dataLabels: position property to 'outside' to draw data text labels outside pie slices with feeler lines.
- Set the dataLabels: displayMode property to "%" to show slice values as a percent of the total pie.

explodeClick

These properties enable/disable and define animation when a chart user clicks on a pie slice.

```
pieProperties: {
   explodeClick: {
         enabled: boolean,
         duration: Number,
         distance: Number,
         limitExplodeCount: boolean
   }
}
```

PARAMETERS:

enabled: true/false. true = enable explode slice on mouse click. false = disable explode slice on mouse click. The default value is false.

duration: duration of the animation. Smaller values = quicker animation. Larger values = slower animation. The default value is 700.

distance: distance to explode the clicked slice from the pie. The default value is 25.

limitExplodeCount: true/false. true = single exploded slice. false = allow multiple exploded slices. The default value is false.

feelerLine

When dataLabels: position is set to 'outside', this property controls the appearance of feeler lines between the pie chart and data text labels.

```
pieProperties: {
   feelerLine: {
         visible: boolean,
         width: number,
         color: 'string',
         dash: 'string'
   },
}
```

PARAMETERS:

visible; true/false controls the visibility of the feeler lines. The default value is true.

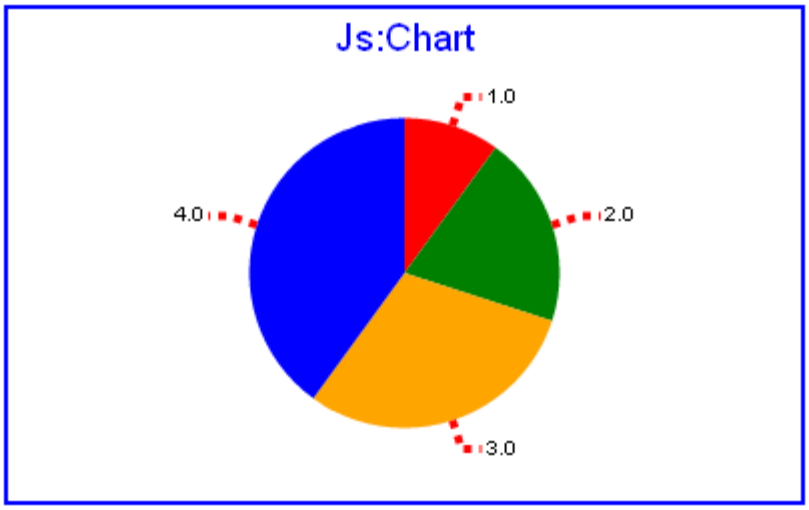
width; a number that defines the width of the feeler lines. The default value is 1.

color; a string that defines the color of the feeler lines. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

dash; a string that defines the dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

EXAMPLE:

```
pieProperties: {
   feelerLine: {visible: true,width: 4,color: 'red',dash: '4 4'}
},
dataLabels: {position: 'outside', visible: true}
```



groupPiesBySelection

When a sub-selection is applied to a pie via colorMode.mode='bySeriesSelection', this property controls the appearance of the selected slice segments.

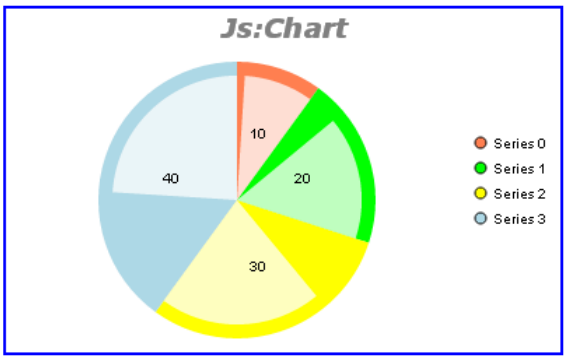
```
pieProperties: {
   groupPiesBySelection: boolean
}
```

PARAMETERS:

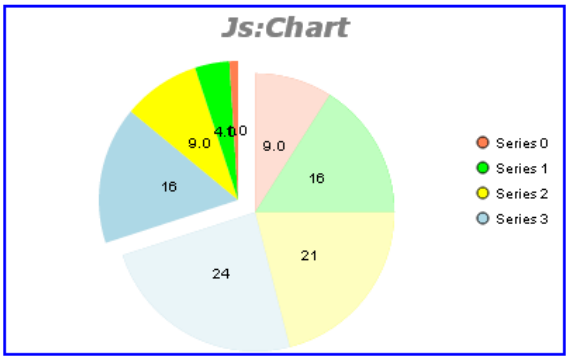
groupPiesBySelection: true/false. true = draw sub-selections together and slightly exploded. false = draw sub-selections on top of the selected slice using dataSelection.unselectedColor.

EXAMPLE:

```
data: [[10,20,30,40]],
chartType: 'pie',
colorMode: {
   mode: 'bySeriesSelection',
   data: [['10%', '20%', '30%', '40%']]
   },
legend: {visible: true},
pieProperties: {groupPiesBySelection: false},
dataSelection: {unselectedColor: 'rgba(255, 255, 255, 0.75)'},
series: [{series: 0, color: 'coral'},{series: 1, color: 'lime'},
{series: 2, color: 'yellow'},{series: 3, color: 'lightblue'}]
```




```
pieProperties: {groupPiesBySelection: true},
```



holesize

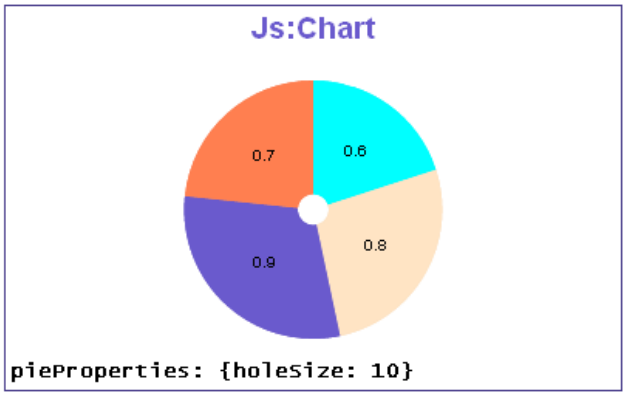
This property defines the size of a transparent circle to draw in the center of a pie chart.

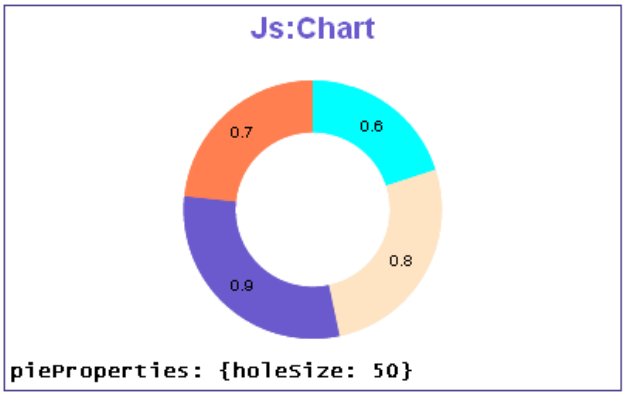
```
pieProperties: {
   holesize: Number | string
}
```

PARAMETERS:

holesize: a number of pixels or a percentage string (e.g., '10%'). The default value is zero.

EXAMPLE:





label

This property controls the visibility and format of the pie chart label.

```
pieProperties: {
   label: {
         visible: boolean,
         font: 'string',
         color: 'string'
   }
}
```

PARAMETERS:

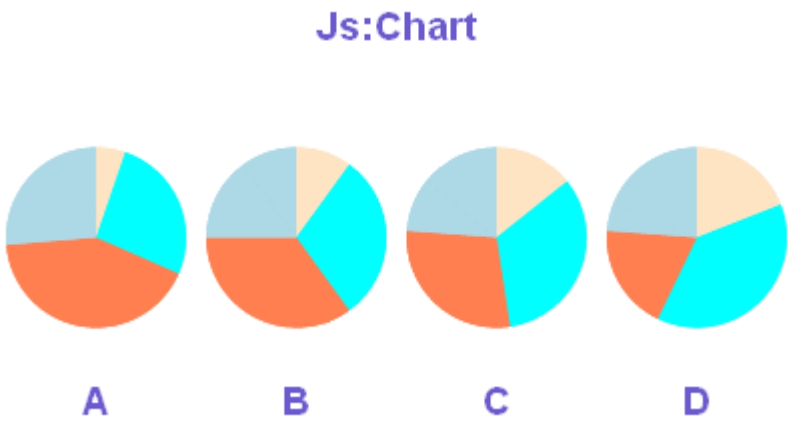
visible; true/false controls the visibility of the pie chart label. The default value is false.

font; a string that defines the size and type face of the pie chart label. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color; a string that defines the color of the pie chart label. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

EXAMPLE:

```
pieProperties: {
   label: {
         visible: true,
         font: 'Bold 14pt Sans-Serif',
         color: 'DarkSlateBlue'
   }
}
```



otherSlice

This property controls the visibility and format of pie slices that are below a specified value.

```
pieProperties: {
   otherSlice: {
         threshold: Number | 'string' | undefined,
         legendLabel: 'string',
         color: 'string' | JSON Object,
         border: {
               width: Number,
               color: 'string',
               dash: 'string'
         }
         showDataValues: boolean,
         marker: {
               shape: 'string',
               border: {
                     width: Number,
                     color: 'string',
                     dash: 'string'
               }
         }
   }
}
```

PARAMETERS:

threshold: a number (absolute data value), a percent string, a 'top n' string, or undefined (disable other slice). If threshold is a number, any values below this threshold are combined into the other slice. If threshold is a percent string (e.g., '2%'), any slices whose overall contribution to the pie is less than this value are merged into the other slice. If the string begins with 'top' followed by a number (e.g., 'top 2'), slices with the highest "n" values will be drawn as individual slices. All other slices will be grouped into the other slice. If threshold is undefined or null, the other slice is disabled. The default value is undefined.

legendLabel: A string identifying the label to show in the legend for the other slice. The default value is 'Other'.

color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object controls the color of the other slice. The default value is grey. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

border/width: The width of the other slice border in pixels. The default value is 1.

border/color: a color defined by a keyword or numerical specification string controls the color of the other slice border. The default value is 'transparent'.

border/dash: a string that defines the border's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

showDataValues: true/false = show/hide a data text label for the other slice. The default value is true.

marker/shape: a string that defines the shape of the marker to show in the legend area for the other slice: arrow, bar, circle, cross, diamond, fiveStar, hexagon, hourglass, house, pin, pirateCross, plus, rectangle, sixStar, square,

thinPlus, tick, or triangle. The default value is 'circle'. Note that bar, cross, and tick markers require a border width and color.

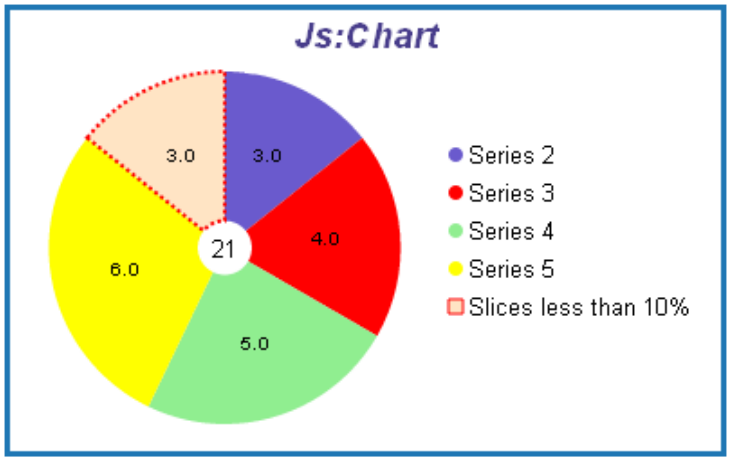
marker/border/width: The width of the marker border in pixels. The default value is 1.

marker/border/color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. The default value is 'transparent'.

marker/border/dash: a string that defines the border's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

EXAMPLE:

```
data = [[1,2,3,4,5,6]],
chartType: 'pie',
pieProperties: {
   holeSize: 15,
   totalLabel: {visible: true},
   otherSlice: {
         threshold: '10%',
         legendLabel: 'Slices less than 10%',
         color: 'bisque',
         border: {width: 2,color: 'red',dash: '2 2'},
         showDataValues: true,
         marker: {
               shape: 'square',
               border: {width: 1,color: 'red',dash: ''}
         }
   }
},
legend: {visible: true},
series: [
   {series: 0, color: 'cyan'},
   {series: 1, color: 'pink'},
   {series: 2, color: 'slateblue'},
   {series: 3, color: 'red'},
   {series: 4, color: 'lightgreen'},
   {series: 5, color: 'yellow'},
   ]
```



rotation

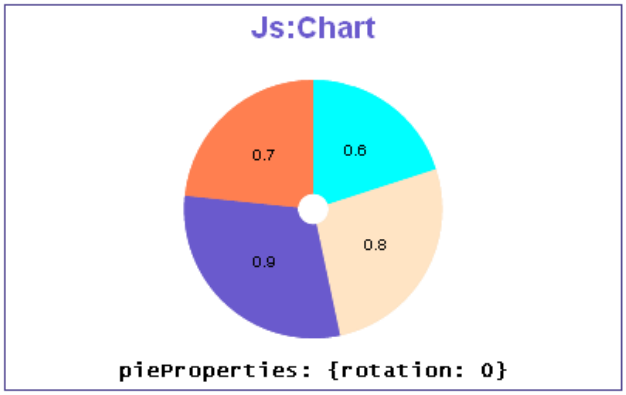
This property rotates a pie chart a specified number of degrees.

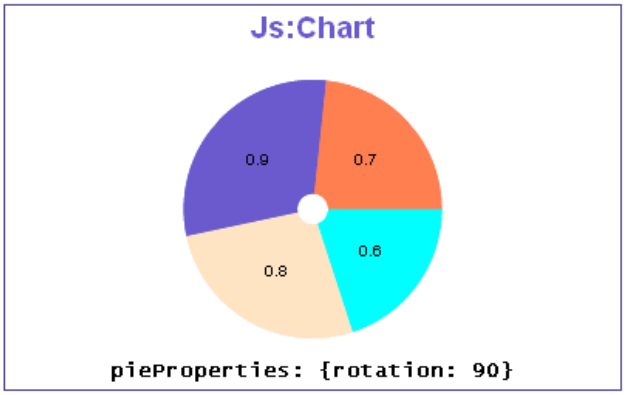
```
pieProperties: {
   rotation: Number
}
```

PARAMETERS:

rotation: 0...359. The default value is zero.

EXAMPLE:





skew

When depth is applied to a pie chart, this property controls the tile of the pie chart.

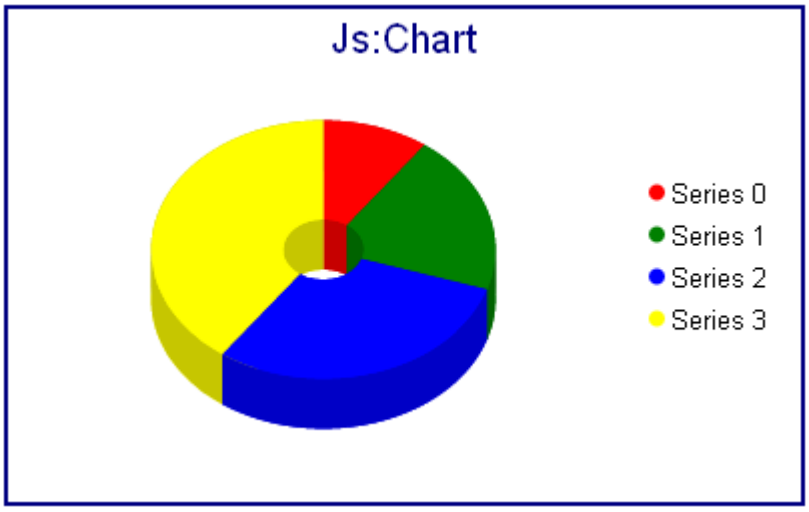
```
pieProperties: {
   skew: Number
}
```

PARAMETERS:

skew: 0...100. The default value is zero.

EXAMPLE:

```
depth: 25,
pieProperties: {holeSize: 20, skew: 25},
```



totalLabel

This property controls the visibility and format of a pie chart's total label (i.e., the total value displayed in the center of the chart).

```
pieProperties: {
   totalLabel: {
         visible: boolean,
         font: 'string',
         color: 'string',
         numberFormat: string | JSON Object | function()
   }
}
```

PARAMETERS:

visible; true/false controls the visibility of the total label. The default value is false.

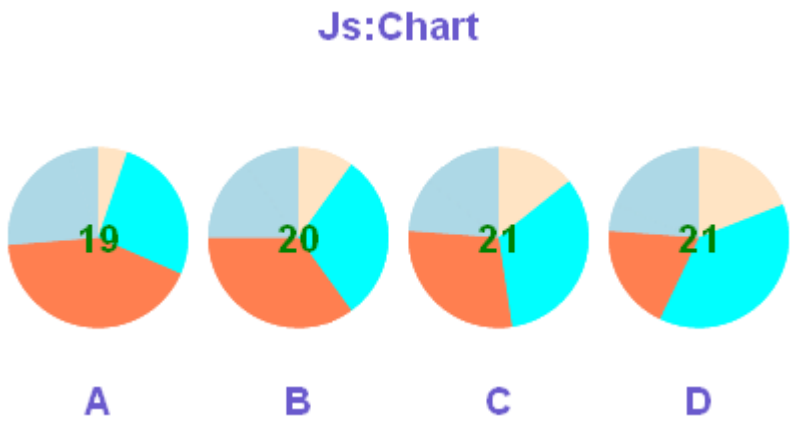
font; a string that defines the size and type face of the total label. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is '10pt Sans-Serif'.

color; a string that defines the color of the total label. See http://www.w3.org/TR/css3-color/#colorunits for the format of this string. The default value is 'black'.

numberFormat; Use a string, JSON object, or function to define the format of the label. See Properties Overview/Formatting Numbers in Section 1 for details and examples. The default value is 'auto'.

EXAMPLE:

```
pieProperties: {
   totalLabel: {
         visible: true,
         font: 'Bold 14pt Sans-Serif',
         color: 'Green'
   }
}
```



Polar Charts (polarProperties)

These properties control the general appearance of polar and radar charts.

```
polarProperties: {
   straightGridLines: boolean,
   extrudeAxisLabels: boolean,
   drawAsArea: boolean
},
```

PARAMETERS:

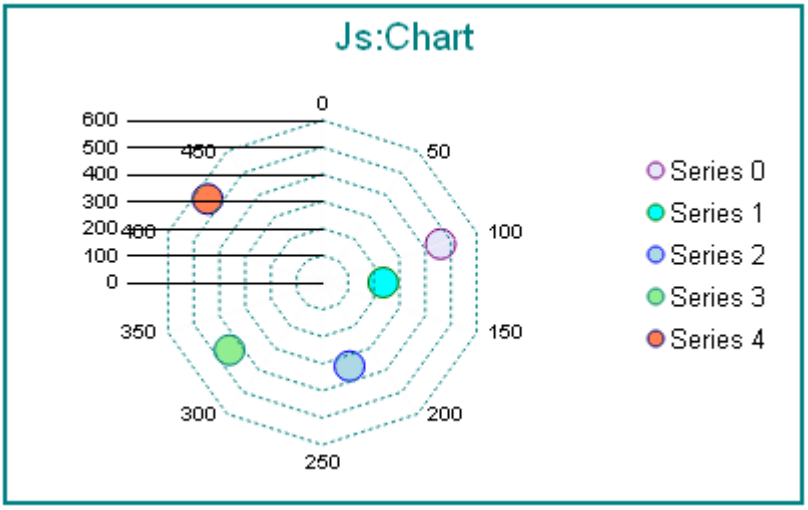
straightGridLines: true = draw straight grid lines with yaxis:majorGrid, false = draw round grid lines with yaxis: majorGrid. The default value is false.

extrudeAxisLabels: true = draw Y-axis labels extruded from the circular grid, false = draw Y-axis labels within the circular grid. The default value is false.

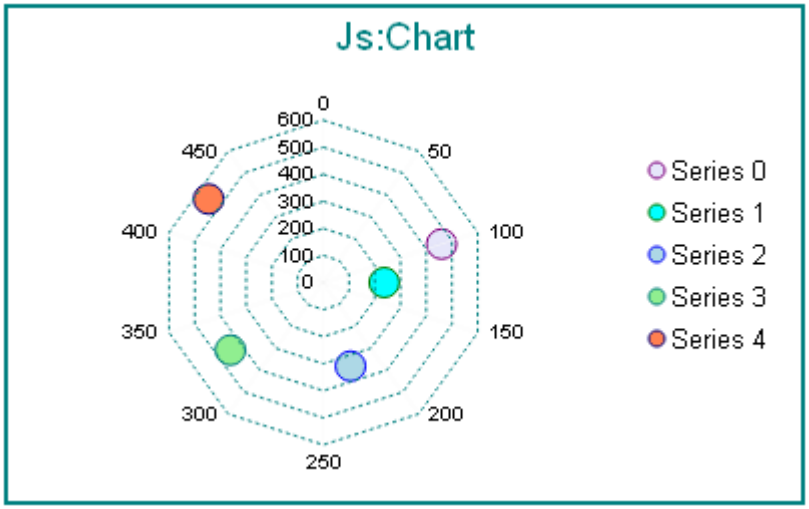
drawAsArea: for radar charts only, true = draw a circular area chart, false = draw a circular line chart. The default value is false.

EXAMPLES:

```
data: [[[100,460]],[[125,225]],[[225,325]],[[325,425]],[[425,525]]],
chartType: 'polar',
legend: {visible: true},
polarProperties: {
   straightGridLines: true,
   extrudeAxisLabels: true
},
yaxis: {
   majorGrid: {
   lineStyle: {width: 1,color: 'teal',dash: '2 2'},
   }
},
series: [
{series: 0, color: 'lavender', marker:{size: 15, shape: 'circle',
border: {width: 1, color: 'purple'}}},
{series: 1, color: 'cyan', marker:{size: 15, shape: 'circle',
border: {width: 1, color: 'green'}}},
{series: 2, color: 'lightblue', marker:{size: 15, shape: 'circle',
border: {width: 1, color: 'blue'}}},
{series: 3, color: 'lightgreen', marker:{size: 15, shape: 'circle',
border: {width: 1, color: 'teal'}}},
{series: 4, color: 'coral', marker:{size: 15, shape: 'circle',
border: {width: 1, color: 'navy'}}},
],
```




```
polarProperties: {
   straightGridLines: true,
   extrudeAxisLabels: false
},
```



NOTES:

- A polar chart is a circular scatter chart.
- Like scatter charts, a polar chart requires two values to draw each marker.
- yaxis properties control the circular grid around polar chart markers. These properties also control the appearance of axis labels and gridlines.
- xaxis properties control the appearance of labels and values on the X-axis (around the outside edge of the circular grid) and X-axis gridlines.

Stock Charts (stockProperties)

A high/low/open/close stock chart requires an array of 4 numbers for each riser: [high, low, open, close]. The first two numbers (high/low) define the high/low line that draws up and down from the box. The second two numbers (open/close) draw the open/close box riser. If either high or low is null, the high-low line is not drawn. If either open or close is null, the open-close box is not drawn. These properties control the general appearance of a stock chart.

```
stockProperties: {
   startTime: 'string' | undefined,
   stopTime: 'string' | undefined,
   interval: 'string' | undefined,
   labelFormat: 'string' | undefined,
   upRiserColor: 'string' | JSON Object,
   downRiserColor: 'string' | JSON Object,
   hiLowLine: {
         width: Number,
         color: 'string' | JSON Object,
         dash: 'string'
   }
},
```

PARAMETERS:

startTime: a string identifying the start time. It can be almost any kind of string denoting time, mixing hours (xx:xx), days of the week, dates, months and years. Examples: "Mon", "1 July 20", "June 2001", "8:00", "6/10/01", "Thu, 4 Apr 1976 14:30", "2 0:00" (February 1), "1 1 0:00" (January 1). The default value is undefined.

stopTime: a string identifying the start time. As with startTime, the string can be almost any date/time format. The default value is undefined.

interval: a string that specifies the time interval: 'seconds', 'minutes', 'hours', 'days', 'weeks', 'months', 'quarters', or 'years'. This property defines the time gap between each group label. If undefined, the library will assign an interval based on the startTime, stopTime, and the number of groups. The default value is undefined.

labelFormat: a string that defines how a time/date number is converted into a label. If undefined, the library will we use a default formats based on the interval. See http://code.google.com/p/datejs/wiki/FormatSpecifiers for available format options. The default value is undefined.

upRiserColor: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient. The default value is '#77b39a'.

downRiserColor: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient. The default value is '#e2675b'.

hiLowLine/width: a number that defines the width of the line in pixels. The default value is 1.

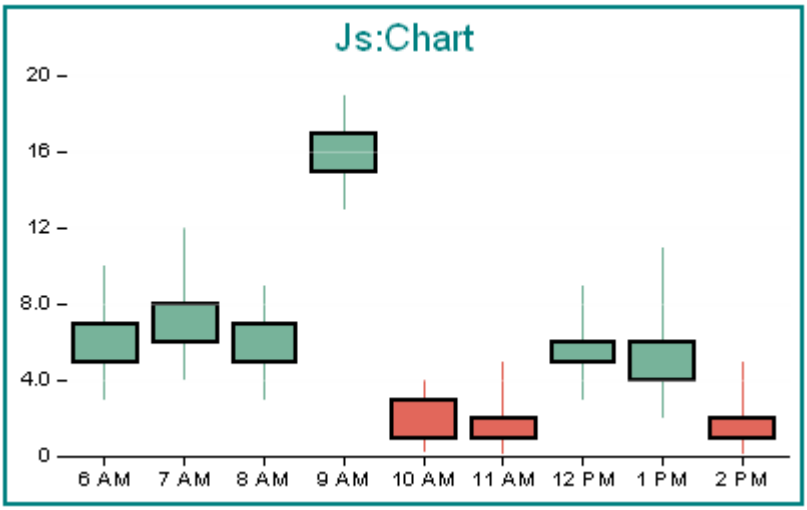
hiLowLine/color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. If a color is not specified, the high-low line for each riser will use the color

assigned to the upRiserColor and downRiserColor. This property colors the high low line for all risers (up and down). The default value is undefined.

hiLowLine/dash: a string that defines the line's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

EXAMPLE:

```
data: [
[[10,3,5,7],[12,4,6,8],[9,3,5,7],[19,13,15,17],[4,.2,3,1],[5,.1,2,1],
[9,3,5,6],[11,2,4,6],[5,.1,2,1]]
],
chartType: 'stock',
blaProperties: {orientation: 'vertical'},
stockProperties: {
   startTime: '6A',
   stopTime: '2P',
   interval: 'hours',
   labelFormat: 'h tt'
   },
```



NOTES:

- A high/low/open/close stock chart requires an array of 4 numbers for each riser: [high, low, open, close].
- The first two numbers (high/low) define the high/low line that draws up and down from the box. The second two numbers (open/close) draw the open/close box riser.
- If either high or low is null, the high-low line is not drawn.
- If either open or close is null, the open-close box is not drawn.

Tagcloud Charts (tagcloudProperties)

These properties control the font and the maximum number of labels to draw in a tagcloud chart.

```
tagcloudProperties: {
   maxNumberOfTags: 50,
   font: "bold 12pt Georgia"
```

PARAMETERS:

maxNumberOfTags: maximum number of labels to show in a tagcloud chart. The default value is 50.

font: a string that defines the type face of labels. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string. The default value is: 'bold 12pt Georgia'.

NOTES:

Although a font specification string may include a point size, this value is not used. The data used to draw the tagcloud chart determines the size of the labels.

Three D Charts (threedProperties)

These properties control the general appearance of 3D charts (i.e., chartType: 'area3D', 'bar3D', and 'surface3D').

```
threedProperties: {
   rotate: Number,
   tilt: Number,
   shadeSides: boolean
},
```

PARAMETERS:

rotate: a number 0...90 defines the rotation of the 3D cube. The default value is 40.

tilt: a number 0...90 defines the tilt of the 3D cube. The default value is 40.

shadeSides: true = shade 3D risers/ false = do not shade 3D risers. The default value is true.

Treemap Charts (treemapProperties)

These properties control the appearance of header cells and cell borders in a treemap chart. This code segment shows the default values from properties.js.

```
treemapProperties: {
   scaleCellFonts: false,
   header: {
         height: undefined,
         fill: 'lightgrey',
         border: {
               width: 0,
               color: 'lightgrey',
               dash: ''
         },
         label: {
               visible: true,
               font: '8pt Sans-Serif',
               color: 'black'
         }
   },
   cellBorder: {
         width: 1,
         color: 'white',
         dash: '',
         outerCellWidth: 3
   }
},
```

header

This property controls the appearance of the header cells in a treemap chart.

```
treemapProperties: {
   header: {
         height: number | string
         fill: string
         border: {
               width: number,
               color: string,
               dash: string
         },
         label: {
               visible: boolean,
               font: string,
               color: string
         }
   },
},
```

PARAMETERS:

header.height: undefined, a number (of pixels), or a percent string (e.g., '5%'). When this property is set to undefined, the header height will be automatically calculated 33% bigger than the header label height. When this property is set to a number, the header height will be the specified number of pixels high. When this property is set to a percent string, the header height will be the specified percentage of the overall height of the treemap. The default value is undefined.

header.fill: undefined, a color definition, or a gradient definition. The default value is 'lightgrey'. See http://www.w3.org/TR/css3-color/#colorunits for the

format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

header.border: These properties define the width, color, and dash style of a border around the header.

header.border.width: a number defines the width of the header border. The default value is 0

header.border.color: a color definition string defines the border color. The default value is 'lightgrey'. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

header.border.dash: a string of numbers defines the border dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

header.label: These properties define the visibility and format of the header label.

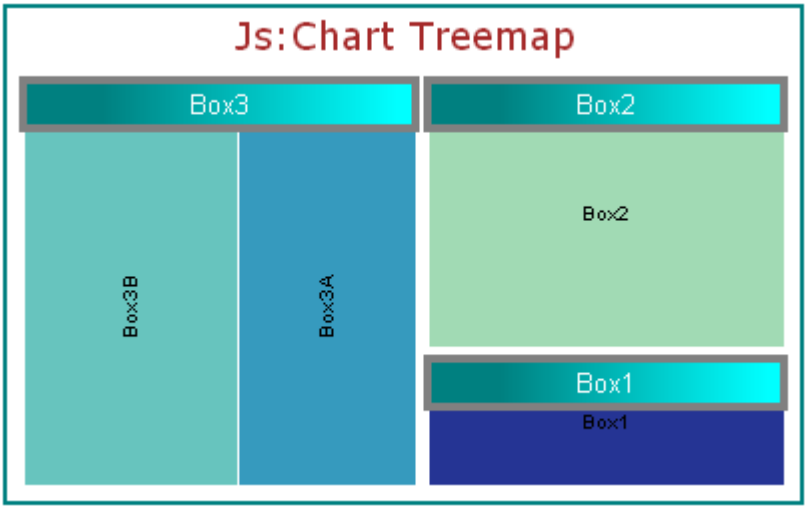
header.label.visible: true/false controls the visibility of the header label. The default value is true.

header.label.font: a font string defines the format of the header label. The default value is '8pt Sans-Serif'. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string.

header.label.color: a color definition string defines the color of the header label. The default value is 'black'. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

EXAMPLE:

```
chartType: 'treemap',
data: {
   Box1: 10,
   Box2: 20,
   Box3: {
         Box3A: 15,
         Box3B: 18
   }
},
treemapProperties: {
header: {height: 25,
   fill: 'linear-gradient(0%,0%,100%,0%, 20% teal, 95% cyan)',
   border: {width: 4,color: 'grey'},
   label: {visible: true,font: '10pt Sans-Serif',color: 'white'}}
   },
```



cellBorder

These properties define the width, color, and dash style of a border around treemap cells.

```
treemapProperties: {
   cellBorder: {
         width: 1,
         color: 'white',
         dash: '',
         outerCellWidth: 3
   }
}
```

PARAMETERS:

cellBorder.width: a number defines the width of the cell border. The default value is 1.

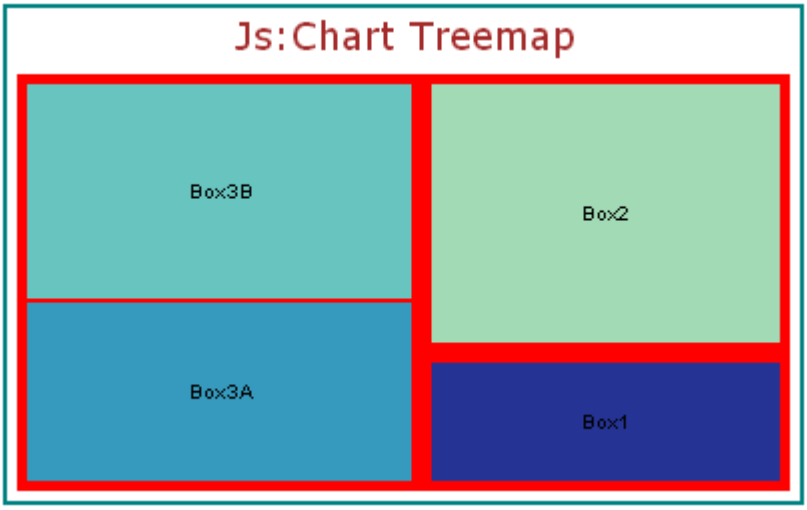
cellBorder.color: a color definition string defines the cell border color. The default value is 'white'. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

cellBorder.dash: a string of numbers defines the cell border dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

cellBorder.outerCellWidth: Defines the width of the border to draw around the top level of cells. The default value is 3. This property controls the width of the borders around only the top level (root) nodes.

EXAMPLE:

```
treemapProperties: {header: {height: 0},
cellBorder: { width: 2,color: 'red', outerCellWidth: 4}
},
```



scaleCellFonts

This property controls the appearance of labels in treemap cells. It does not affect the appearance of labels in header cells.

```
treemapProperties: {
   scaleCellFonts: boolean
}
```

scaleCellFonts: true/false. When true, labels drawn inside each cell will be have their font size scaled according to the cell area. When false, labels are not drawn when the label size exceeds the cell size. The default value is false.

Waterfall Charts (waterfallProperties)

Waterfall charts graphically illustrate the cumulative effect of sequentially introducing positive or negative values to an initial value. The initial and final (or total) values are represented by whole columns drawn from the axis baseline. Intermediate positive and negative values are drawn as floating columns.

The following properties control the general layout of waterfall charts. This code segment shows the default settings from properties.js.

```
waterfallProperties: {
   appendTotalRiser: true,
   positiveRiserColor: '#77b39a',
   negativeRiserColor: '#e2675b',
   zeroRiserColor: '#7593bd',
   otherRiserColor: '#aaaaaa',
   subtotalRisers: [],
   otherRisers: [],
   connectorLine: {
         width: 1,
         color: 'black',
         dash: ''
   }
},
```

appendTotalRiser

This property controls the appearance of a total riser as the last riser in a waterfall chart.

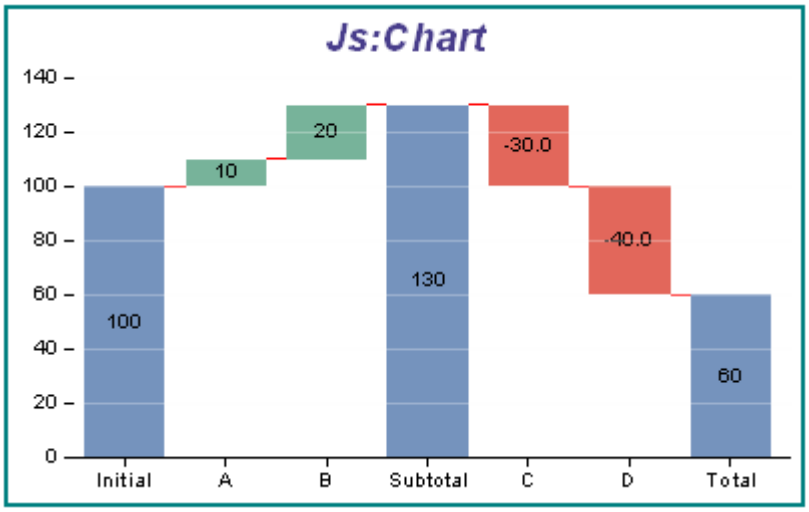
```
waterfallProperties: {
   appendTotalRiser: boolean,
},
```

PARAMETERS:

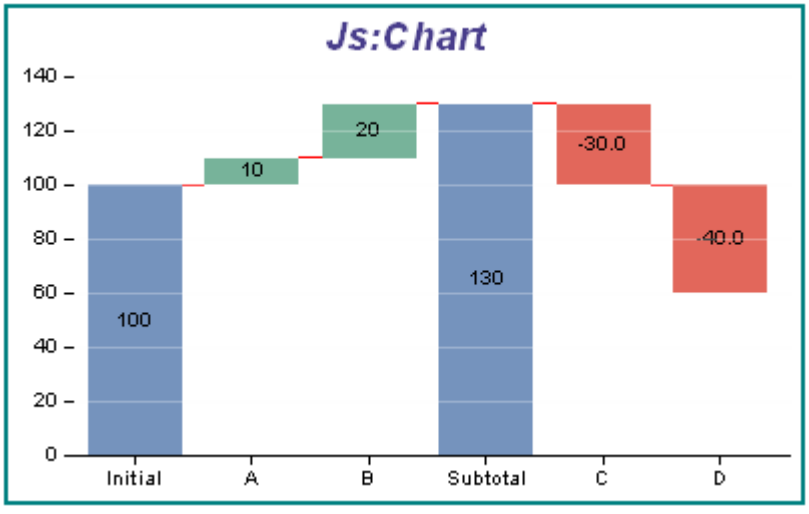
appendTotalRiser: true = draw total riser, false = do not draw total riser. The default value is true.

EXAMPLES:

```
data: [[100,10, 20, 130, -30, -40]],
chartType: 'waterfall',
blaProperties: {orientation: 'vertical'},
groupLabels: ['Initial', 'A', 'B', 'Subtotal', 'C',  'D','Total'],
waterfallProperties: {appendTotalRiser: true}
```




```
waterfallProperties: {appendTotalRiser: false}
```



connectorLine

These properties control the appearance of connector lines between risers in a waterfall chart:

```
waterfallProperties: {
   connectorLine: {
         width: number,
         color: 'string',
         dash: 'string'
   }
},
```

PARAMETERS:

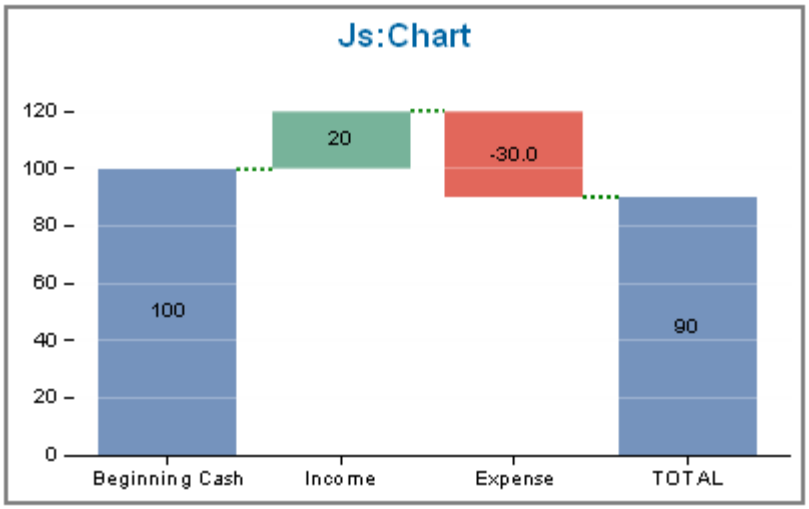
width: The width of the connector line in pixels. The default value is 1.

color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is black.

dash: a string that defines the line's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

EXAMPLE:

```
chartType: 'waterfall',
blaProperties: {orientation: 'vertical'},
waterfallProperties: {
   connectorLine: {width: 2, color: 'green', dash: '2 2'},
},
groupLabels: ["Beginning Cash", "Income", "Expense","TOTAL"]
```



negativeRiserColor

This property controls the color of risers that represent negative values in a waterfall chart.

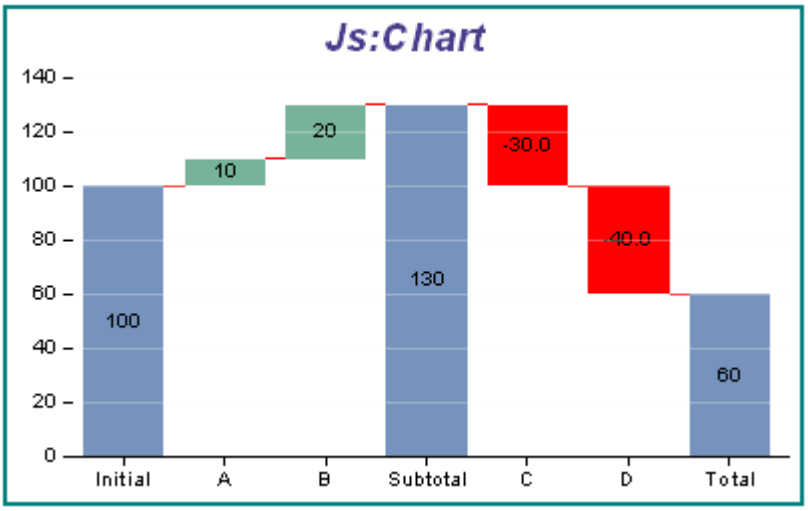
```
waterfallProperties: {
   negativeRiserColor: 'string' | JSON Object
},
```

PARAMETERS:

negativeRiserColor: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient. The default value is #e2675b.

EXAMPLES:

```
data: [[100,10, 20, 130, -30, -40]],
chartType: 'waterfall',
blaProperties: {orientation: 'vertical'},
groupLabels: ['Initial', 'A', 'B', 'Subtotal', 'C',  'D','Total'],
waterfallProperties: {
   negativeRiserColor: 'red'
}
```



NOTES:

You can also use series-specific properties to format these risers individually. Example:

```
series: [
   {series: 0, group: 4, color: 'coral'},
   {series: 0, group: 5, color: 'pink'},
]
```

otherRiserColor

This property controls the color of risers that represent "other" values (if any) in a waterfall chart. Other values are defined in the otherRisers[] array.

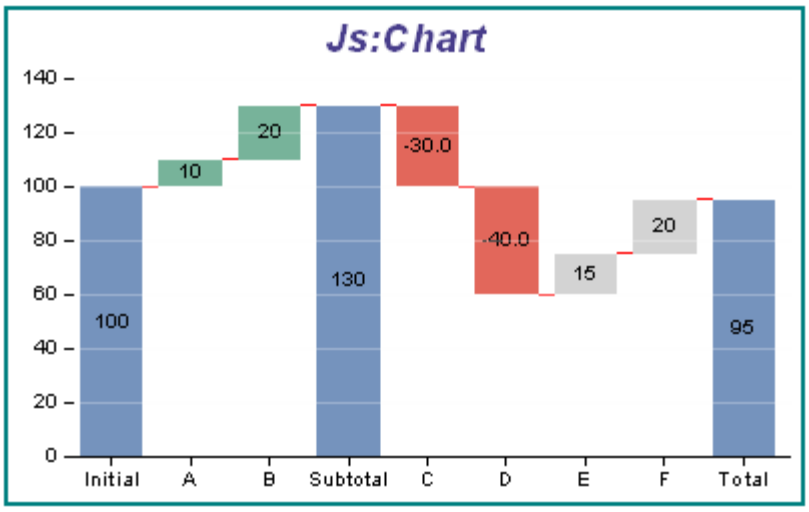
```
waterfallProperties: {
   otherRiserColor: 'string' | JSON object
},
```

PARAMETERS:

otherRiserColor: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient. The default value is '#aaaaaa' (grey).

EXAMPLE:

```
data: [[100,10, 20, 130, -30, -40, 15, 20]],
chartType: 'waterfall',
blaProperties: {orientation: 'vertical'},
groupLabels: ['Initial', 'A', 'B', 'Subtotal', 'C',  'D', 'E',
'F','Total'],
waterfallProperties: {
   otherRiserColor: 'lightgrey',
   otherRisers: ['E', 'F'],
}
```



NOTES:

You can also use series-specific properties to format these risers individually. Example:

```
series: [
   {series: 0, group: 6, color: 'lightgreen'},
   {series: 0, group: 7, color: 'cyan'},
]
```

otherRisers[]

This property defines one or more groups/values to be interpreted as "other" values. Risers representing these values will be assigned the color defined by the otherRiserColor property.

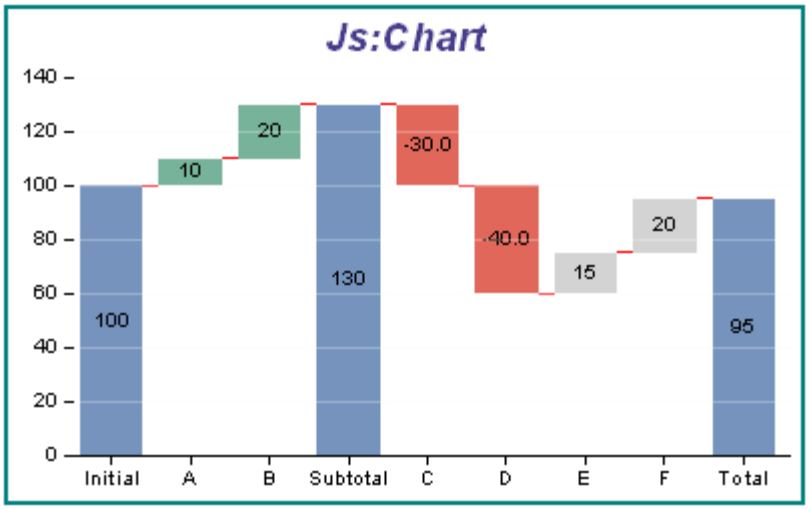
```
waterfallProperties: {
   otherRisers: [number | 'string', ... number | 'string'],
},
```

PARAMETERS:

otherRisers: An array of numbers (matching group indices), or strings (matching group names).

EXAMPLE:

```
data: [[100,10, 20, 130, -30, -40, 15, 20]],
chartType: 'waterfall',
blaProperties: {orientation: 'vertical'},
groupLabels: ['Initial', 'A', 'B', 'Subtotal', 'C',  'D', 'E',
'F','Total'],
waterfallProperties: {
   otherRiserColor: 'lightgrey',
   otherRisers: ['E', 'F'],
}
```



positiveRiserColor

This property controls the color of risers that represent positive values in a waterfall chart.

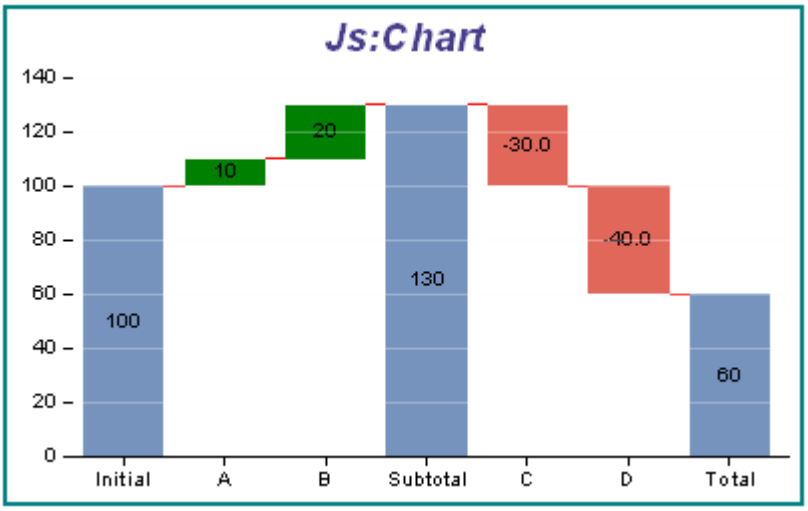
```
waterfallProperties: {
   positiveRiserColor: 'string' | JSON object,
},
```

PARAMETERS:

positiveRiserColor: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient The default value is '#77b39a'.

EXAMPLES:

```
data: [[100,10, 20, 130, -30, -40]],
chartType: 'waterfall',
blaProperties: {orientation: 'vertical'},
groupLabels: ['Initial', 'A', 'B', 'Subtotal', 'C',  'D','Total'],
waterfallProperties: {
   positiveRiserColor: 'green'
}
```



NOTES:

You can also use series-specific properties to format these risers individually. Example:

```
series: [
   {series: 0, group: 1, color: 'limegreen'},
   {series: 0, group: 2, color: 'lightgreen'},
]
```

subtotalRisers[]

This property defines one or more groups/values to be interpreted as "subtotal" values.

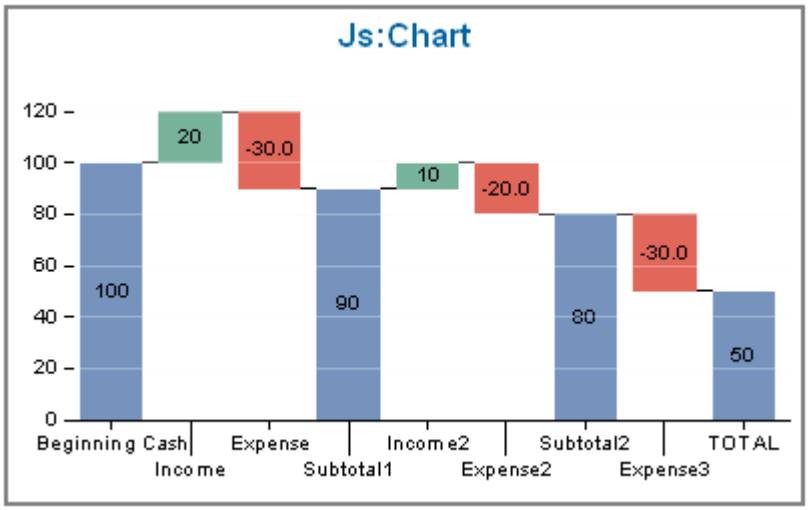
```
waterfallProperties: {
   subtotalRisers: []
},
```

PARAMETERS:

subtotalRisers: An array of numbers (matching group indices), or strings (matching group names).

EXAMPLES:

```
data: [[100,20,-30,90,10,-20,80,-30]],
chartType: 'waterfall',
blaProperties: {orientation: 'vertical'},
waterfallProperties: {
   subtotalRisers: ['Subtotal1','Subtotal2'],
},
groupLabels: ["Beginning Cash", "Income", "Expense",
"Subtotal1","Income2","Expense2","Subtotal2","Expense3","TOTAL"]
```



zeroRiserColor

This property controls the color of risers that represent the initial value, subtotal values (if any) and the total value in a waterfall chart.

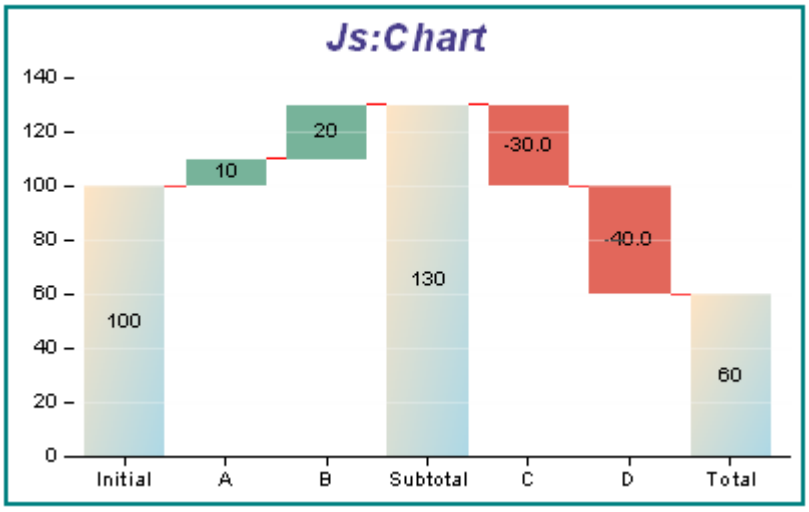
```
waterfallProperties: {
   zeroRiserColor: 'string' | JSON object
},
```

PARAMETERS:

zeroRiserColor: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient. The default value is '#7593bd' (blue).

EXAMPLE:

```
waterfallProperties: {
   zeroRiserColor: {
         type: 'linear',
         start: {x: '0%', y: '0%'},end: {x: '100%', y: '100%'},
         stops: [[0, 'bisque'],[1, 'lightblue']]
   }
}
```



NOTES:

You can also use series-specific properties to format these risers individually. Example:

```
series: [
   {series: 0, group: 0, color: 'lightblue', border: {width: 2,
color: 'blue'}},
   {series: 0, group: 3, color: 'cyan', border: {width: 2, color:
'teal'}},
   {series: 0, group: 6, color: 'turquoise', border: {width: 2,
color: 'green'}},
]
```

Section 5: Special Features

Animation (introAnimation)

This property enables/disables and defines the duration of animation. When enabled, risers/markers are animated when the chart is drawn. Bar/Line/Area risers enter the chart from the base of the frame and become static when they reach their assigned values. Circular risers (e.g., bubble markers) are animated from the inside to the outer edge. Pie slices are drawn clockwise. Animation can be applied to all chart types except 3D charts and charts where 2.5D depth has been applied with the depth property.

```
introAnimation: {
   enabled: boolean,
   duration: number
},
```

PARAMETERS:

enabled: true = enable animation, false = disable animation. The default value is false

duration: a number that defines the duration of the animation. Larger numbers produce slower animation. The default value is 1000.

Color Modes (colorMode)

This property defines the color mode for drawing chart risers and markers. In the default configuration (bySeries), risers/markers in a series are colored according to the values specified by the series.color property. The default values are series: 0, color: red, series: 1, color: green, series: 2, color: orange. This property can be used to apply different color modes to the chart's risers and markers. Some color modes are not applicable or sensible for some chart types. For example, the waterfall chart properties and stock chart properties define colors for positive risers and negative risers.

```
colorMode: {
   mode: 'string',
   colorList: ['string',...'string'],
   data: [numbers | strings] | JSON Object
},
```

PARAMETERS:

mode; a string that defines the chart's color mode: byGroup, byGroupSelection, byHeight, byInterpolation, byInterpolationAlt, byMetric, bySeries (the default), or bySeriesSelection.

- byGroup: risers/markers in a group a collored according to values specified in the series.color property.
- byGroupSelection: risers/markers are colored byGroup and colorMode.data defines segments of the risers/markers to be colored as defined by dataSelection.unselectedColor. Only applicable for a single series of data.
- byHeight: the colors defined in *colorList* create a gradient such that the first color is at the numeric axis minimum, the last color is at the numeric axis maximum, and the remaining colors are interpolated between these. This gradient is then used to color the risers. byHeight is only valid for charts with an ordinal axis and a single numeric axis.

- byInterpolation: the first series is colored the first color in the *colorList* array, the last series is colored the last color, and the series in between are interpolated accordingly.
- byInterpolationAlt: identical to byInterpolation, except it does one additional step of mixing up the order of the chosen series colors. This is useful because now colors between adjacent risers will be more different and easier to distinguish.
- byMetric: colorMode.data determines how colors in colorMode.colorList are applied in single-series bar, line, area, pie, bubble, scatter, and, histogram charts. For heatmap and treemap chart, byMetric mode can be used to apply dataSelection.unselectedColor to segments of color cells.
- bySeries: risers/markers in a series are colored according to the values specified by the series.color property.
- bySeriesSelection: risers/markers are colored bySeries and colorMode.data defines segments of the risers/markers to be colored as defined by dataSelection.unselectedColor. Only applicable for a single series of data.

colorList: For byHeight, byInterpolation, byInterpolationAlt, and byMetric color modes, an array of colors defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. When mode is byMetric, the colorList is not used for heatmap and treemap charts. If only one color is specified for byHeight, byInterporation, or byInterpolationAlt, a lighter version of that color is used as an implicit second color. The default value is undefined.

data: For byGroupSelection, byMetric, and bySeriesSelection, arrays of values or percentage strings or a JSON Object define how colors are applied. The default value is undefined. The data supplied here depends on the chart type, the raw data that defines the chart, and the color mode (byGroupSelection, byMetric, or bySeriesSelection).

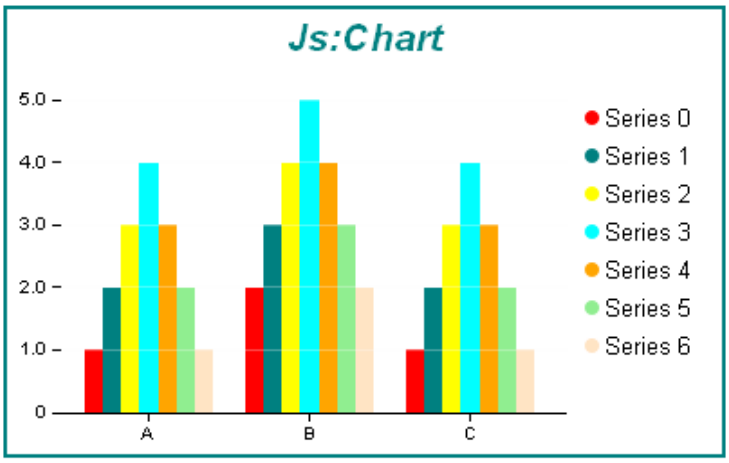
- Area/Bar/Line: For byMetric mode, specify an array of values or percentage strings for each riser that identify how the colors in colorList should be applied to each riser. For byGroupSelection or bySeriesSelection, specify an array of values or percentage strings for each riser that identify segments of each riser that should colored with dataSelection.unselectedColor.
- Bubble: The data set for a bubble chart defines an X-position, Y-position, and marker size (e.g., [[[10,20,30],[40,50,60],[70,80,90]]]). For byMetric mode, data can be set to an array of percentage strings that breaks up the marker size values into pie slices (e.g., [['25%','50%','25%']]). This will draw pie markers on the bubble chart regardless of the series marker shape setting using the colors in colorList for each pie slice. For byGroupSelection or bySeriesSelection mode, set data to an array of values or percentage strings that identify how segments of each bubble should be colored with dataSelection.unselectedColor.
- Pie: For byMetric mode, set data to an array of values to divide each slice into a main slice and one or more sub-slices that are colored according to colorList. Any portion of the main slice that is not included in a sub-slice is colored with dataSelection.unselectedColor. The fill color of the main slice is used as the border color for each sub-slice. Pie charts that use this color mode usually explode all slices to make the chart more readable. For byGroupSelection or bySeriesSelection, set colorMode.data to an array of numbers or an array of percentage strings (one value per slice) to show a slice as partially selected. Set pieProperties.groupPiesBySelection=false to draw the selected portion of each slice as a sub-slice on top of each slice. Set

piePropertiesgroupPiesBySelection=true to reorder the pie with all selected sub-slices together and slightly exploded. Portions of pie slices that are not selected draw with the color specified by dataSelection.unselectedColor.

- Scatter: In a normal scatter chart, each marker is defined by an X-position and Y-position (e.g., [[[10,20],[20,40],[30,70]]]). For byMetric mode, data can be set to an array of percentage strings that breaks up each marker into pie slices (e.g., [['25%','50%','25%']]). This will draw pie markers on the scatter chart regardless of the series marker shape setting using the colors in colorList for each pie slice. For byGroupSelection or bySeriesSelection mode, set data to an array of values or percentage strings that identify how segments of each marker should be colored with dataSelection.unselectedColor.
- Treemap: For byMetric mode, data defines the size of the *colored* area of each cell - the remaining portion of each cell will be colored according to dataSelection.unselectedColor. Each value in the metric data should be either a *smaller* value than its sibling in the raw data, or a percent string ('50%'). The format of colorMode.data depends on the format of the data (array or JSON object) that is used to draw the treemap.

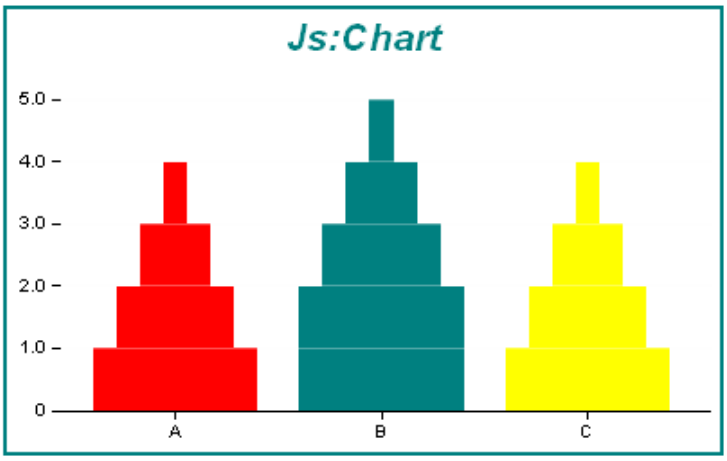
EXAMPLE mode: 'bySeries' (the default):

```
colorMode: {mode: 'bySeries'}, // the default
series: [
   {series: 0, color: 'red'},
   {series: 1, color: 'teal'},
   {series: 2, color: 'yellow'},
   {series: 3, color: 'cyan'},
   {series: 4, color: 'orange'},
   {series: 5, color: 'lightgreen'},
   {series: 6, color: 'bisque'},
]
```



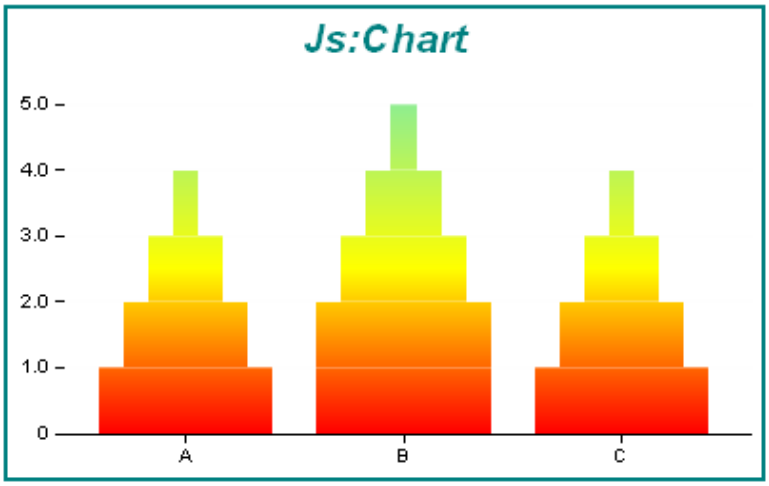
EXAMPLE mode: 'byGroup'

```
colorMode: {mode: 'byGroup'}
```



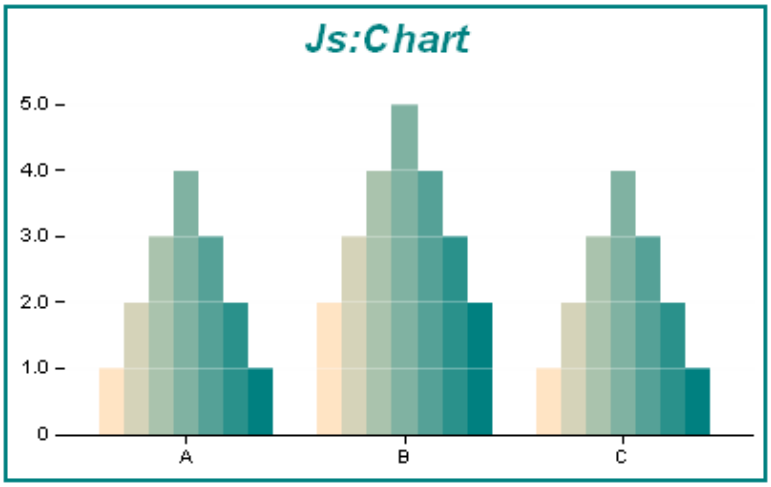
EXAMPLE mode:' byHeight'

```
colorMode: {
   mode: 'byHeight',
   colorList: ['red', 'yellow','lightgreen']
}
```



EXAMPLE mode: 'byInterpolation'

```
colorMode: {
   mode: 'byInterpolation',
   colorList: ['bisque', 'teal']
}
```



Data Selection (dataSelection)

This property enables/disables and defines interactions that allow chart users to select data (risers, markers, pie chart slices, etc.).

SYNTAX:

```
dataSelection: {
         enabled: false,
         selectionMode: ['click', 'ctrlClick', 'dragRect'],
         unselectedColor: 'grey',
         unselectedBorder: {
               width: 0,
               color: 'transparent',
               dash: ''
         },
         selectionRect: {
               fill: 'rgba(120, 120, 120, 0.6)',
               border: {
                     width: 0,
                     color: 'black',
                     dash: ''
               }
         },
         eventCallback: undefined,
         linkedCharts: []
   },
```

PARAMETERS:

enabled: true/false enables/disables data selection.

selectionMode: an array of one or more strings that defines the mouse events the user can use to change the chart selection. Valid options are 'click', 'ctrlClick', 'dragRect'. When 'ctrlClick' is included as a selection mode, shift click or control click will select a data object.

unselectedColor: undefined, a color, or a percentage string in the range '-100%' to '100%' (e.g., '30%') to lighten or darken the riser color. This applies to all risers that are ***not*** selected. The default value is 'grey'.

unselectedBorder: defines the width, color, and dash style of a border around unselected objects.

unselectedBorder/width: a number that defines the width of the border. The default value is zero.

unselectedBorder/color: defines the color of the border. The default value is black.

unselectedBorder/dash: a string that defines the border's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

selectionRect: When *selectionMode* includes the 'dragRect' option, the properties define the rectangle.

selectionRect/fill: a color that defines the fill area of the rectangle. The default value is 'rgba(120, 120, 120, 0.6)'.

selectionRect/border/width: a number that defines the width of the border. The default value is zero.

selectionRect/border/color: defines the color of the rectangle border. The default value is black.

selectionRect/border/dash: a string that defines the border's dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line).

eventCallback: a function to be called when the user selects something on the chart. The function assigned to this property can have two arguments:

```
chart.dataSelection.eventCallback = function(selectionList, obj)
```

selectionList is an array of objects with the following properties:

```
{
   object, // riser
   series, // series #
   group,  // group #
   misc,   // miscellaneous (e.g., bar, riser, wedge, needle, etc.)
   row,    // row # (for Not Yet Implemented matrix charts)
   col     // column # (for Not Yet Implemented matrix charts)
}
```

obj is a single object with {chart, event} properties. Example:

```
chart.dataSelection.eventCallback = function(selectionList, obj) {
   var event = obj.event;
   var chart = obj.chart;
    ... other member handling code ...
}
```

linkedCharts: an array of tdgchart instances that should update their selection list when this chart's selection list changes. Also see: linkDataSelectionCharts()

NOTES:

The *object* in *selectionList* will always be "riser" for dataSelection. For previewSelection, the object identifies the selected chart object (e.g., dataLabels, axis labels, title, subtitle, etc.).

Error Bars (errorBars)

These properties define error bars that can be drawn on bar risers, line risers, area risers, bubble markers, and scatter markers to visualize where risers are above/below expected values.

```
errorBars: {
   yData: undefined,
   xData: undefined,
   hatWidth: '50%',
   line: {
         color: 'black',
         width: 1,
         dash: ''
   },
   marker: {
         color: 'grey',
         shape: 'diamond',
         rotation: 0,
         size: undefined,
         border: {
               width: 1,
               color: 'black',
               dash: ''
         }
   }
},
```

PARAMETERS:

yData: undefined for no error bars on Y-axis data or one or more arrays defining high, marker position (optional), and low values for each riser. Each array can be: [highValue,lowValue] or [highValue, markerPosition, lowValue].

xData: undefined for no error bars on X-axis data or one or more arrays defining high, marker position (optional), and low values for each riser. Each array can be: [highValue,lowValue] or [highValue, markerPosition, lowValue].

hatWidth: a number in pixels or a string that expresses a percentage of the width of the riser.

line/color: a color defined by a keyword or numerical specification string that defines the color of the line and hats. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is black.

line/width: a number that defines the width of the error bar line. The default value is 1.

line/dash: a string that defines the line's dash style. Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line). Use '' for a solid line.

marker/color: a color defined by a keyword or numerical specification string or a gradient defined by a string or JSON object. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient.

marker/shape: a string that defines the shape of the marker: arrow, bar, circle, cross, diamond, fiveStar, hexagon, hourglass, house, pin, pirateCross, plus, rectangle, sixStar, square, thinPlus, tick, or triangle. The default value is 'diamond'. Note that bar, cross, and tick markers require a border width and color.

marker/rotation: a number 0...360 that defines the angle (in degrees) of the marker. The default value is zero.

marker/size: a number that defines the diamenter of the marker in pixels or undefined (to let the library choose the marker size). The default value is undefined.

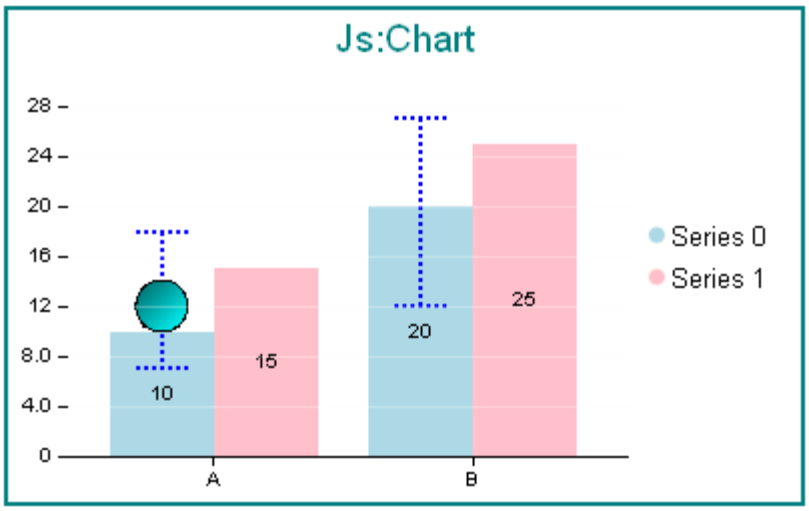
marker/border/width: a number defines the width of the border in pixels.

marker/border/color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is 'black'.

marker/border/dash: a string that defines the border dash style. The default value is '' (a solid line). Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes.

EXAMPLE:

```
blaProperties: {orientation: 'vertical'},
legend: {visible: true},
errorBars: {
hatWidth: '50%',
   yData: [[[7,12,18],[12,27]]],
   line: {
         color: 'blue',
         width: 2,
         dash: '2 2'
   },
   marker: {
         color: 'linear-gradient(0,0,100%,100%, 20% teal, 95% cyan)',
         shape: 'circle'
         }
},
series: [
{series: 0, color: 'lightblue'},
{series: 1, color: 'pink'},
]
```



HTML Tooltips (htmlToolTip)

When tooltips are defined for one or all series, these properties enable/disable and define HTML-based (div) style tooltips for any chart tooltips. The series-specific tooltip property defines the text to show in the tooltip.

```
htmlToolTip: {
   enabled: false,
   mouseMargin: 10,
   style: undefined
   autoTitleFont: 'bold 12pt Sans-Serif',
   autoContentFont: '10pt Sans-Serif',
   snap: false,
   sticky: false
   },
```

PARAMETERS:

enabled: true/false enables/disables HTML-based tool tips. If true, uses HTML-based (div) style tooltip for any chart tooltips. If false, uses standard style tooltips.

mouseMargin: Distance from top of cursor to bottom of tooltip, in pixels.

style: undefined, 'seriesFill', or an object or string defining CSS properties.

autoTitleFont: When series.tooltip is set to 'auto', use a CSS font string to define the formatting of automatic tooltip title text. The default value is 'bold 12pt Sans-Serif'.

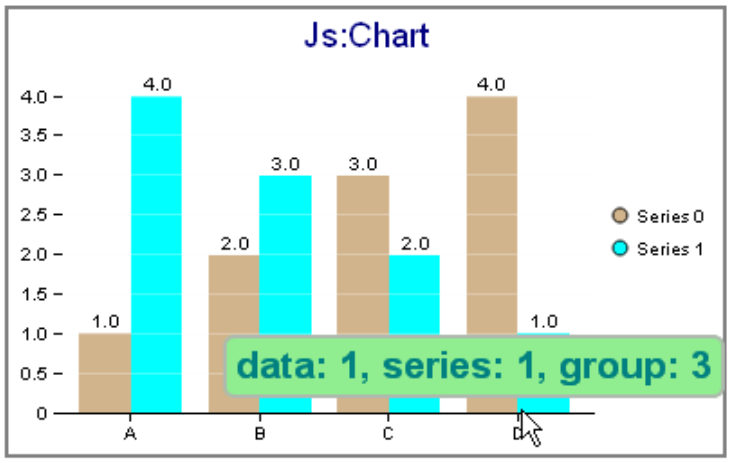
autoContentFont: When series.tooltip is set to 'auto', use a CSS font string to define the formatting of automatic tooltip content text. The default value is '10pt Sans-Serif'.

snap: true/false enables/disables snapping tooltips.

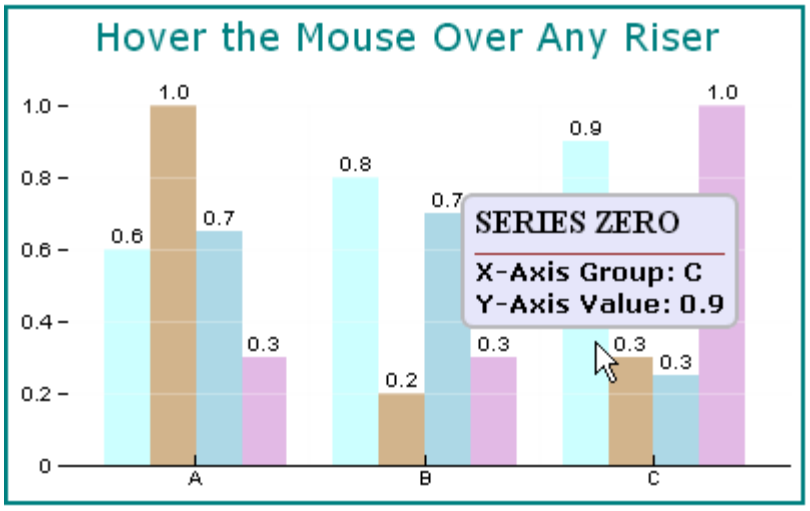
sticky: true/false enables/disables sticky tooltips. If false (the default), the tooltip is hidden when the mouse moves off any riser. If true, the tooltip is only hidden when the mouse moves out of the chart frame.

EXAMPLES:

```
legend: {visible: true},
blaProperties: {orientation: 'vertical'},
htmlToolTip: {enabled: true,
   style: {background: 'lightgreen',font: 'bold 16pt Arial',
         color: 'teal'}
},
series: [
   {series: 'all',tooltip: function(a,b,c) {return 'data: ' + a + ',
series: ' + b + ', group: ' + c;}},
   {series: 0, color: 'tan'},{series: 1, color: 'cyan'}
]
```




```
title: {text: "Hover the Mouse Over Any Riser",font: '14pt
Verdana',color: 'teal'},
border:{width:2, color:'teal'},
htmlToolTip: {enabled: true,
   style: {background: 'lavender'},
   autoTitleFont: 'bold 12pt Times New Roman',
   autoContentFont: 'bold 10pt Verdana'
},
xaxis: {title: {text: 'X-Axis Group'}},
yaxis: {title: {text: 'Y-Axis Value'}},
blaProperties: {orientation: 'vertical'},
series: [
{series: 'all',tooltip: 'auto'},
{series: 0, color: 'rgb(204,255,255)', label: 'SERIES ZERO'},
{series: 1, color: 'tan', label: 'SERIES ONE'},
{series: 2, color: 'lightblue', label: 'SERIES TWO'},
{series: 3, color: 'rgb(226,185,229)', label: 'SERIES THREE'},
]
```



Interaction (interaction)

This property defines user interaction with the chart. For bar, line, area, bubble, and scatter charts, mousedrag: 'pan' will reposition the risers/markers, axis labels, and data labels (if shown) when a user clicks and drags the mouse inside the chart frame. For 3D charts, mousedrag: 'rotate' will rotate the chart frame when a user clicks and drags the mouse around the chart frame. When an other slice is defined with pieProperties.otherSlice, the click: 'otherSliceDrillDown' option will drill-down into the other slice components (i.e., the drilled-down pie chart will show the slices that make up the other slice).

```
interaction: {
   click: 'string'
   mousedrag: 'string'
   dblclick: 'string'
},
```

PARAMETERS:

click: a string that defines interaction on mouse click: undefined or 'otherSliceDrillDown'. The default value is undefined.

mousedrag: a string that defines the interaction on mouse drag: 'select', 'pan', 'riserDrag' (for chartType: bar, blaProperties.seriesLayout='sideBySide' only), 'rotate' (for 3D charts only), or undefined. The default value is undefined.

dblclick: a string that defines the interaction on mouse double click: 'resetView' or undefined. The default value is undefined.

Mouse Over Indicator (mouseOverIndicator)

This property defines a color to highlight the riser when the mouse hovers over a riser. For line, combo, and pareto charts, the marker properties define a marker to draw when the mouse hovers over a line riser.

- For line charts with no markers (series#.marker.visible=false), Js:Chart will add one marker to the line at the point on the line nearest to the cursor. This marker is created with the properties defined by chart.mouseOverIndicator.marker (not the series dependent marker).
- For line charts with markers (series#.marker.visible=true) and scatter charts, Js:Chart will draw the series-defined marker nearest to the cursor as larger (zoomed) with a grey border. The appearance of this border cannot be controlled, and can only be disabled by setting mouseOverIndicator.enabled to false.
- Markers on bubble charts are also drawn with a grey border, but they do not zoom - the marker size is dependent on the data.
- For other chart types (bar, area, pie, tagcloud, treemap, heatmap, etc.), Js:Chart uses the standard highlight defined by mouseOverIndicator.color and no markers.

```
mouseOverIndicator: {
   enabled: boolean,
   color: 'string'  | JSON Object | undefined,
   marker: {
         color: 'string' | JSON Object,
         size: Number,
         shape: 'string',
         rotation: Number,
         position: 'string',
         border: {
               width: Number,
               color: 'string',
               dash: 'string'
         }
   }
}
```

PARAMETERS:

enabled: true/false enables/disables the mouse over indicator. The default value is false (disabled).

color: defines a color to apply when the mouse hovers over a riser. A color can be defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. You may also specify a percentage string in the range '100%' to '100%' to lighten/darken the series color by the specified percentage. The default value is undefined.

marker: for line, combo, and pareto charts, these properties define a marker to draw when the mouse hovers over a line riser.

marker/color: defines the color of the marker. It can be color can be defined by a keyword or numerical specification string or a gradient defined by a string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. See Gradient Definitions for the format of a linear or radial gradient. The default value is 'lightblue'.

marker/size: a number that defines the size of the marker. The default value is 5.

marker/shape: a string that defines the shape of markers for a series: arrow, bar, circle, cross, diamond, fiveStar, hexagon, hourglass, house, pin, pirateCross, plus, rectangle, sixStar, square, thinPlus, tick, or triangle. Note that bar, cross, and tick markers require a border width and color. The default value is 'circle'.

marker/rotation: a number 0...360 that defines the angle (in degrees) of the marker. The default value is zero.

marker/position: a string defines the position of the marker. The default value is 'top'.

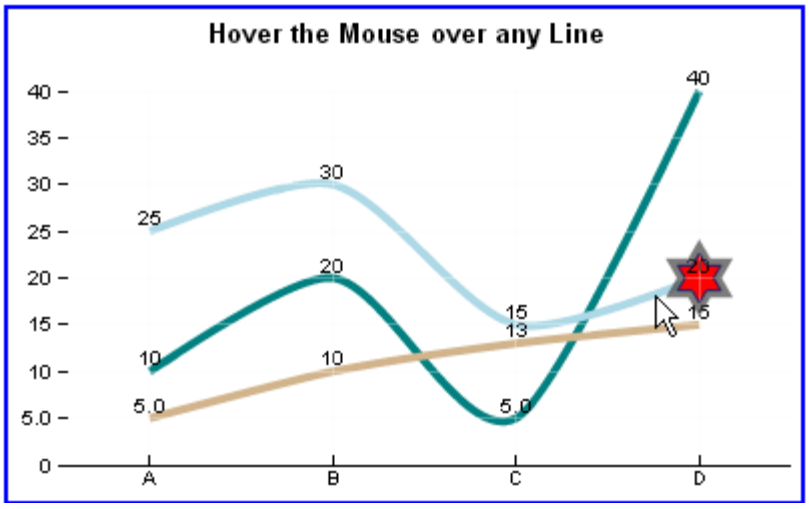
marker/border/width: a number defines the width of the marker border in pixels. The default value is 1.

marker/border/color: a color defined by a keyword or numerical specification string. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string. The default value is 'darkblue'.

marker/border/dash: a string that defines the border dash style. Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes. The default value is '' (a solid line).

EXAMPLE:

```
mouseOverIndicator: {
   enabled: true,
   marker: {
         color: 'red',
         size: 25,
         shape: 'sixStar',
         rotation: 0,
         position: 'top',
         border: {width: 1,color: 'darkblue'}
   }
},
```



Preview Selection (previewSelection)

This property enables/disables and defines the behavior of user-selected chart objects. Related objects can be auto-selected by setting *selectRelatedObjects* to true.

```
previewSelection: {
  enabled: false,
  selectionMode: ['click'],
  selectRelatedObjects: false,
  selectedColor: undefined,
  selectedBorder: undefined,
  eventCallback: undefined
}
```

PARAMETERS:

enabled: true/false enables/disables preview selection

selectionMode: An array of strings. Valid options are 'click', 'rightClick'. Defines the mouse events the user can use to change the chart selection. The default value is 'click'.

selectRelatedObjects: If true, selecting an instance of one chart object will also select all related objects (risers, data labels, axis labels, gridlines, etc).

selectedColor: undefined, a color, or a percentage string in the range '-100%' to '100%' (e.g., '30%') to lighten or darken the riser color. This applies to all *__selected__* risers. The default value is undefined.

selectedBorder: undefined or a style object (i.e., {width: 10, color: 'black', dash: '5 5'}). The default value is undefined.

eventCallback: undefined or function to be called when the user selects an object on the chart. The function assigned to this property can have two arguments:

```
chart.previewSelection.eventCallback = function(selectionList, obj)
```

selectionList is an array of objects with the following properties:

```
{
   object, // title, subtitle, riser, dataLabel, etc.
   series, // series #
   group,  // group #
   misc,   // miscellaneous (e.g., bar, wedge, needle, etc.)
   row,    // row # (for Not Yet Implemented matrix charts)
   col     // column # (for Not Yet Implemented matrix charts)
}
```

obj is a single object with {chart, event} properties. Example:

```
chart.previewSelection.eventCallback = function(selectionList, obj) {
   var event = obj.event;
   var chart = obj.chart;
    ... other member handling code ...
}
```

Reference Lines (referenceLines)

These properties define one or more reference lines to draw on an axis.

```
referenceLines: [
   {
   value: number | 'string',
   axis: 'string',
   line: {
         color: 'string',
         width: number,
         dash: 'string'
   },
   label: {
         text: 'string',
         font: 'string',
         color: 'string'
   },
   anchor: 'string',
   showValue: boolean
   }
]
```

PARAMETERS:

value: a number or string that defines where on the axis to draw the line. For a numeric axis, the number must be a value that is visible on the axis and a string must be a percentage value in the range '0%'...'100%'. For the ordinal axis, a string must be a group label that is visible on the axis and a number must be in the range 0...1 (e.g., .5 will draw the reference line in the center of the chart).

axis: a string that defines the axis on which to draw the line: 'x', 'y', 'y2' or undefined (do not draw line).

line/color: a color defined by a keyword or numerical specification string that defines the color of the line. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

line/width: a number that defines the width of the line.

line/dash: a string that defines the line's dash style. Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line). Use '' for a solid line.

label/text: (optional) a string that defines the label to draw beside a reference line. Use '' or undefined Empty string, or undefined, means draw no label

label/font; a string that defines the size and type face of the label. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string.

label/color: a color defined by a keyword or numerical specification string that defines the color of the label. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

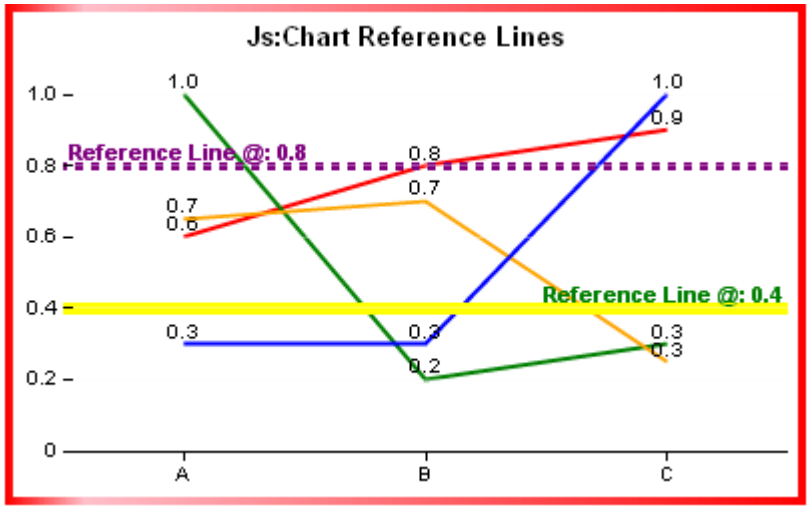
anchor: a string that defines where to draw the reference label relative to the line: 'start' (same side as axis labels) or 'end' (opposite side)

showValue: true/false: true = draw *value* at the specified *anchor* location relative to the line. It will be appended to the label (if defined). false = do not draw label..

EXAMPLES:

```
chartType: 'line',
```

```
blaProperties: {orientation: 'vertical', seriesLayout: 'absolute'},
title: {text: 'Js:Chart Reference Lines'},
referenceLines: [
   {
         value: 0.4,
         axis: 'y',
         line: {
               color: 'yellow',
               width: 6,
               dash: ''
         },
         label: {
               text: 'Reference Line @:',
               font: 'bold 8pt Sans-Serif',
               color: 'green'
         },
         anchor: 'end',
         showValue: true
   },
   {
         value: 0.8,
         axis: 'y',
         line: {
               color: 'purple',
               width: 4,
               dash: '4 4'
         },
         label: {
               text: 'Reference Line @: ',
               font: 'bold 8pt Sans-Serif',
               color: 'purple'
         },
         anchor: 'start',
         showValue: true
   },
]
```



Trend Lines (trendline)

These properties draw a trendline and equation label in bubble and scatter charts.

```
trendline: {
   enabled: false,
   mode: undefined
```

```
   line: {
         width: 1,
         color: 'black',
         dash: ''
   },
   equationLabel: {
         visible: false,
         font: '8pt Sans-Serif',
         color: 'black'
         mode: 'equation'
   }
},
```

PARAMETERS:

enabled: true/false enables/disables the trendline.

mode: a string that defines the trendline mode: exponential', 'geometric', 'hyperbolic', 'linear', 'logarithmic', 'logQuadratic', 'modExponential', 'modHyperbolic', 'polynomial', 'quadratic', 'rational' or undefined. The default value is undefined.

line/width: a number that defines the width of the trendline. The default value is 1.

line/color: a color defined by a keyword or numerical specification string that defines the color of the trendline. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

line/dash: a string that defines the line's dash style. Use a string of numbers that defines the width of a dash followed by the width of the gap between dashes (e.g., dash: '1 1' draws a dotted line). Use '' for a solid line.

equationLabel/visible: true/false controls the visibility of the equation label

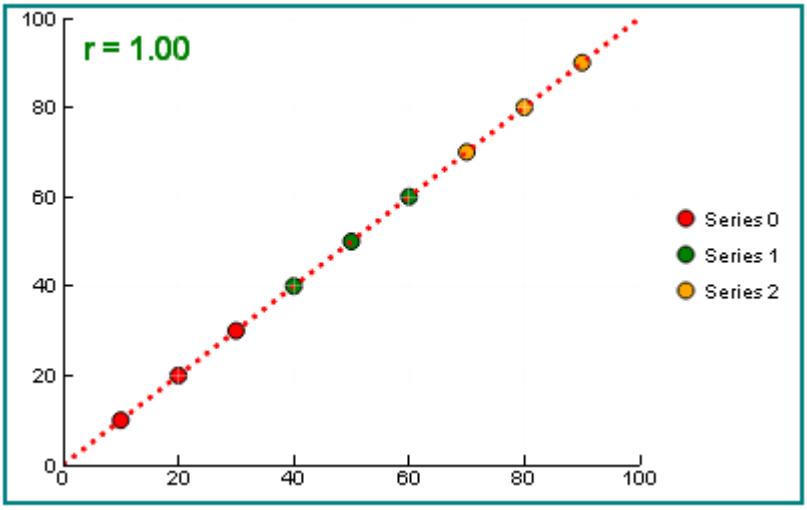
equationLabel/font; a string that defines the size, style, and, typeface of the equation label. The default value is '8pt Sans-Serif'. See http://www.w3.org/TR/CSS2/fonts.html#font-shorthand for the format of this string.

equationLabel/color: a color defined by a keyword or numerical specification string. The default value is 'black'. See http://www.w3.org/TR/css3-color/#colorunits for the format of a color string.

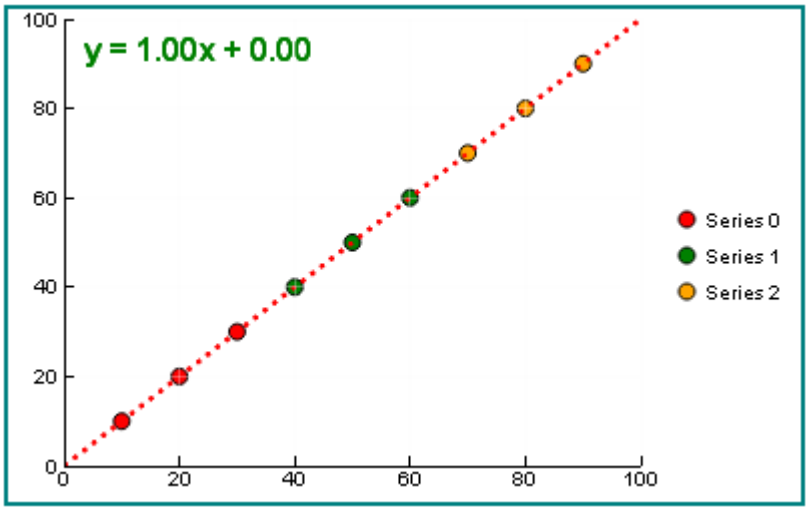
equationLabel/mode: a string that defines the equation label mode: 'equation' (equation in slope intercept form) or 'r' (correlation coefficient). The default value is 'equation'.

EXAMPLES:

```
data: [[[10,10],[20,20],[30,30]],[[40,40],[50,50],[60,60]],[[70,70],
[80,80],[90,90]]],
title: {visible: false,text: "dataLabels: displayMode:
seriesLabel,",font: '8pt Sans-Serif'},
chartType: 'scatter',
legend: {visible: true},
trendline: {enabled: true, mode: 'linear',
   line: {width: 2,color: 'red',dash: '2 4'},
   equationLabel: {visible: true,font: 'bold 12pt Sans-Serif',color:
'green',mode: 'r'}
},
dataLabels: {visible: false},
```




```
trendline: {
   enabled: true,
   mode: 'linear',
   line: {width: 2,color: 'red',dash: '2 4'},
   equationLabel: {visible: true,font: 'bold 12pt Sans-Serif',color:
'green',mode: 'equation'}
```



Appendix A: Default Properties

The following table shows the values from the default properties file (e.g., properties.js) that will be used when properties are not defined. Your Js:Chart package may include multiple default properties files for different chart types.

Property	Default Value
allowBackendFallback	false
axisAutoLayout.rotate45	true
axisAutoLayout.rotate90	true
axisAutoLayout.truncate	true
axisAutoLayout.stagger	true
axisAutoLayout.skip	true
backend	'js'
border.color	'transparent'
border.dash	''
border.width	0
catchErrors	true
chartFrame.border.color	'transparent'
chartFrame.border.dash	''
chartFrame.border.width	0
chartFrame.fill.color	'transparent'
chartFrame.shadow	false
chartsPerRow	undefined
chartType	'bar'
colorMode.colorList	undefined
colorMode.data	undefined
colorMode.mode	'bySeries'
data	[[0.6, 0.8, 0.9], [1.0, 0.2, 0.3], [0.65, 0.7, 0.25], [0.3, 0.3, 1.0]]
dataLabels.color	'black'
dataLabels.displayMode	'x'
dataLabels.font	'7.5pt Sans-Serif'
dataLabels.formatCallback	undefined
dataLabels.numberFormat	'auto'
dataLabels.position	'top'
dataLabels.visible	true
dataSelection.enabled	false
dataSelection.eventCallback	undefined
dataSelection.eventCallback	undefined
dataSelection.linkedCharts	[]
dataSelection.selectionMode	['click', 'ctrlClick', 'dragRect']
dataSelection.selectionRect.border.color	'black'
dataSelection.selectionRect.border.dash	''
dataSelection.selectionRect.border.width	0
dataSelection.selectionRect.fill	'rgba(120, 120, 120, 0.6)'
dataSelection.unselectedBorder.color	'transparent'
dataSelection.unselectedBorder.dash	''
dataSelection.unselectedBorder.width	0

Property	Default Value
dataSelection.unselectedColor	'grey'
dataSubset.startGroup	undefined
dataSubset.stopGroup	undefined
depth	undefined
fill.color	'white'
footnote.align	'center'
footnote.color	'black'
footnote.font	'10pt Sans-Serif'
footnote.text	'Chart Footnote'
footnote.tooltip	undefined
footnote.visible	false
groupLabels	"ABCDEFGHIJKLMNOPQRSTUVWXYZ".split('')
height	250
htmlToolTip.autoContentFont	'10pt Sans-Serif'
htmlToolTip.autoTitleFont	'bold 12pt Sans-Serif'
htmlToolTip.enabled	false
htmlToolTip.mouseMargin	10
htmlToolTip.snap	false
htmlToolTip.sticky	false
htmlToolTip.style	undefined
interaction.click	undefined
interaction.mousedrag	undefined
interaction.dblclick	undefined
introAnimation.duration	1000
introAnimation.enabled	false
labelPadding.frame.bottom	5
labelPadding.frame.left	5
labelPadding.frame.right	5
labelPadding.frame.top	5
labelPadding.label.bottom	5
labelPadding.label.left	5
labelPadding.label.right	5
labelPadding.label.top	5
mouseOverIndicator.color	'yellow'
mouseOverIndicator.enabled	false
mouseOverIndicator.marker.border.width	1
mouseOverIndicator.marker.border.color	'darkblue'
mouseOverIndicator.marker.border.dash	''
mouseOverIndicator.marker.color	'lightblue'
mouseOverIndicator.marker.position	'top'
mouseOverIndicator.marker.rotation	0
mouseOverIndicator.marker.shape	'circle'
mouseOverIndicator.marker.size	5
previewSelection.enabled	false
previewSelection.eventCallback	undefined
previewSelection.selectedBorder	undefined
previewSelection.selectedColor	undefined

Property	Default Value
previewSelection.selectionMode	['click']
previewSelection.selectRelatedObjects	false
riserBevel	undefined
riserCycleEndLightness	0.8
riserDepthGap	0.2
riserShadow	false
subtitle.align	'center'
subtitle.color	'black'
subtitle.font	'10pt Sans-Serif'
subtitle.text	'Chart Subtitle'
subtitle.tooltip	undefined
subtitle.visible	false
swapData	false
swapDataAndLabels	false
title.align	'center'
title.color	'black'
title.font	'10pt Sans-Serif'
title.text	'Chart Title'
title.tooltip	undefined
title.visible	true
width	400
legend.backgroundcolor	'transparent'
legend.labels.color	'black'
legend.labels.font	'7.5pt Sans-Serif'
legend.lineStyle.color	'black'
legend.lineStyle.dash	''
legend.lineStyle.width	0
legend.markerPosition	'left'
legend.markerSize	8
legend.maxEntries	undefined
legend.position	'right'
legend.shadow	false
legend.title.color	'black'
legend.title.font	'10pt Sans-Serif'
legend.title.text	'Legend Title'
legend.title.visible	false
legend.visible	false
legend.xy.x	330
legend.xy.y	80
xaxis.altFrameColor	undefined
xaxis.baseLineStyle.color	'black'
xaxis.baseLineStyle.dash	''
xaxis.baseLineStyle.width	1
xaxis.bIsLog	false
xaxis.bodyLineStyle.color	'transparent'
xaxis.bodyLineStyle.dash	''
xaxis.bodyLineStyle.width	1

Property	Default Value
xaxis.centerGroupLabels	false
xaxis.colorBands	[]
xaxis.intervalMode	undefined
xaxis.intervalValue	undefined
xaxis.invert	false
xaxis.labels.color	'black'
xaxis.labels.font	'7.5pt Sans-Serif'
xaxis.labels.rotation	undefined
xaxis.labels.visible	true
xaxis.majorGrid.aboveRisers	true
xaxis.majorGrid.lineStyle.color	'rgba(255, 255, 255, 0.3)'
xaxis.majorGrid.lineStyle.dash	''
xaxis.majorGrid.lineStyle.width	1
xaxis.majorGrid.visible	false
xaxis.majorGrid.ticks.length	5
xaxis.majorGrid.ticks.lineStyle.color	'black'
xaxis.majorGrid.ticks.lineStyle.width	1
xaxis.majorGrid.ticks.style	'outer'
xaxis.majorGrid.ticks.visible	true
xaxis.max	undefined
xaxis.min	undefined
xaxis.minorGrid.count	undefined
xaxis.minorGrid.lineStyle.color	'black'
xaxis.minorGrid.lineStyle.dash	''
xaxis.minorGrid.lineStyle.width	1
xaxis.minorGrid.ticks.length	5
xaxis.minorGrid.ticks.lineStyle.color	'black'
xaxis.minorGrid.ticks.lineStyle.width	1
xaxis.minorGrid.ticks.style	'inner'
xaxis.minorGrid.ticks.visible	false
xaxis.minorGrid.visible	false
xaxis.mode	undefined
xaxis.numberFormat	'auto'
xaxis.swapChartSide	false
xaxis.title.color	'black'
xaxis.title.font	'7.5pt Sans-Serif'
xaxis.title.text	'X Axis Title'
xaxis.title.visible	false
xaxis.timeAxis.enabled	false
xaxis.timeAxis.interval	undefined
xaxis.timeAxis.labelFormat	undefined
xaxis.timeAxis.startTime	undefined
xaxis.timeAxis.stepSize	undefined
xaxis.timeAxis.stopTime	undefined
yaxis.altFrameColor	undefined
yaxis.baseLineStyle.color	'black'
yaxis.baseLineStyle.dash	''

Property	Default Value
yaxis.baseLineStyle.width	1
yaxis.bIsLog	false
yaxis.bodyLineStyle.color	'transparent'
yaxis.bodyLineStyle.dash	''
yaxis.bodyLineStyle.width	1
yaxis.colorBands	[]
yaxis.colorScale.colors	['#253494', '#2C7FB8', '#41B6C4', '#A1DAB4']
yaxis.intervalMode	undefined
yaxis.intervalValue	undefined
yaxis.invert	false
yaxis.labels.color	'black'
yaxis.labels.font	'7.5pt Sans-Serif'
yaxis.labels.visible	true
yaxis.majorGrid.aboveRisers	true
yaxis.majorGrid.lineStyle.color	'rgba(255, 255, 255, 0.3)'
yaxis.majorGrid.lineStyle.dash	''
yaxis.majorGrid.lineStyle.width	1
yaxis.majorGrid.ticks.length	5
yaxis.majorGrid.ticks.lineStyle.color	'black'
yaxis.majorGrid.ticks.lineStyle.width	1
yaxis.majorGrid.ticks.style	'inner'
yaxis.majorGrid.ticks.visible	true
yaxis.majorGrid.visible	true
yaxis.max	undefined
yaxis.min	undefined
yaxis.minorGrid.count	undefined
yaxis.minorGrid.lineStyle.color	'black'
yaxis.minorGrid.lineStyle.dash	''
yaxis.minorGrid.lineStyle.width	1
yaxis.minorGrid.ticks.length	5
yaxis.minorGrid.ticks.lineStyle.color	'black'
yaxis.minorGrid.ticks.lineStyle.width	1
yaxis.minorGrid.ticks.style	'inner'
yaxis.minorGrid.ticks.visible	false
yaxis.minorGrid.visible	false
yaxis.mode	undefined
yaxis.numberFormat	'auto'
yaxis.swapChartSide	false
yaxis.title.color	'black'
yaxis.title.font	'7.5pt Sans-Serif'
yaxis.title.text	'Y Axis Title'
yaxis.title.visible	false
y2axis.altFrameColor	undefined
y2axis.baseLineStyle.color	'black'
y2axis.baseLineStyle.dash	''
y2axis.baseLineStyle.width	1
y2axis.bodyLineStyle.color	'transparent'

Property	Default Value
y2axis.bodyLineStyle.dash	''
y2axis.bodyLineStyle.width	1
y2axis.intervalMode	undefined
y2axis.intervalValue	undefined
y2axis.invert	false
y2axis.labels.color	'black'
y2axis.labels.font	'7.5pt Sans-Serif'
y2axis.labels.visible	true
y2axis.majorGrid.aboveRisers	true
y2axis.majorGrid.lineStyle.color	'rgba(255, 255, 255, 0.3)'
y2axis.majorGrid.lineStyle.dash	''
y2axis.majorGrid.lineStyle.width	1
y2axis.majorGrid.ticks.length	5
y2axis.majorGrid.ticks.lineStyle.color	'black'
y2axis.majorGrid.ticks.lineStyle.width	1
y2axis.majorGrid.ticks.style	'inner'
y2axis.majorGrid.ticks.visible	true
y2axis.majorGrid.visible	true
y2axis.max	undefined
y2axis.min	undefined
y2axis.minorGrid.count	undefined
y2axis.minorGrid.lineStyle.color	'black'
y2axis.minorGrid.lineStyle.dash	''
y2axis.minorGrid.lineStyle.width	1
y2axis.minorGrid.ticks.length	5
y2axis.minorGrid.ticks.lineStyle.color	'black'
y2axis.minorGrid.ticks.lineStyle.width	1
y2axis.minorGrid.ticks.style	'inner'
y2axis.minorGrid.ticks.visible	false
y2axis.minorGrid.visible	false
y2axis.numberFormat	'auto'
y2axis.title.color	'black'
y2axis.title.font	'7.5pt Sans-Serif'
y2axis.title.text	'Y2 Axis Title'
y2axis.title.visible	false
zaxis.altFrameColor	undefined
zaxis.baseLineStyle.color	'black'
zaxis.baseLineStyle.dash	''
zaxis.baseLineStyle.width	1
zaxis.bIsLog	false
zaxis.bodyLineStyle.color	'transparent'
zaxis.bodyLineStyle.dash	''
zaxis.bodyLineStyle.width	1
zaxis.colorBands	[]
zaxis.intervalMode	undefined
zaxis.intervalValue	undefined
zaxis.invert	false

Property	Default Value
zaxis.labels.color	'black'
zaxis.labels.font	'7.5pt Sans-Serif'
zaxis.labels.visible	true
zaxis.majorGrid.aboveRisers	true
zaxis.majorGrid.lineStyle.color	'rgba(255, 255, 255, 0.3)
zaxis.majorGrid.lineStyle.dash	''
zaxis.majorGrid.lineStyle.width	1
zaxis.majorGrid.ticks.length	5
zaxis.majorGrid.ticks.lineStyle.color	'black'
zaxis.majorGrid.ticks.lineStyle.width	1
zaxis.majorGrid.ticks.style	'outer'
zaxis.majorGrid.ticks.visible	true
zaxis.majorGrid.visible	true
zaxis.max	undefined
zaxis.min	undefined
zaxis.minorGrid.count	undefined
zaxis.minorGrid.lineStyle.color	'black'
zaxis.minorGrid.lineStyle.dash	''
zaxis.minorGrid.lineStyle.width	1
zaxis.minorGrid.ticks.length	5
zaxis.minorGrid.ticks.lineStyle.color	'black'
zaxis.minorGrid.ticks.lineStyle.width	1
zaxis.minorGrid.ticks.style	'inner'
zaxis.minorGrid.ticks.visible	false
zaxis.minorGrid.visible	false
zaxis.mode	undefined
zaxis.swapChartSide	false
zaxis.title.color	'black'
zaxis.title.font	'10pt Sans-Serif'
zaxis.title.text	'Z Axis Title'
zaxis.title.visible	false
blaProperties.barGroupGapWidth	0.2
blaProperties.comboCharts.areaSeriesLayout	undefined
blaProperties.comboCharts.barSeriesLayout	undefined
blaProperties.comboCharts.lineSeriesLayout	undefined
blaProperties.lineConnection	'linear'
blaProperties.lineFillEffect	undefined
blaProperties.orientation	'horizontal'
blaProperties.seriesLayout	'sideBySide'
blaProperties.stackTotalLabel.color	'black'
blaProperties.stackTotalLabel.font	'10pt Sans-Serif'
blaProperties.stackTotalLabel.numberFormat	'auto'
blaProperties.stackTotalLabel.visible	false
boxPlotProperties.connectorLine.color	'black'
boxPlotProperties.connectorLine.dash	''

Property	Default Value
boxPlotProperties.connectorLine.width	1
boxPlotProperties.drawHatAsBox	false
boxPlotProperties.hatWidth	'100%'
boxPlotProperties.medianLine.color	'black'
boxPlotProperties.medianLine.dash	''
boxPlotProperties.medianLine.width	1
bulletProperties.drawFirstValueAsBar	true
funnelProperties.baseWidth	'20%'
funnelProperties.groupLabel.color	'black'
funnelProperties.groupLabel.font	'10pt Sans-Serif'
funnelProperties.groupLabel.visible	false
funnelProperties.riserGap	0
funnelProperties.topWidth	'100%'
ganttProperties.durationValues	false
ganttProperties.interval	undefined
ganttProperties.labelFormat	undefined
ganttProperties.staggerRisers	true
ganttProperties.startTime	undefined
ganttProperties.stopTime	undefined
gaugeProperties.axisTickLength	"30%"
gaugeProperties.axisWidth	"22%"
gaugeProperties.endAngle	45
gaugeProperties.fill.color	'transparent'
gaugeProperties.groupLabel.color	'black'
gaugeProperties.groupLabel.font	'10pt Sans-Serif'
gaugeProperties.groupLabel.visible	false
gaugeProperties.needleBase.border.color	'transparent'
gaugeProperties.needleBase.border.dash	''
gaugeProperties.needleBase.border.width	0
gaugeProperties.needleBase.color	'#1f77b4'
gaugeProperties.needleBase.size	"6%"
gaugeProperties.outerBorder.border.color	'transparent'
gaugeProperties.outerBorder.border.dash	''
gaugeProperties.outerBorder.border.width	0
gaugeProperties.outerBorder.fill.color	'#1f77b4'
gaugeProperties.outerBorder.width	'10%'
gaugeProperties.secondaryNeedlesAsMarkers	false
gaugeProperties.startAngle	135
gaugeProperties.totalLabel.color	'black'
gaugeProperties.totalLabel.font	'10pt Sans-Serif'
gaugeProperties.totalLabel.numberFormat	'auto'

Property	Default Value
gaugeProperties.totalLabel.visible	false
histogramProperties.binCount	undefined
histogramProperties.binSize	undefined
histogramProperties.startBinValue	undefined
paraboxProperties.activeGroup	undefined
pieProperties.explodeClick.distance	25
pieProperties.explodeClick.duration	700
pieProperties.explodeClick.enabled	false
pieProperties.explodeClick.limitExplodeCount	false
pieProperties.feelerLine.color	'black'
pieProperties.feelerLine.dash	''
pieProperties.feelerLine.visible	true
pieProperties.feelerLine.width	1
pieProperties.groupPiesBySelection	false
pieProperties.holeSize	0
pieProperties.label.color	'black'
pieProperties.label.font	'10pt Sans-Serif'
pieProperties.label.visible	false
pieProperties.otherSlice.border.color	'transparent'
pieProperties.otherSlice.border.dash	''
pieProperties.otherSlice.border.width	1
pieProperties.otherSlice.color	'grey'
pieProperties.otherSlice.legendLabel	'Other'
pieProperties.otherSlice.marker.border.color	'transparent'
pieProperties.otherSlice.marker.border.dash	''
pieProperties.otherSlice.marker.border.width	0
pieProperties.otherSlice.marker.shape	'circle'
pieProperties.otherSlice.showDataValues	true
pieProperties.otherSlice.threshold	undefined
pieProperties.rotation	0
pieProperties.skew	0
pieProperties.totalLabel.color	'black'
pieProperties.totalLabel.font	'10pt Sans-Serif'
pieProperties.totalLabel.numberFormat	'auto'
pieProperties.totalLabel.visible	false
polarProperties.drawAsArea	false
polarProperties.extrudeAxisLabels	false
polarProperties.straightGridLines	false
stockProperties.downRiserColor	'#e2675b'
stockProperties.hiLowLine.color	undefined
stockProperties.hiLowLine.dash	''

Property	Default Value
stockProperties.hiLowLine.width	1
stockProperties.interval	undefined
stockProperties.labelFormat	undefined
stockProperties.startTime	undefined
stockProperties.stopTime	undefined
stockProperties.upRiserColor	'#77b39a'
tagcloudProperties.font	"bold 12pt Georgia"
tagcloudProperties.maxNumberOfTags	50
threedProperties.rotate	40
threedProperties.shadeSides	true
threedProperties.tilt	40
treemapProperties.cellBorder.color	'white'
treemapProperties.cellBorder.dash	''
treemapProperties.cellBorder.outerCellWidth	3
treemapProperties.cellBorder.width	1
treemapProperties.header.border.color	'lightgrey'
treemapProperties.header.border.dash	''
treemapProperties.header.border.width	0
treemapProperties.header.fill	'lightgrey'
treemapProperties.header.height	undefined
treemapProperties.header.label.color	'black'
treemapProperties.header.label.font	'8pt Sans-Serif'
treemapProperties.header.label.visible	true
treemapProperties.scaleCellFonts	false
waterfallProperties.appendTotalRiser	true
waterfallProperties.connectorLine.color	'black'
waterfallProperties.connectorLine.dash	''
waterfallProperties.connectorLine.width	1
waterfallProperties.negativeRiserColor	'#e2675b'
waterfallProperties.otherRiserColor	'#aaaaaa'
waterfallProperties.otherRisers	[]
waterfallProperties.positiveRiserColor	'#77b39a'
waterfallProperties.subtotalRisers	[]
waterfallProperties.zeroRiserColor	'#7593bd'