KoolChart for HTML5

USER'S GUIDE

Version 3.0

RiaMore Soft

Contents

1. Overview 5
   1.1. About KoolChart for HTML5 5
   1.2. System Requirements 5
   1.3. KoolChart System Configuration 5
   1.4. KoolChart System Components 6
   1.5. Supported Chart Types 6
2. Creating Charts 8
   2.1. Registering KoolChart License Key 8
   2.2. Creating a Basic Chart 8
   2.3. Viewing KoolChart Versions 13
3. KoolChart Data Formats 13
   3.1. XML - Creating an XML Data 13
   3.2. Array - Creating an Array Data 15
4. KoolChart System Interfacing 17
   4.1. Setting chartVars 17
   4.2. Interface Functions 18
5. Using Layouts 20
   5.1. About Layouts 20
   5.2. Setting <KoolChart> in Layouts 22
   5.3. Setting <Options> in Layouts 22
      5.3.1. Creating <Caption> and <SubCaption> 23
      5.3.2. Creating <Legend> 23
      5.3.3. Using Data Editor 23
   5.4. Setting Styles (CSS) in Layouts 24
6. Using Layouts for Chart Types 29
   6.1. About Common Properties of Charts 29
   6.2. About Axes of Cartesian Charts 30
      6.2.1. CategoryAxis and LinearAxis 32
      6.2.2. DateTimeAxis and LogAxis 33
   6.3. Column 2D Charts 34
   6.4. Column 3D Charts 35
   6.5. Cylinder 3D Charts 36
      6.5.1. Cylinder 3D Column Charts 36
      6.5.2. Cylinder 3D Bar Charts 38
   6.6. Bar 2D Charts 39
   6.7. Bar 3D Charts 40

6.8. Pie and Doughnut Charts 41
6.9. Bubble 3D Charts 45
6.10. Area Charts 46
6.11. Plot Charts 47
6.12. Line Charts 49
6.13. Dashed-Line Charts 51
6.14. Combination Charts 52
6.15. Real-Time Charts 54
6.15.1. Creating Real-Time Charts Based on the Number of Data (Using <CategoryAxis>) 55
6.15.2. Creating Real-Time Charts Based on Time (Using <DateTimeAxis>) 56
6.15.3. Using HttpServiceRepeater in Charts 57
6.16. Radar Charts 58
6.17. Target vs Actual Charts 63
6.18. Scroll Charts 64
6.19. Broken Axis Charts 68
6.20. History Charts 69
6.21. From-To Charts 72
6.22. Matrix Charts 75
6.23. Image Charts 77
6.24. Wing Charts 80
6.25. Real-Time Premium Charts 81
6.26. Candlestick Charts 89
6.27. Gauge Charts 95
6.27.1. Circular Gauge 95
6.27.2. Half-Circular Gauge 98
6.27.3. Cylinder Gauge 98
6.27.4. Linear Gauge 99
6.28. Slide Charts 100
7. Using Layouts for Advanced Users 103
7.1. Displaying Numeric Values in Charts 103
7.2. Setting Colors in Series Items 104
7.3. Applying axis styles 105
7.3.1. About Axis Styles 105
7.3.2. Showing and Hiding Axes Using Properties 107
7.3.3. Changing the Position of Axes 107
7.3.4. Creating Dual Y-Axes 108
7.3.5. Adding Thousands Separators 109
7.3.6. Adding Currency Symbols 110

7.3.7. Using Date Formats in DateTimeAxis ... 112
7.3.8. Adding Titles ... 114
7.4. Designing Chart Background ... 115
7.4.1. Adding Gridlines ... 115
7.4.2. Adding Images ... 117
7.5. Creating Effects in Charts ... 118
7.6. Showing DataTips (ToolTips) on Charts ... 120
7.7. Creating Lines between Stacked DataSets in Column Charts ... 121
7.8. Setting Functions for Click Events on Items ... 122
7.9. Setting User-Defined Functions ... 125
7.9.1. DataTips (ToolTips) Functions ... 126
7.9.2. Axis Labels ... 128
7.9.3. Numeric Values ... 130
7.9.4. Filling Colors ... 132
7.10. Creating Areas and Lines ... 134
7.11. Zooming and Showing Crosshairs ... 137
7.12. Adding Memos ... 141
7.13. Using Vertical Lines in Line Charts ... 143
7.14. Changing Layouts and Data Dynamically ... 143
7.15. Creating Multiple Charts in an HTML Page ... 146
7.16. Real-Time Chart Example - A Stock Monitoring Chart ... 149
7.17. Exporting Charts as Image ... 151
7.18. Sending Chart Image to Server ... 151
7.19. Exporting Charts in Mobile ... 152
8. For Visually Impaired Users ... 153
8.1. Text Substitution ... 153
8.2. Patterns ... 153
9. Using Themes ... 155
9.1. Registering Themes ... 155
9.2. Applying Themes ... 155
9.3. Back to Default Theme ... 155
9.4. Removing Themes ... 155

1. Overview

1.1. About KoolChart for HTML5

KoolChart for HTML5 is a comprehensive charting solution that allows you to add visualization features into your applications. It is a pure JavaScript (HTML5 Canvas) solution and you do not have to install any plugin. KoolChart for HTML5 is platform independent and a non-browser specific World Wide Web solution. As it provides you with more than 30 types of 2D/3D charts and gauges, you can experience a higher level of user interface from KoolChart for HTML5. Please visit our website at http://www.koolchart.com and click Demo, you will see a decent user interface and the highly interactive charting solution.

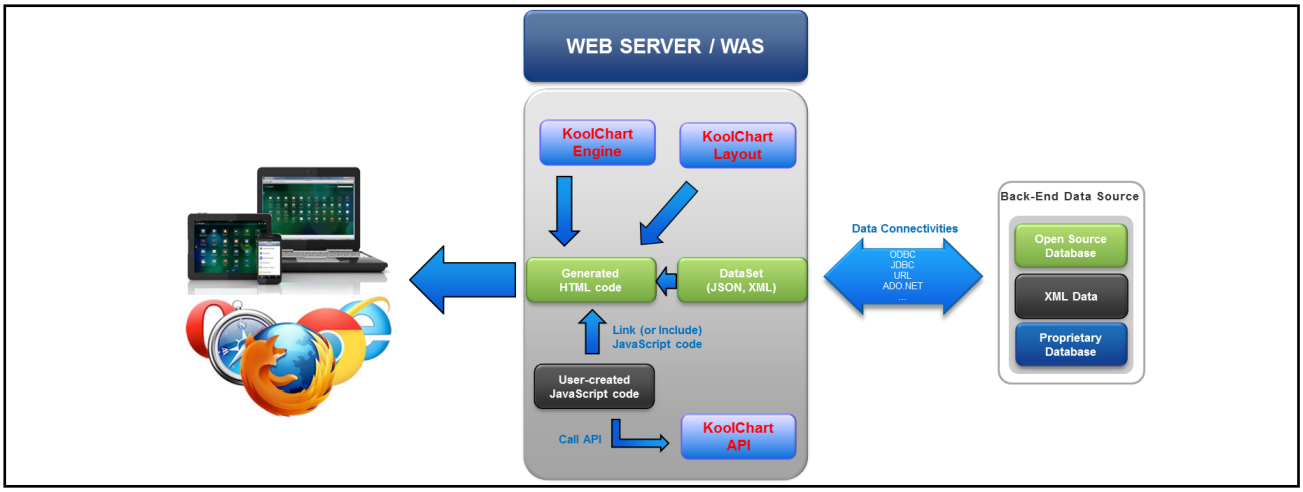
1.2. System Requirements

- Server -- No restrictions.
- Client -- HTML5 Canvas supported browsers.

IE	Firefox	Safari	Chrome	iPhone	Android
9.0+	3.0+	3.0+	3.0+	1.0+	1.0+

For IE7 or IE8, ExplorerCanvas (excanvas.js which is packaged in KoolChart for HTML5) can be used to bring the feature which supports the HTML5 canvas tag to Internet Explorer, but it may degrade the performance of chart rendering. In some cases, the print function may not work properly in IE7 and IE8. Depending on which browser (or version) is used, the feature and the shape of the chart will be different.

1.3. KoolChart System Configuration



<Figure 1 Data Transmission between Server and Client>

Layout and Dataset are two most important elements to create a chart using KoolChart. The format of the layout is XML, and it has detailed information of the chart, which determines the chart type, the animation effect, the axis style, the background color, the label format, etc. Please refer to <5. Using Layouts> for further information.
KoolChart for HTML5 supports two formats for the dataset, which are XML-formatted dataset and Array-formatted dataset. You can choose whichever fits your programming environment. Please refer to <3. KoolChart Data Formats> for further information.

1.4. KoolChart System Components

The installation CD of KoolChart for HTML5 has the following directories.

1. KoolChart
   This directory has minimum files required for running KoolChart. JavaScript files are in JS directory, and image and CSS files are in Assets directory. You can easily set up KoolChart on your web server by copying the KoolChart directory to the server. You need to include the JavaScript file which is under JS directory in your HTML file (or any server side script such as ASP, PHP, etc).

2. LicenseKey
   This directory has the license file (KoolChartLicense.js) for KoolChart.

3. Docs
   Two documents (User Manual and API reference guide) are in the Docs directory. The API reference guide is in the api directory. To view the API reference guide, open index.html file in your web browser.

4. Samples
   This directory has the HTML and XML files for the sample charts.

1.5. Supported Chart Types

The following table shows the JavaScript file names and the corresponding chart types that can be created by using the JavaScript files. You can choose the layout file provided in samples to create the chart you want. The layout files provided in samples have basic settings, so that you will want to change the settings and to add the properties that are needed to create your chart. Please refer to <5. Using Layouts> for further information.

JavaScript File Name	Chart Types
KoolChart.js	Basic Charts – Column, Bar, Pie, Line, Area, etc
KoolRadarChart.js	Radar Chart
KoolHistoryChart.js	History Chart
KoolRealtimeChart.js	Real-Time Chart
KoolScrollChart.js	Scroll Chart
KoolMatrixChart.js	Matrix Chart
KoolImageChart.js	Image Chart
KoolCandleChart.js	Candle Chart
KoolWingChart.js	Wing Chart
KoolBrokenChart.js	Broken Chart
KoolRealtimePremium.js	Real-Time Premium Chart
KoolGaugeChart.js	Gauge Chart
KoolIntegration.js	All Charts

2. Creating Charts

2.1. Registering KoolChart License Key

Once you have obtained the appropriate license key, you must register it before you use KoolChart for HTML5.
The license file is in the following path of the product CD.

- /LicenseKey/KoolChartLicense.js

All you have to do for applying the license key is to include KoolChartLicense.js in the head section of your HTML document (between <head> and </head>).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>

<!--Registering the license key for KoolChart -->
<script src="KoolChartLicense.js" type="text/javascript" language="javascript"></script>
                                                    ...
                                                    ...
                                                    ...
```

< Example 1 Registering the License Keys for KoolChart >

2.2. Creating a Basic Chart

The following example shows how you can create a 3D column chart with a single dataset using the provided samples.
Copy the following directories to your working directory.

- KoolChart JS library: KoolChart/JS/KoolChart.js
- License file: LicenseKey/KoolChartLicense.js
- Sample HTML file: Samples/Column_3D.html

* The license file must be included in your HTML file to render KoolChart.

Column_3D.html is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<link rel="stylesheet" type="text/css" href="./KoolChartSample.css" />

<!--   If you want to support IE 7/8, you'll need the conditional excanvas include. -->
<!--[if IE]><script language="javascript" type="text/javascript" src="../KoolChart/JS/excanvas.js
"></script><![endif]-->

<!-- To include the license key. -->
<script language="javascript" type="text/javascript" src="../LicenseKey/KoolChartLicense.js"></script>

<!-- To include KoolChart JS library -->
<script language="javascript" type="text/javascript" src="../KoolChart/JS/KoolChart.js"></script>


<script type="text/javascript">

// ----------------------- Here we start creating a 3D column chart. -----------------------

//Setting the name of the function which is called when KoolChart is ready to be created. <4.1 Setting
chartVars>
var chartVars = "KoolOnLoadCallFunction=chartReadyHandler";

//Creating a chart.
//Parameters:
//1. Chart ID: You can use any name you like.
//2. DIV ID: The chart will be placed in this DIV.
//3. chartVars: chartVars: Variables used for creating the chart.
//4. Chart width: Default value is 100%
//5. Chart height: Default value is 100%
KoolChart.create("chart1", "chartHolder", chartVars, "100%", "100%");
```

```
// The JavaScript function which is set to the value of KoolOnLoadCallFunction.
// This function will call two functions, setLayout() and setData() which are two main functions of
KoolChart.
//Parameters:
//ID: The chart ID which is used as the first parameter of KoolChart.create().
function chartReadyHandler(id) {
        document.getElementById(id).setLayout(layoutStr);
        document.getElementById(id).setData(chartData);
}

//Setting the layout using the XML-formatted string. <5. Using Layouts>
var layoutStr =
'<KoolChart backgroundColor="0xFFFFFF"   cornerRadius="12" borderStyle="solid">'
+'      <Options>'
+'              <Caption text="Anual Report"/>'
+'      </Options>'
+'      <NumberFormatter id="numFmt" precision="0"/>'
+'      <Column3DChart showDataTips="true">'
+'              <horizontalAxis>'
+'                      <CategoryAxis categoryField="Month" />'
+'              </horizontalAxis>'
+'              <series>'
+'                      <Column3DSeries labelPosition="inside" yField="Profit"
displayName="Profit">'
+'                              <showDataEffect>'
+'                                      <SeriesInterpolate/>'
+'                              </showDataEffect>'
+'                      </Column3DSeries>'
+'              </series>'
+'      </Column3DChart>'
+'</KoolChart>';

// Setting the dataset using JSON format <3.2 Array - Creating an Array Data>
var chartData = [ {"Month":"Jan", "Profit":10000},
                {"Month":"Feb", "Profit":15000},
                {"Month":"Mar", "Profit":12000},
                {"Month":"Apr", "Profit":30200},
                {"Month":"May", "Profit":28000},
                {"Month":"Jun", "Profit":12000},
                {"Month":"Jul", "Profit":22000},
```

```
                    {"Month":"Aug", "Profit":13000},
                    {"Month":"Sep", "Profit":22000},
                    {"Month":"Oct", "Profit":29000},
                    {"Month":"Nov", "Profit":18000},
                    {"Month":"Dec", "Profit":30000} ];

// ---------------------- The end of the configuration for creating a chart. ----------------------
</script>
</head>
<body>

<div class="container">

        <div class="header">
                <p>KoolChart – Column3D Chart</p>
        </div>

        <div class="desc">
                Column3D Chart
                <button onclick="viewSrc()" style="font:11px Arial">View Layout & Data</button>
        </div>
        <div class="content">
                <!-- The DIV in which the chart is placed -->
                <div id="chartHolder" style="width:600px; height:400px;">
                </div>
        </div>
</div>
</body>
</html>
```

<Example 2 Creating a Chart to an HTML Document >

The following list contains more detailed descriptions of the sample HTML document.

1. Set the doctype of your HTML file. (You can use other doctype)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
```

2. Include KoolChart.js and KoolChartLicense.js.

```
<script language="javascript" type="text/javascript"
src="../LicenseKey/KoolChartLicense.js"></script>
<script language="javascript" type="text/javascript" src="../KoolChart/JS/KoolChart.js"></script>
```

3. Include the KoolChart CSS file (KoolChart.css) to create Preloader, Legend, Scroll chart or History chart.

```
<link rel="stylesheet" type="text/css" href="../KoolChart/Assets/KoolChart.css"/>
```

4. Create the parent DIV of the chart in which the chart is created.

```
<div id="chartHolder" style="width:600px; height:400px;"></div>
```

5. Create the layout and the dataset.

6. Call the KoolChart.create() function.

```
KoolChart.create("chart1", "chartHolder", chartVars, "100%", "100%");
```

7. If you want the chart you have created to be shown in IE7 or IE8, include excanvas.js. IE7 and IE8 do not support HTML5, but excanvas.js makes those browsers emulate HTML5. (You should keep in mind that in the performace perspective, IE7 or IE8 with excanvas.js is much worse than HTML5 supported browsers.)

```
<!--[if IE]><script language="javascript" type="text/javascript"
src="../KoolChart/JS/excanvas.js"></script><![endif]-->
```

Please refer to http://excanvas.sourceforge.net for excanvas.js

2.3. Viewing KoolChart Versions

You will want to know the version of KoolChart that you are currently using to request technical support. In order to find the KoolChart version you are currently using, you can check the installation CD or your web page that display a chart created by using KoolChart as follows:

As you need to use the KoolChart.create() function to create the chart, you can also read the value of KoolChart.version in your JavaScript. To find the version information, you can type KoolChart.version in the console or alert dialog.

```
alert (KoolChart.version);
```

3. KoolChart Data Formats

KoolChart for HTML5 supports two types of data format, XML and Array. This chapter describes how you transform your original data into KoolChart-supported data format.

3.1. XML – Creating an XML Data

When you use the XML format for your dataset, be sure the rule that you must begin with <item> and end with </item> for each data. KoolChart displays as many items in the chart as the number of <item>. If no <item> in the XML data, KoolChart displays nothing in the chart.

The single dataset (single series) is used when you have one numeric data, and the multiple datasets (multiple series) are used when you have more than two numeric data.
The following data is for the report of monthly revenue, cost and profit (3 numeric data), which needs to use multiple datasets.

Month	Revenue	Cost	Profit
Jan.	10,000	5,000	5,000
Feb.	15,000	7,000	8,000
Mar.	12,000	6,000	6,000
Apr.	30,200	4,000	26,200
May.	28,000	10,000	18,000
Jun.	12,000	5,000	7,000
Jul.	22,000	10000	12,000
Aug.	13,000	6,000	7,000

Sep	22,000	10,000	12,000
Oct.	19,000	8,000	11,000
Nov.	18,000	7,500	10,500
Dec.	40,000	12,000	28,000

<Table 1 Data to Be Transformed to XML Format>

< XML Data Format>

```
<items>
        <item>
            <Month>Jan</Month>
            <Revenue>10000</ Revenue >
            <Cost>5000</Cost>
            <Profit>5000</Profit>
        </item>
        <item>
            <Month>Feb</Month>
            <Revenue>15000</Revenue>
            <Cost>7000</Cost>
            <Profit>8000</Profit>
        </item>
                    .
                    .
                    .
        <item>
            <Month>Dec</Month>
            <Revenue>30000</Revenue>
            <Cost>12000</Cost>
            <Profit>18000</Profit>
        </item>
</items>
```

You must start with <item> and end with </item> for each data.

<Example 3 XML Data Format>

Once you create XML-formatted data and save the file as the filename, multiData.xml, you need to pass the URL of the filename as a parameter of the setDataURL() function as follows:

document.getElementById("chart1").setDataURL(http://www.koolchart.com/singleData.xml);

Note: Because the setDataURL() function is using the RPC method, the URL does not have to have the name of the XML file. As long as the data returned by the URL is XML-formatted data, you can use any type of server-side script name (e.g. JSP, PHP, etc.,) as an URL.

3.2. Array – Creating an Array Data

Here is an example of using Array-formatted data for the report of monthly profit (1 numeric data).

Month	Profit
Jan.	10,000
Feb.	15,000
Mar.	12,000
Apr.	30,200
May.	28,000
Jun.	12,000
Jul.	22,000
Aug.	13,000
Sep	22,000
Oct.	29,000
Nov.	18,000
Dec.	30,000

```
var chartData = [{"Month":"Jan", "Profit":10000},
        {"Month":"Feb", "Profit":15000},
        {"Month":"Mar", "Profit":12000},
        {"Month":"Apr", "Profit":30200},
        {"Month":"May", "Profit":28000},
        {"Month":"Jun", "Profit":12000},
        {"Month":"Jul", "Profit":22000},
        {"Month":"Aug", "Profit":13000},
        {"Month":"Sep", "Profit":22000},
        {"Month":"Oct", "Profit":29000},
        {"Month":"Nov", "Profit":18000},
        {"Month":"Dec", "Profit":30000} ];
```

<Example 4 Transforming a Single Dataset to Array-Formatted Data>

And the following is an example of Array-formatted data for the report of monthly profit, cost and revenue (multiple datasets).

```
var chartData = [{"Month":"Jan", "Revenue":10000, "Cost":5000, "Profit":5000},
        {"Month":"Feb", "Revenue":15000, "Cost":7000, "Profit":8000},
        {"Month":"Mar", "Revenue":12000 , "Cost":6000, "Profit":6000},
        {"Month":"Apr", "Revenue":30200, "Cost":4000, "Profit":26200},
        {"Month":"May", "Revenue":28000, "Cost":10000, "Profit":18000},
        {"Month":"Jun", "Revenue":12000, "Cost":5000, "Profit":7000},
        {"Month":"Jul", "Revenue":22000, "Cost":10000, "Profit":12000},
        {"Month":"Aug", "Revenue":13000, "Cost":6000, "Profit":7000},
```

```
{"Month":"Sep", "Revenue":22000, "Cost":10000, "Profit":12000},
{"Month":"Oct", "Revenue":29000, "Cost":8000, "Profit":21000},
{"Month":"Nov", "Revenue":18000, "Cost":7500, "Profit":10500},
{"Month":"Dec", "Revenue":30000, "Cost":12000, "Profit":18000}];
```

<Example 5 Transforming a Multiple Datasets to Array-Formatted Data>

You can pass the Array-formatted data above to the chart as follows:

The variable name, chartData, is passed to the setData() function as a parameter.

```
document.getElementById("chart1").setData(chartData);
```

4. KoolChart System Interfacing

4.1. Setting chartVars

chartVars is the indispensable variable of KoolChart and it is used to set layout and dataset. This variable is passed to KoolChart as a parameter from your JavaScript.

The following rules are for the variable, chartVars.

- The type of chartVars is string.
- The separator & is used to assign more than 2 variables to chartVars. The method to assign the dataset and the layout to chartVars is as follows:

```
var layoutURL = "./Column_3D_Layout.xml";
var chartVars = "layoutURL="+layoutURL;

var dataURL = "./singleData.xml";
chartVars += "&dataURL="+dataURL;
```

The following table shows the properties that can be set by using chartVars.

Properties	Values	Descriptions
layoutURL	URL	layout URL
dataURL	URL	dataset URL
KoolOnLoadCallFunction	JavaScript function name	This function is called when KoolChart is ready to be created (executed only one time). Parameter: ID – User-defined ID
displayCompleteCallFunction	JavaScript function name	This function is called when KoolChart is completely rendered. Parameter: ID – User-defined ID
useDataEditor	true, false	Indicates whether or not to use Data Editor
usePattern	true, false	Indicates whether of not to use patterns for visually impaired users

<Table 2 Properties of chartVars>

- The difference between KoolOnLoadCallFunction and displayCompleteCallFunction.

KoolOnLoadCallFunction is used to set the dataset and the layout to create the chart, and it is called when KoolChart is ready to be created.

displayCompleteCallFunction is useful if you want to do additional jobs after the chart has been created, and it is called when KoolChart is completely rendered.

```
var KoolOnLoadCallFunction = "KoolChartOnLoad";
chartVars = "KoolOnLoadCallFunction="+KoolOnLoadCallFunction;

function KoolChartOnLoad(id)
{
    ...
    ...
}
```

<Example 6 Usage Example: chartVars>

Once you set chartVars as above, you must pass chartVars to the KoolChart.create() function as a parameter as follows:

```
KoolChart.create("chart1", "chartHolder", chartVars,   500, 500);
```

4.2. Interface Functions

KoolChart provides you with the following functions. You can use these functions to set the dataset and the layout.

1. setDataURL(value): Sets the URL of XML-formatted data.
2. setLayoutURL (value): Sets the URL of XML-formatted layout.
3. setData (value): Sets the value of Array-formatted data (or XML-formatted data).
4. setLayout (value): Sets the layout string.
5. setSlideDataSet (value): Sets the datasets used in Slide chart.
6. setSlideLayoutSet (value): Sets the layouts used in Slide chart.
7. saveAsImage(): Gets the image snapshot of the chart.
8. getSnapshot(): Gets the image snapshot of the chart which is encoded in base64.
9. resize(): Resize the chart.
10. legendAllCheck(value): Selects all (value=true) or Clears all (value=false) legend items if *useVisibleCheck="true"*.
11. showAdditionalPreloader(): Shows Preloader.
12. removeAdditionalPreloader():Removes Preloader.
13. visibleItemSize(): Defines the number of items displayed in Scroll chart.

14. hasNoData(): Pops up the message box when the chart has no data.
15. changeScrollBarSize(): Changes the size of the scroll bar.
16. showDataEditor(): Shows Data Editor
17. hideDataEditor(): Hides Data Editor

There are 6 combinations of the setLayout (setLayoutURL) function and the setData (setDataURL) function that can generate the chart using KoolChart as follows:

	Layout	Data	Functions
Method 1	XML URL	XML URL	setLayoutURL(URL), setDataURL(URL)
Method 2	XML URL	String	setLayoutURL(URL), setData(Array)
Method 3	XML URL	XML String	setLayoutURL(URL), setData(String)
Method 4	String	XML URL	setLayout(String), setDataURL(URL)
Method 5	String	Array	setLayout(String), setData(Array)
Method 6	String	XML String	setLayout(String), setData(String)

<Table 3 Methods to Set Layout and Dataset>

You can change the layout and the dataset dynamically using the functions above. In order to see the usage example of the functions, you can refer to the samples included in the installation CD.
Open index.html in your web browser, and click "Layout and Data Integration".

5. Using Layouts

5.1. About Layouts

The layout of KoolChart begins with <KoolChart> and ends with </KoolChart>, and it comprises of 3 parts that are <Options>, <Chart> and <Style>. In the <Options> part, you can add <Caption>, <SubCaption> and <Legend>.

The format of the KoolChart layout is XML.

The following example shows how to set the value of the layout properties. <*fill*> is a property of the Column3DSeries object. You can create the SolidColor instance as the value of the <*fill*> property as follows:

```
<KoolChart>
    <Column3DChart showDataTips="true">
                .
                .
                .
        <series>
            <Column3DSeries yField="Profit">
                <fill>
                    <SolidColor color="0xFF0000">
                </fill>
            </Column3DSeries>
        </series>
     </Column3DChart>
</KoolChart>
```

<Example 7 Example: How to Set Property Value>

For user's convenience, KoolChart provides another method to create the instance of value. You can create an ID for the value in advance, and use the ID to set the value of the properties.
The following example shows how to define an ID of <SolidColor>, and set the ID to the values of the two <fill> properties.

```
<KoolChart>
    <SolidColor id="color1" color="0xFF0000"/>
    <Column3DChart showDataTips="true">
                .
                .
                .
```

```
        <series>                                    //The ID should be enclosed in the curly braces
            <Column3DSeries yField="Profit" fill="{color1}"/>
            <Column3DSeries yField="Cost" fill="{color1}"/>
        </series>
    </Column3DChart>
</KoolChart>
```

<Example 8 Example: How to Use ID>

A good usage example of ID is that you can create IDs for two vertical axes, and use the IDs to create the instances of axes for two different series.

The two terminologies, Chart and Series, will be used frequently in this document. The following table describes key concepts of Chart and Series.

	Descriptions	Remarks
Chart	It represents the visual part of the chart. It defines the appearances of the chart such as axes, background, size, etc. The most important role of Chart is to specifiy the position (coordinates) of series.	ColumnChart, Column3DChart, PieChart, Pie3DChart, BarChart, etc
Series	It represents the actual data. If you have three numeric data for a chart, you need to define three series for each data.	ColumnSeries, Column3DSeries, BarSeries, Pie3DSereis, etc

<Table 4 Descriptions of Chart and Series >

5.2. Setting 〈KoolChart〉 in Layouts

<KoolChart>/</KoolChart> indicate the start/end of the layout. The main role of <KoolChart> is to specify the overall design of the chart by setting the properties such as cornerRadius, borderStyle, backgroundColor, etc.

The following example shows how to use the properties of <KoolChart>.

```
<KoolChart cornerRadius="12" borderStyle="solid" backgroundColor="0xFFFF77">
        <Options>
            <Caption text="Anual Report" />

                ……

</KoolChart>
```

<Example 9 Example: Properties of <KoolChart>>

5.3. Setting 〈Options〉 in Layouts

<Options> is optional. You can add <Caption>, <SubCaption>, <Legend> and <DataEditor> to between <Options> and </Options>.

```
<KoolChart>
        <Options>
                <Caption text="Annual Report"/>   // Title
                <SubCaption text="2008"/>   //Subtitle
                <Legend/>   //Legend
        </Options>
        <Column3DChart showDataTips="true" width="100%" height="100%" >
            <series>
                <Column3DSeries yField="Profit" displayName="Profit">
                .
               .
               .
        </Column3DChart>
</KoolChart>
```

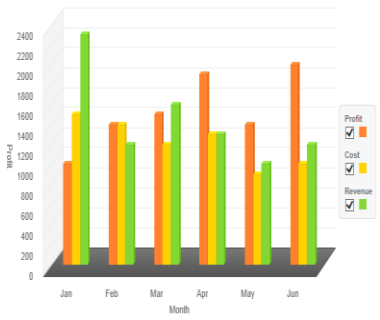
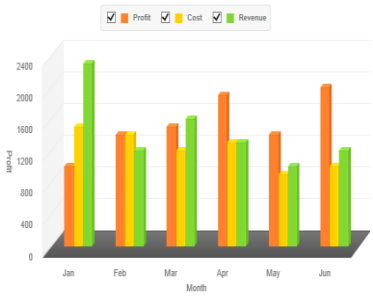
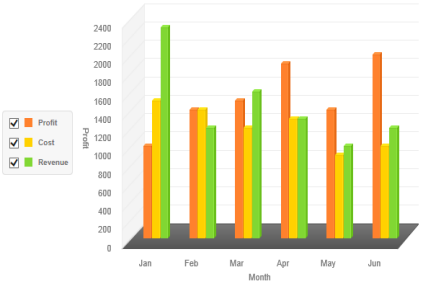
<Example 10 Usage Example: Options>

5.3.1. Creating <Caption> and <SubCaption>

<Example 10 Usage Example: Options> shows the layout for a Column3D chart. You can make the title or the subtitle of the chart by setting the value of the *text* property in <Caption> or <SubCaption>.

5.3.2. Creating <Legend>

You will want to make the legend in the chart with multiple series. In order to make the legend in the chart, you need to add <Legend> to between <Options> and </Options>, and to set the value of the *displayName* property of <series> (e.g. if you want to create a Column3D chart, you need to define <Column3Dseries> between <series> and </series>). The value of the *displayName* property will be displayed in the legend.

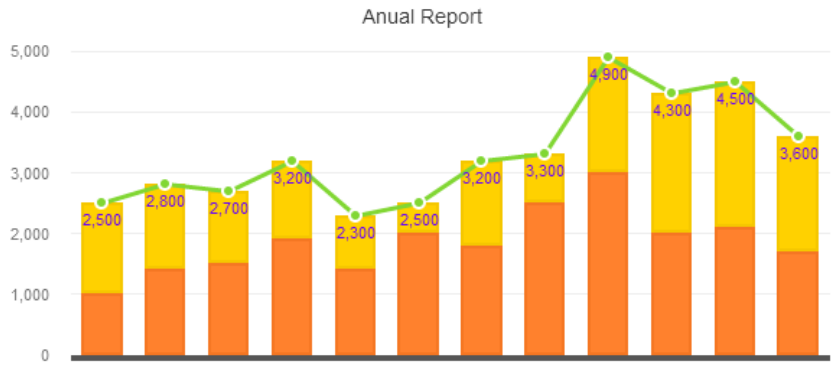
OUTPUTS			
PROPERTIES	position="right" direction="vertical" useVisibleCheck="true" vAlign="middle"	position="top" direction="horizontal" useVisibleCheck="true" hAlign="right"	position="left" direction="vertical" useVisibleCheck="false" vAlign="bottom"

<Example 11 Example: Properties of Legend>

5.3.3. Using Data Editor

Data Editor is displayed at the below of the chart, and you can double-click the cell to modify the data of the chart. To use Data Editor in the chart, you need to set chartVars in the chart, and to add

<DataEditor> to between <Options> and </Options>. For detailed information of the properties of <DataEditor>, please refer to the API reference. (Open Docs/api/inedxe.html in your browser and go to the section of DataEditor.



	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Revenue	2,500	2,800	2,700	3,200	2,300	2,500	3,200	3,300	4,900	4,300	4,500	3,600
Cost	1,500	1,400	1,200	1,300	900	500	1,400	800	1,900	2,300	2,400	1,900
Profit	1,000	1,400	1,500	1,900	1,400	2,000	1,800	2,500	3,000	2,000	2,100	1,700

<Figure 2 Using DataEditor >

5.4. Setting Styles (CSS) in Layouts

<Style> is used to define the styles of the chart, and you need to follow the rules below when you create <Style> in the chart.

1. You must define <Style> as a first-level descendant of <KoolChart>.
2. The style name begins with a dot (.) followed by the character string that begins with a lowercase first character.
3. The properties of the style must be enclosed in the curly braces ({}).
4. The colon (:) must be placed between the property name and the property value as a seperator and the property definition ends with the semicolon (;).

The following table shows the correct and incorrect examples.

Correct example	Incorrect example
<KoolChart> <Options>	<KoolChart> <Options>

...
 	
<Style>
.KoolChartStyle
{
backgroundColor:0xFFFFFE;
borderColor:0x77EE9E;
cornerRadius:12;
borderThickness:3;
borderStyle:solid;
}
<Style>
</KoolChart>
	...
	
	<Style>
	ChartStyle
	[
	backgroundColor:0xFFFFFE;
	borderColor:"0x77EE9E";
	cornerRadius,12;
	borderThickness:3;
	borderStyle:'solid';
	]
	<Style>
	</KoolChart>

<Table 5 Example: Correct and Incorrect representations of <Style>>

Once you create the style (KoolChartStyle) as above, you can apply the style as follows:

```
<KoolChart styleName="KoolChartStyle">
        <Options>
                <Caption text="Annual Report"/>
        </Options>

         ……

        <Style>
        .KoolChartStyle
        {
        backgroundColor:0xFFFFFE;
        borderColor:0x77EE9E;
        cornerRadius:12;
        borderThickness:3;
        borderStyle:solid;
        }
        <Style>

</KoolChart>
```

<Example 12 Example: How to Apply Styles 1>

In the above example, KoolChartStyle is defined as a first-level descendant of <KoolChart>. To apply the

style to an element in the chart, you need to set the value of the *styleName* property of the element as follows:

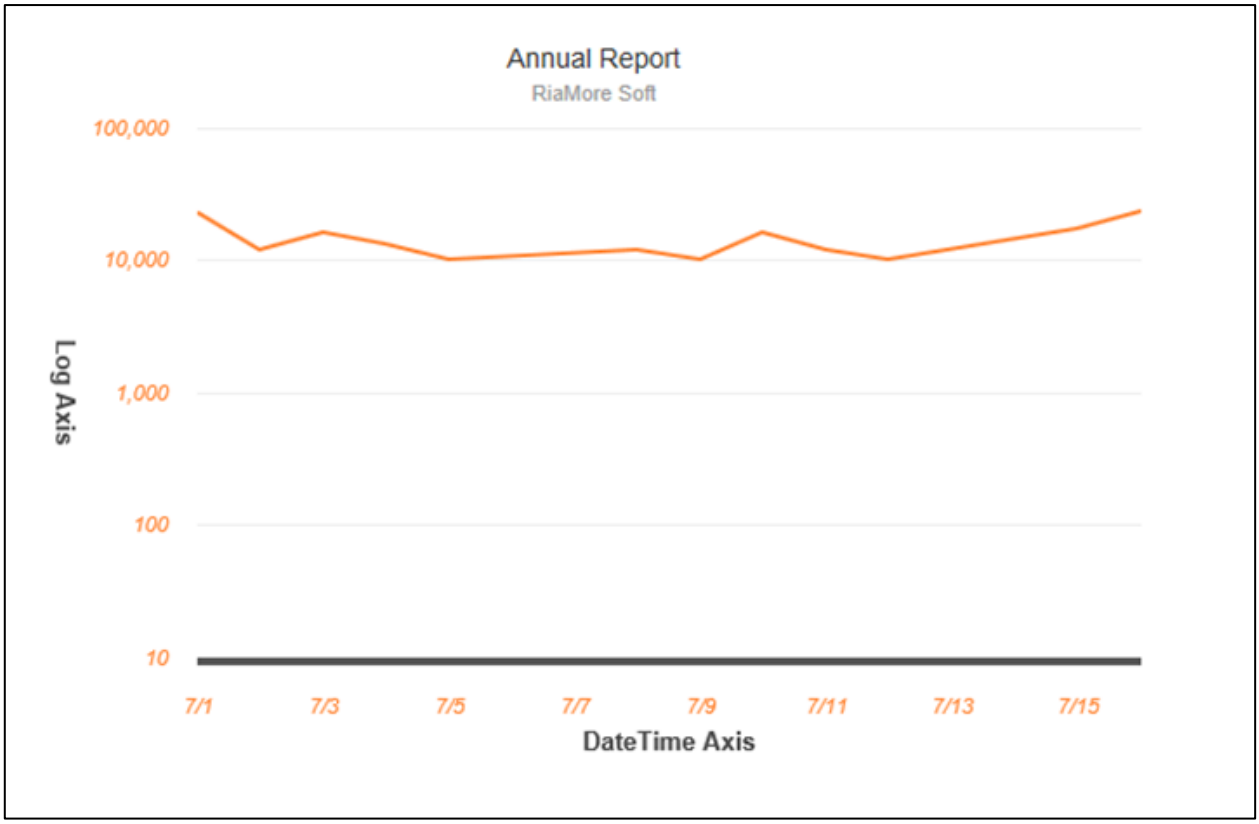
styleName="the style name you defined". e.g. styleName="KoolChartStyle"

The usage example of the style is as follows:

```
<KoolChart styleName="KoolChartStyle">
        <Options>
                <Caption text="Annual Report" styleName="captionStyle"/>
                <SubCaption text="RiaMore Soft" styleName="subCaptionStyle"/>
        </Options>
        <DateFormatter id="dateFmt" formatString="M/D"/>
        <NumberFormatter id="numFmt"/>
        <Line2DChart showDataTips="true" styleName="chartStyle">
                <horizontalAxis>
                        <CategoryAxis id="hAxis" categoryField="Month" title="Horizontal Axis"/>
                </horizontalAxis>
                <horizontalAxisRenderers>
                        <Axis2DRenderer axis="{hAxis}"  axisTitleStyleName="chartAxisStyle"/>
                </horizontalAxisRenderers>
                <series>
                        <Line2DSeries yField="Revenue" displayName="Revenue">
                                <showDataEffect>
                                        <SeriesInterpolate/>
                                </showDataEffect>
                        </Line2DSeries>
                </series>
        </Line2DChart>
        <Style>
                .KoolChartStyle    {
                        backgroundColor:#FFFFFE;
                        borderColor:#77EE9E;
                        cornerRadius:12;
                        borderThickness:3;
                        borderStyle:solid;
                }
                .captionStyle {
                        fontSize:12;
                        fontFamily:Tahoma;
```

```
                fontWeight:bold;
                color:#777777;
        }
        .subCaptionStyle {
                fontSize:11;
                fontStyle:italic;
                color:#777777;
        }
        .chartStyle {
                fontSize:11;
                fontStyle:italic;
                color:#0000FF;
        }
        .chartAxisStyle {
                color : #4691E1;
                fontSize : 14;
                fontWeight : bold;
                fontStyle : italic;
        }
    </Style>
</KoolChart>
```

<Example 13 Example: How to Apply Styles 2>



<Figure 3 Example: Output of the Chart with Styles>

6. Using Layouts for Chart Types

6.1. About Common Properties of Charts

Charts can be divided into two general categories, which are Cartesian charts that have axis and Polar charts that have no axis. The following table describes the properties of Cartesian charts.

Properties	Values	Descriptions
horizontalAxis	CategoryAxis, LinearAxis, DateTimeAxis, LogAxis	Specifies the horizontal axis (X-axis).
verticalAxis	CategoryAxis, LinearAxis, DateTimeAxis, LogAxis	Specifies the vertical axis (Y-axis).
horizontalAxisRenderer	Axis3DRenderer, AxisRenderer	Specifies the renderer of the horizontal axis.
verticalAxisRenderer	Axis3DRenderer, AxisRenderer	Specifies the renderer of the vertical axis.
series	Chart series (e.g. Column3DSeries, Line2DSeries, etc)	Specifies the series of the chart.
annotationElements	GridLines, Image, CanvasElement, CrossRangeZoomer, etc	Specifies the front elements of the chart (top of the stacking order in Z-index).
backgroundElements	GridLines, Image, CanvasElement, etc	Specifies the background elements of the chart.
showDataTips	true | false	Whether or not to display tooltips when mouseover event occurs.
paddingTop	Number	Specifies the top margin.
paddingBottom	Number	Specifies the bottom margin.
paddingLeft	Number	Specifies the left margin.
paddingRight	Number	Specifies the right margin.
itemClickJsFunction	JavaScript function	Specifies the name of the function called when the item is clicked.
dataTipJsFunction	JavaScript function	Specifies the name of the function called to display the user-defined tooltips.
gutterLeft	Number	Specifies the left margin of the axis.
gutterRight	Number	Specifies the rigth margin of the

		axis.
gutterTop	Number	Specifies the top margin of the axis.
gutterBottom	Number	Specifies the bottom margin of the axis.

<Table 6 Properties: Cartesian Charts>

The following table describes the properties of Polar charts.

Properties	Values	Descriptions
innerRadius	0.0 ~ 1.0 (default: 0.0)	Specifies the size of the empty space of the Doughnut chart. As the value reaches 1.0, the size of the empty space increases. (0.0 = Pie chart).
showDataTips	true | false (default: false)	Whether or not to display tooltips when mouseover event occurs.
explodable	true | false (default: true)	Whether or not to explode out a clicked piece of the doughnut (or pie).
series	PieGradationSeries, Pie3DSeries	Specifies the series of the chart.
itemClickJsFunction	JavaScript function	Specifies the name of the function called when the item is clicked.
dataTipJsFunction	JavaScript function	Specifies the name of the function called to display the user-defined tooltips.

<Table 7 Properties: Polar Charts>

6.2. About Axes of Cartesian Charts

Except Pie charts and Doughnut charts, all charts (Cartesian charts) have the X-axis (horizontal axis) and the Y-axis (vertical axis).

In the chart, there are two types of axis. The one is the category axis and the other is the numeric axis. There is only one axis for the category axis in KoolChart, which is CategroyAxis. Normally, CategroyAxis can be used to define the axis for text labels. It is also useful when you are not able to quantify label, but

you want to group labels, e.g. Departments - Management Division, Research Department, etc or Months - January, Febuary, etc.

For the numeric axis, there are three types of axis in KoolChart, which are LinearAxis, LogAxis and DateTimeAxis. LinearAxis is used for continuous data, which is general numeric value. LogAxis is used for the logarithmic function, and DataTimeAxis is used for the date & time formatted value.

The following table describes the properties of the axis.

Axes	Properties	Values	Descriptions
CategoryAxis	categoryField	The field name (e.g. Month)	Specifies the field name of the axis. You cannot omit this value in CategoryAxis.
	displayName	String (character string)	Specifies the text displayed in tooltips.
	title	Stirng (character string)	Specifies the title of the axis.
	labelJsFunction	JavaScript Function	Specifies the name of the function called to display the user-defined label.
LinearAxis	interval	Number	Specifies the gap between two consecutive labels.
	displayName	String (character string)	Specifies the text displayed in tooltips.
	minimum	Number	Specifies the minimum value of the axis label.
	maximum	Number	Specifies the maximum value of the axis label.
	title	Stirng (character string)	Specifies the title of the axis.
	labelJsFunction	Javascript Function	Specifies the name of the function called to display the user-defined label.
DateTimeAxis	dataUnits	milliseconds, seconds, minutes, hours, days, weeks, months, years	Specifies the unit of the data.
	labelUnits	milliseconds, seconds, minutes, hours, days, weeks, months, years	Specifies the unit of the label.
	title	Stirng (character string)	Specifies the title of the axis.
	interval	Number	Specifies the interval (time, date) between two consecutive labels. (If there is no enough space between two consecutive labels, the value will be ignored.)
	dataInterval	Number	Specifies the interval between two consecutive data. For example, If you set

			as dataUnits="second" and the actual interval between two consecutive data is 3 seconds, you should set as dataInterval=4. It means that KoolChart treats every 3 seconds as a unit and renders it at 4th position. (In some types of the chart, this property can be ignored.)
	displayName	String (character string)	Specifies the text displayed in tooltips.
	displayLocalTime	false|true (default: false)	Specifies whether or not the local time zone is used. If the value is false, Greenwich Mean Time will be used.
	labelJsFunction	Javascript Function	Specifies the name of the function called to display the user-defined label.
LogAxis	interval	Multiplier of 10	Specifies the lables as the multiplier of 10.
	minimum	Number	Specifies the minimum value of the axis label.
	maximum	Number	Specifies the maximum value of the axis label.
	title	Stirng (character string)	Specifies the title of the axis.
	displayName	String (character string)	Specifies the text displayed in tooltips.
	labelJsFunction	JavaScript Function	Specifies the name of the function called to display the user-defined label.

<Table 8 Properties: Axis>

6.2.1. CategoryAxis and LinearAxis

In the following example, CategoryAxis is defined as the horizontal axis. The value of CategoryField should be the same to the name of the data item in the chart. In this example, the Month field in the dataset is used as the category name of the horizontal axis. If you do not define the axis, the default axis will be LinearAxis.

```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
  <Options>
    <Caption text="Anual Report"/>
  </Options>
  <Column3DChart showDataTips="true">

    <horizontalAxis>   // CategoryAxis is defined as the horizontal axis.
        <CategoryAxis categoryField="Month" title="Category Axis" />
    </horizontalAxis>
```

```
    <verticalAxis>   // LinearAxis is defined as the vertical axis.
        <LinearAxis maximum="3500" title="Linear Axis"/>
    </verticalAxis>
    <series>
        <Column3DSeries yField="Profit" displayName="Profit">
            <showDataEffect>
                <SeriesInterpolate />
            </showDataEffect>
        </Column3DSeries>
    </series>
  </Column3DChart>
</KoolChart>
```

<Example 14 Example: CategoryAxis and LinearAxis>

6.2.2. DateTimeAxis and LogAxis

```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
  <Options>
      <Caption text="Anual Report"/>
  </Options>
  <Line2DChart showDataTips="true">

      <horizontalAxis> // DataTimeAxis is defined as the horizontal axis.
          <DateTimeAxis dataUnits="days" labelUnits="days" title="DateTime Axis" interval="3"
          displayName="Date" displayLocalTime="true"/>
      </horizontalAxis>

      <verticalAxis> // LogAxis is defined as the vertical axis.
          <LogAxis title="Log Axis" interval="10" minimum="10" maximum="10000" />
      </verticalAxis>
      <series>
          <Line2DSeries xField="Date" yField="Profit" displayName="Profit"/>
      </series>
  </Line2DChart>
</KoolChart>
```

<Example 15 Example: DateTimeAxis and LogAxis>

*** Note: If you want to use DataTimeAxis, you must define the DateTime field in the series.**

For example, if you want to use DateTimeAxis for the horizontal axis in the Column chart, you must specify xField (In the Bar chart, you must define yField for the vertical axis.). xField represents the field name of the corresponding time range. The usage of xField is same as that of categoryField in

CategoryAxis, but you must define it as the property of the series (not the property of <DateTimeAxis>).

6.3. Column 2D Charts

The Column2D chart starts with <Column2DChart> and ends with </Column2Dchart>. You can define <horizontalAxis>, <verticalAxis>, <backgroundElements>, <series>, etc., as the children of <Column2DChart>.

The following is an example of the Column2D chart with a single dataset (single series). In this layout, <Column2DSeries> is defined as a child of <Column2DChart>, which means the data will be represented as a column series.

The data representation of the chart is defined in <series>. You should make sure that the properties of <series> are correctly defined. The Chart node (e.g. Column2DChart, Bar2Dchart, etc) is the parent node of <series>, and it determines the overall appearances of the chart.

```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Anual Report"/>
        </Options>
        <Column2DChart showDataTips="true">
                <horizontalAxis>
                        <CategoryAxis categoryField="Month"/>
                </horizontalAxis>
                <series>
                        <Column2DSeries yField="Profit" itemRenderer="SemiCircleColumnItemRenderer">
                                <showDataEffect>
                                        <SeriesInterpolate/>
                                </showDataEffect>
                            <fill>
                                <SolidColor color="0xFF0000" alpha="0.5"/>
                            </fill>
                            <stroke>
                                <Stroke color="0xFFFF00" weight="1"/>
                            </stroke>
                        </Column2DSeries>
                </series>
        </Column2DChart>
</KoolChart>
```

<Example 16 Example: Column2D Chart >

The *itemRenderer* property of <Column2DSeries> determines the shape of each data item. The valid values of the *itemRenderer* property of the ColumnSeries series are SemiCircleColumnItemRenderer and GradientColumnItemRenderer.

- SemiCircleColumnItemRenderer
- GradientColumnItemRenderer

6.4. Column 3D Charts

The Column3D chart starts with <Column3DChart> and ends with </Column3Dchart>. <Column3DSeries> is the series of the Column3D chart. The following layout shows how to define the multiple series of the Column3D chart.

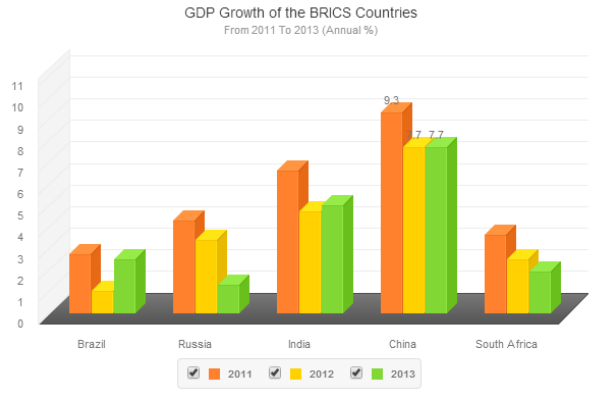
```
<KoolChart>
    <Column3DChart showDataTips="true" type="clustered">
            // Creating a Column3D chart.
    <horizontalAxis// Using CategoryAxis in the X-axis and the value of the categoryField property is Month.
        <CategoryAxis categoryField="Month"/>
    </horizontalAxis>
    <series> // Setting the series node.
        // Two column series are defined. (Profit and Cost are a pair of data)
       <Column3DSeries yField="Profit">
                    <showDataEffect> // Using the animation effect for data rendering.
                    <SeriesInterpolate/>
                    </showDataEffect>
        </Column3DSeries>
        <Column3DSeries yField="Cost"> // Using Cost for the Y-axis.
                    <showDataEffect>
                       <SeriesInterpolate/>
                    </showDataEffect>
        </Column3DSeries>
  </series></Column3DChart></KoolChart>
```

<Example 17 Example: Column3D Chart>

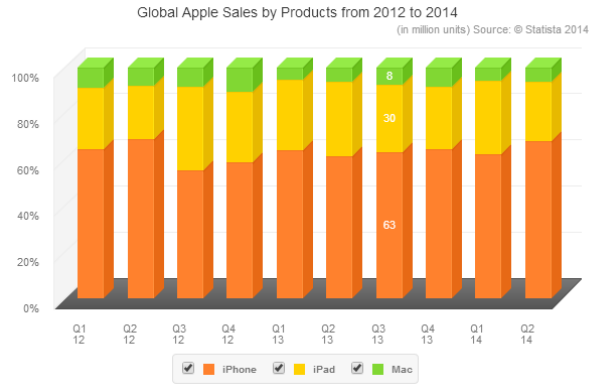
The *type* property of the Column chart determines the display method of the chart. The following four types are used in the Column chart.

- clustered: This is the default value for the Column chart. The clustered type is mostly used in the multiple dataset (multiple series).
- stacked: Data are stacked on the top of each other. Each data represents the cumulative value of the data beneath it.
- overlaid: Data are represented as the overlapped columns. The data in the back can be hidden.
- 100%: Data are stacked on the top of each other, adding up to 100%. Each data represents as a percentage.

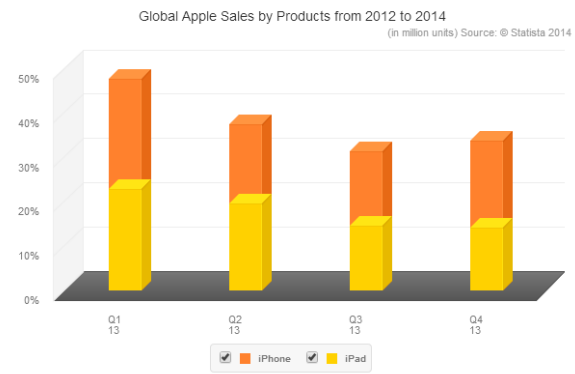
The following figures show the outputs of each type of the Column chart.



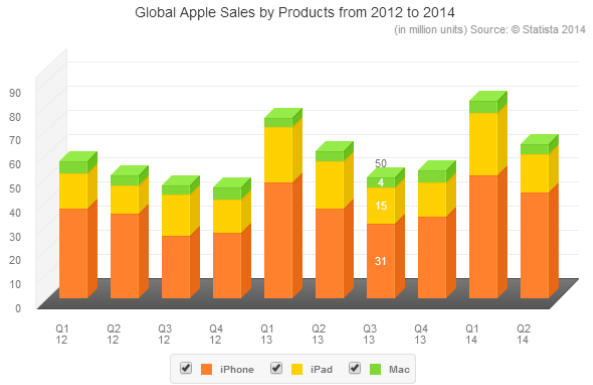
<type="clustered">



<type="100%">



<type="overlaid">



<type="stacked">

<Figure 4 Output: Types of Column Charts>

6.5. Cylinder 3D Charts

6.5.1. Cylinder 3D Column Charts

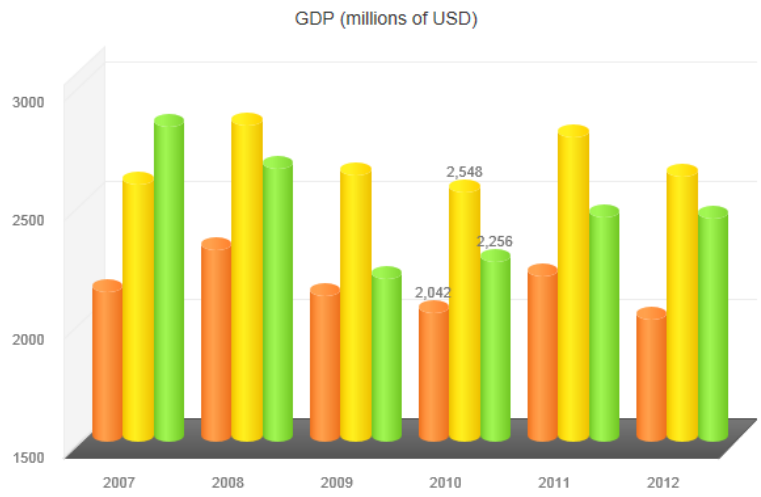
The Cylinder3D column chart is the same to the Column3D chart except that the shapes of the column are different. The cylinder-shaped column is used instead of the cube-shpaed column for the Cylinder3D column chart. All other properties of the Cylinder3D column in the layout are the same to those of the Column3D chart

You can create a Cylinder3D column chart by using the layout of the Column3D chart and just changing the value of the *itemRenderer* property to "CylinderItemRender".

The following layout shows how to create a Cylinder3D column chart by changing the *itemRenderer* property in the layout of the Column3D chart shown in <Example 17 Example: Column3D Chart>.

```
<KoolChart>
    <Column3DChart showDataTips="true" type="clustered">
            //Creating a Cylinder3D column chart.
    <horizontalAxis//Using CategoryAxis in the X-axis and the value of the categoryField property is Month.
        <CategoryAxis categoryField="Month"/>
    </horizontalAxis>
    <series> //Setting the series node.
        // Two column series are defined. (Profit and Cost are a pair of data)
       <Column3DSeries yField="Profit" itemRenderer="CylinderItemRenderer">
                        <showDataEffect> //Using the animation effect for data rendering.
                          <SeriesInterpolate/>
                        </showDataEffect>
        </Column3DSeries>
        <Column3DSeries yField="Cost"// Using Cost for the Y-axis.
itemRenderer="CylinderItemRenderer">
                        <showDataEffect>
                          <SeriesInterpolate/>
                        </showDataEffect>
        </Column3DSeries>
</series></Column3DChart></KoolChart>
```

<Example 18 Example: Cylinder3D Column Chart>

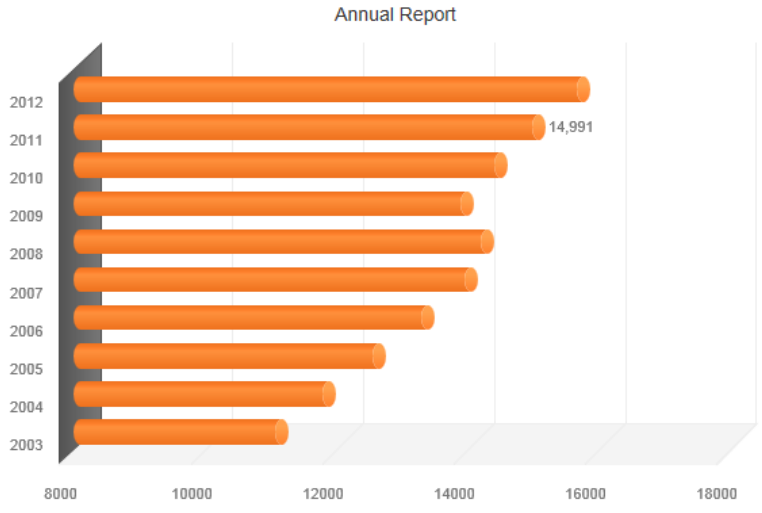


<*Figure 5 Output: Cylinder3D Column Chart*>

6.5.2. Cylinder 3D Bar Charts

The cylinder-shaped bar is used instead of the cube-shpaed bar for the Cylinder3D Bar chart. All other properties of the Cylinder3D Bar chart in the layout are indentical to those of the Bar3D chart.

You can create the Cylinder3D Bar chart by using the layout of the Bar3D chart and just changing the value of the *itemRenderer* property to "BarCylinderItemRender". For the details of how to create the layout of the Cylinder3D Bar chart, please refer to <6.5.1 Cylinder 3D Column Charts>.



<*Figure 6 Output: Cylinder3D Bar Chart*>

6.6. Bar 2D Charts

The Bar chart is the same to the Column chart except that the numeric data is displayed along the X-axis (horizontal axis).

The Bar2D chart starts with <Bar2Dchart> and ends with </Bar2Dchart>.

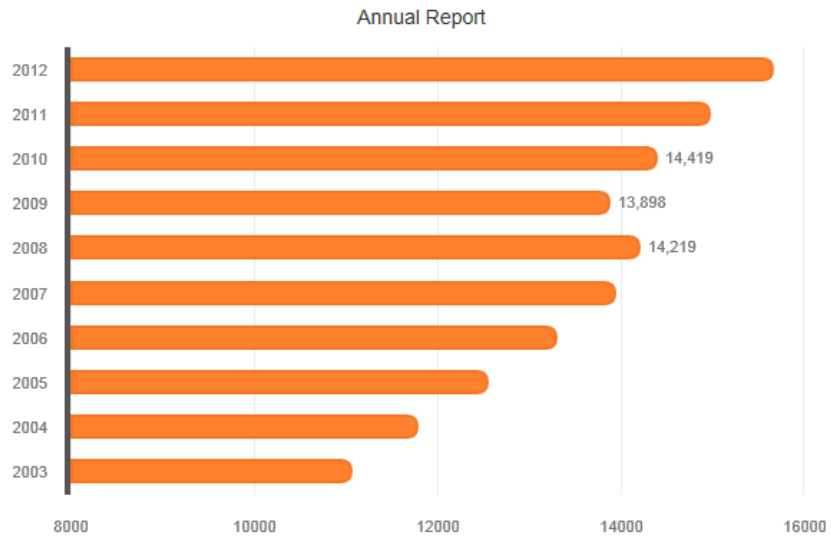
The valid values of the *itemRenderer* property of the Bar2D chart are SemiCircleBarItemRenderer and GradientBarItemRenderer.

- SemiCircleBarItemRenderer
- GradientBarItemRenderer

You can refer to the API document (./Docs/api) for further information of the properties of <Bar2Dchart>.

```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Anual Report"/>
        </Options>
        <Bar2DChart showDataTips="true">
                <verticalAxis>
                        <CategoryAxis categoryField="Month"/>
                </verticalAxis>
                <series>
                        <Bar2DSeries xField="Profit" itemRenderer="SemiCircleBarItemRenderer">
                                <showDataEffect>
                                        <SeriesInterpolate/>
                                </showDataEffect>
                        </Bar2DSeries>
                </series>
        </Bar2DChart>
</KoolChart>
```

<Example 19 Example: Bar2D Chart>



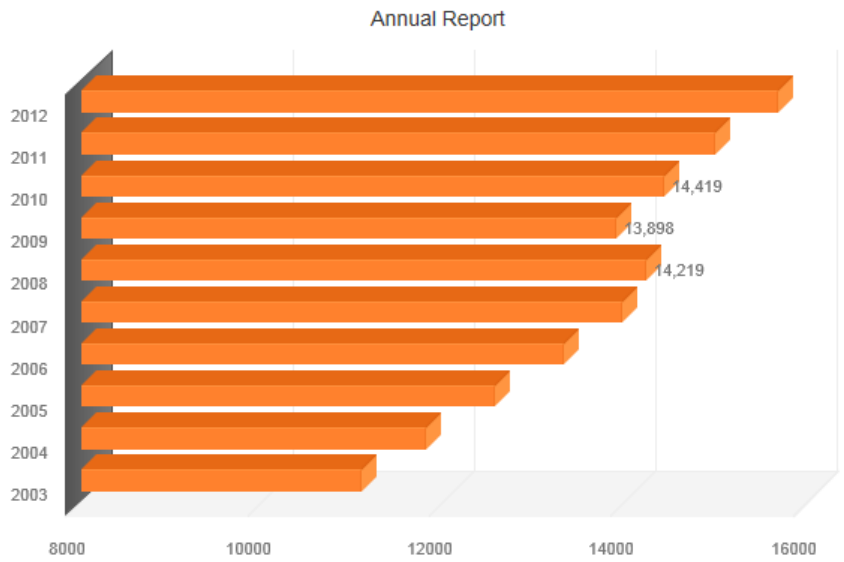
<*Figure 7 Output: Bar2D Chart*>

6.7. Bar 3D Charts

The definition of the Bar3D chart starts with <Bar3Dchart> and ends with </Bar3Dchart>.
You can refer to the API document (./Docs/api) for further information of the properties of <Bar3Dchart>.

```
<KoolChart>
    <Options>
        <Caption text="Annual Report"/>
    </Options>
    <Bar3DChart showDataTips="true">
        <verticalAxis>      //Using CategoryAxis in the Y-axis (vertical axis) and the value of the categoryField
property is Month.
            <CategoryAxis categoryField="Month" />
        </verticalAxis>
        <series>
            <Bar3DSeries xField="Profit">   //Using Profit for the X-axis (xField)
                <showDataEffect>
                    <SeriesInterpolate />
                </showDataEffect>
            </Bar3DSeries>
        </series>
    </Bar3DChart>
</KoolChart>
```

<*Example 20 Example: Bar3D Chart*>



<Figure 8 Output: Bar3D Chart>

6.8. Pie and Doughnut Charts

The definition of the Pie2D chart and the Doughnut2D charts start with <Pie2Dchart> and ends with </Pie2Dchart>. Also the definition of the Pie3D chart and the Doughnut3D chart starts with <Pie3Dchart> and ends with </Pie3Dchart>. The series of the Pie2D chart and the Doughnut2D chart is <Pie2DSeries> and the series of the Pie3D chart and the Doughnut3D chart is <Pie3DSeries>.

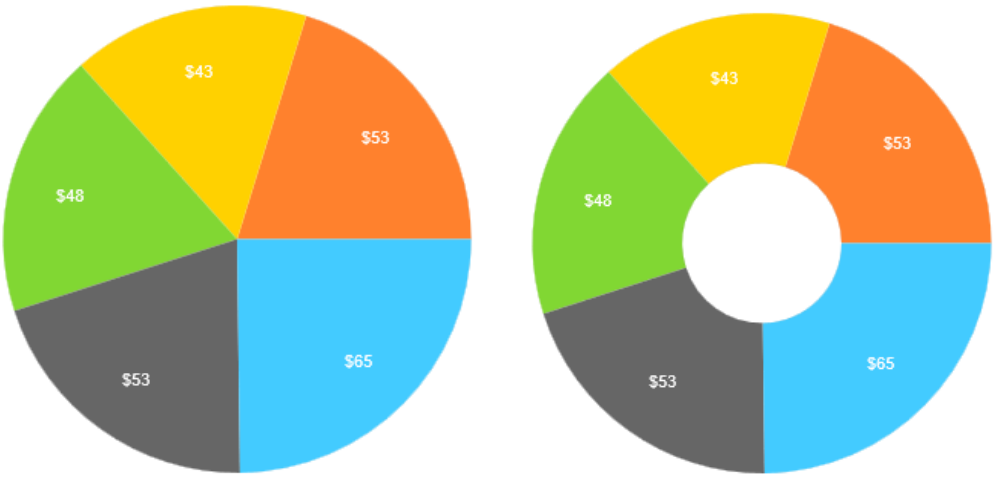
The layouts of the Pie chart and the Doughnut chart are the same except that the values of the *innerRadius* property are different.

The value of the *innerRadius* property is the percentage value of the hole compared to the entire pie's radius. The valid range of the value is from 0 to 1. The value of the *innerRadius* property for the Pie chart is 0 and as the value reaches 1, the size of the inner hole increases (0 = Pie chart).

```
<KoolChart>
  <Pie2DChart innerRadius="0" showDataTips="true">
    <series>
        <Pie2DSeries
            nameField="Month" //The field name displayed in tooltips
            field="Profit" // The numeric data field.
            depth="0.05" // The depth of the 2D chart (value range: 0 ~ 0.2)
             labelPosition="callout">

            <showDataEffect>    <!--Using the animation effect for data rendering.-->
```

```
                <SeriesInterpolate/>
            </showDataEffect>

            <fills> <!-- colors are used alternately to fill the slices. -->
                <SolidColor color="0xff0000"/>
                <SolidColor color="0xffffff"/>
            </fills>
        </Pie2DSeries>
    </series>
  </Pie2DChart>
<KoolChart>
```

<*Example 21 Example: Pie2D Chart*>



<innerRadius = 0> <innerRadius = 0.33>

<*Figure 9 Output: Pie Chart and Doughnut Chart* >

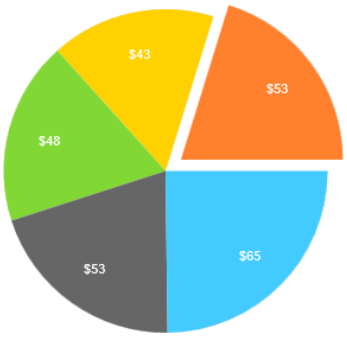
As you can see from the examples above, the value of the *innerRadius* property determines whether the layout will generate the Pie chart or the Doughnut chart.

The value of the *labelPosition* property specifies the position where to place the label of the Polar chart. The following describes the values of the *labelPosition* property.

- none: Does not draw the value of numeric data.
- inside: Draws label inside the chart.
- outside: Draws label outside the chart.
- callout: Draws label outside the chart with the line.
- insideWithCallout: Draws label inside the pie, but if not enough space in the slice, the "callout" method is used.

You can explode out a single slice by clicking a slice of the Pie chart or the Doughnut chart. The default value of the *explodable* property is true.

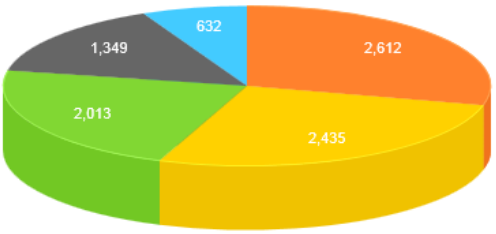
```
<KoolChart>
        <Options>
                <Caption text="Annual Report"/>
                <SubCaption text="2008"/>
        </Options>
        <Pie2DChart          explodable="true"          innerRadius="0"
showDataTips="true">
        ......
<KoolChart>
```



<Example 22 Layout and Output: Pie Chart with explodable="true">

```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Annual Report"/>
                <SubCaption text="2008"/>
        </Options>
        <Pie3DChart showDataTips="true" explodable="false">
                <series>
                        <Pie3DSeries nameField="Month" field="Profit" labelPosition="inside">
                                <showDataEffect>
                                        <SeriesZoom/>
                                </showDataEffect>
                        </Pie3DSeries>
                </series>
        </Pie3DChart>
</KoolChart>
```

<Example 23 Example: Pie3D Chart>

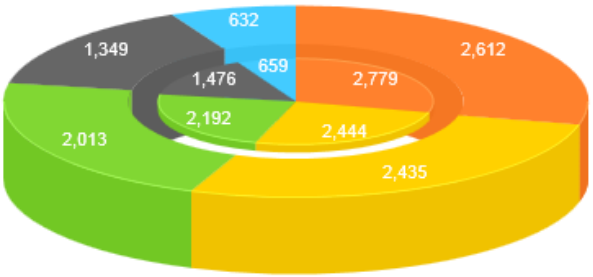


<Figure 10 Output: Pie3D Chart>

The stacked Pie 3D chart can be created using two series. The following is an example that shows how to create a stacked Pie 3D chart.

```
<KoolChart cornerRadius="12" borderStyle="solid">
    <Options>
        <Caption text="Annual Report" />
        <SubCaption text="2008" />
    </Options>
    <Pie3DChart showDataTips="true" explodable="false">
        <series>
            <Pie3DSeries nameField="Month" field="Profit" displayName="Profit" labelPosition="inside">
                <showDataEffect>
                    <SeriesZoom />
                </showDataEffect>
            </Pie3DSeries>
            <Pie3DSeries nameField="Month" field="Cost" displayName="Cost" labelPosition="inside">
                <showDataEffect>
                    <SeriesZoom />
                </showDataEffect>
            </Pie3DSeries>
        </series>
    </Pie3DChart>
</KoolChart>
```

<Example 24 Pie Stacked 3D Chart>



<Figure 10 Output: Stacked Pie 3D Chart>

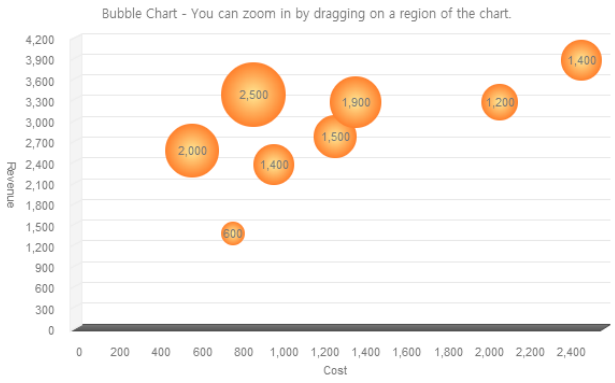
6.9. Bubble 3D Charts

The definition of the Bubble3D chart starts with <Bubble3Dchart> and ends with </Bubble3Dchart>. The series of the Bubble3D chart is <Bubble3DSeries>. The Bubble3D chart has a set of three numeric values. Each value represents the values of X, Y axes and radius. In the Bubble3D chart, the size of the bubble is determined by the third value, radius.

As the series represents numeric data, you must set the *xField*, *yField* and *radiusField* properties of <Bubble3DSeries>.

```
<KoolChart>
  <Bubble3DChart showDataTips="true">
    <series>
        <Bubble3DSeries xField="Profit" yField="Cost" radiusField="Revenue">

            <showDataEffect>
                <SeriesInterpolate />
            </showDataEffect>
            <fills>   <!-- 6 colors are used alternately to fill the bubbles. -->
                <SolidColor color="0xFF0000"/>
                <SolidColor color="0x00FF00"/>
                <SolidColor color="0x0000FF"/>
                <SolidColor color="0xFF00FF"/>
                <SolidColor color="0xFFFF00"/>
                <SolidColor color="0xFFFFFF"/>
            </fills>
        </Bubble3DSeries>
</series></Bubble3DChart></KoolChart>
```

<Example 25 Example: Bubble3D Chart>

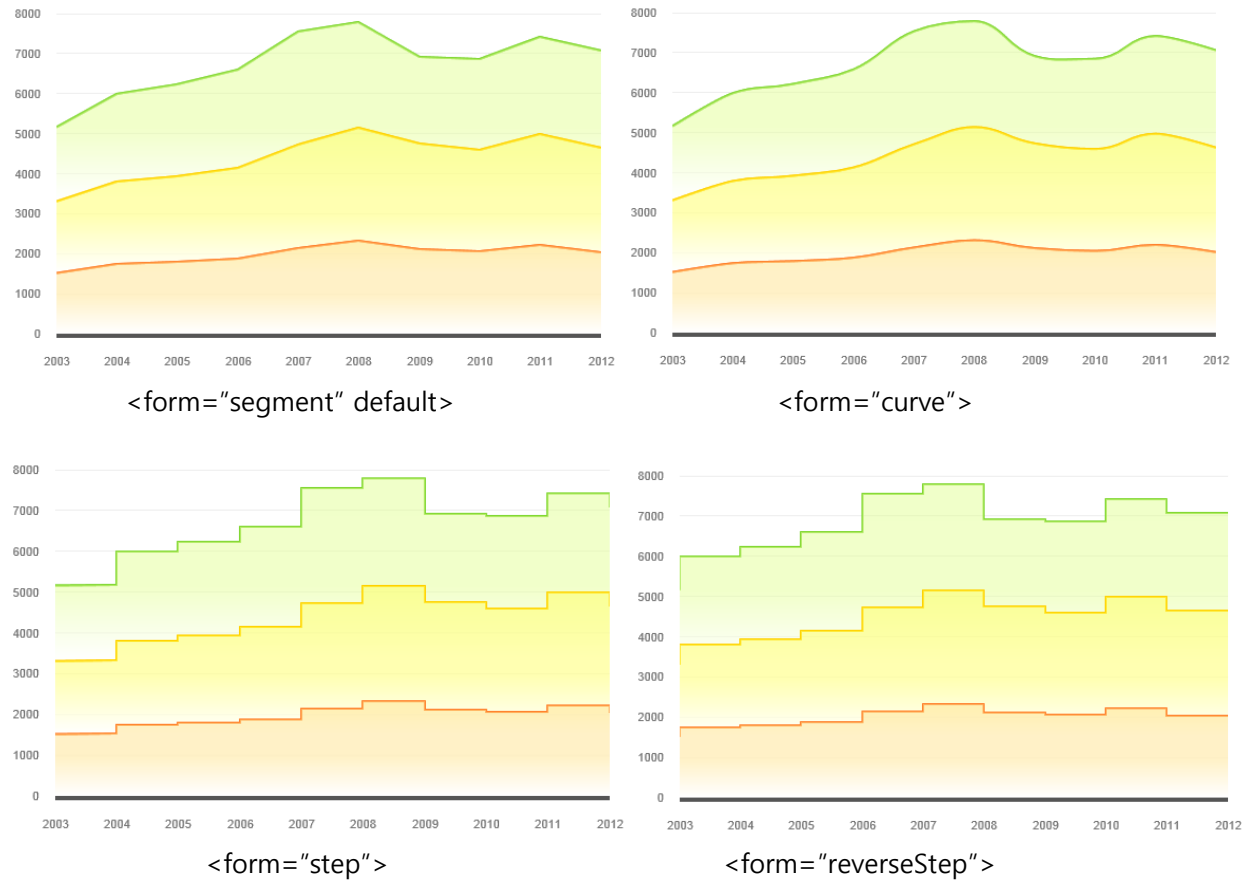


<Figure 12 Output: Bubble3D Chart>

6.10. Area Charts

There are three types of the Area chart in KoolChart, which are overlaid, stacked and 100%. You can specify the type of the Area chart by setting the *type* property of the chart. The display methods of the Area chart are identical to those of the Column chart. The definition of the Area chart starts <Area2DChart> and ends with </Area2DChart>.

The *form* property of <Area2DSeries> determines how to display the series data.



<form="segment" default> <form="curve">

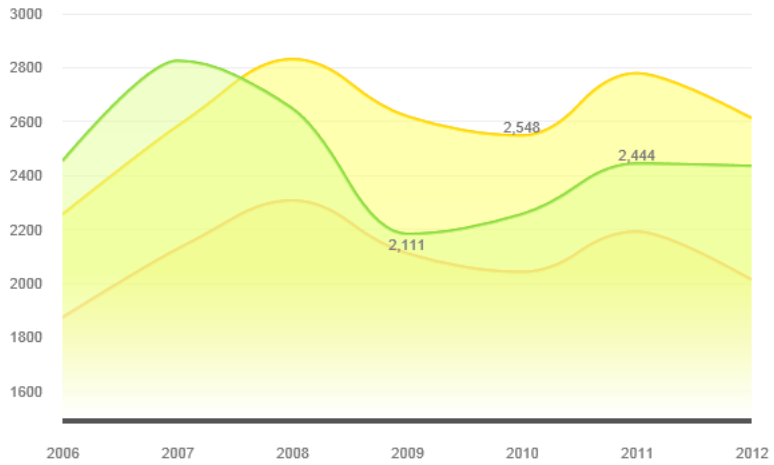
<form="step"> <form="reverseStep">

<*Figure 13 Outputs: Types of Area Chart*>

```
<KoolChart>
    <Area2DChart showDataTips="true" type="stacked">
        <horizontalAxis>
            <CategoryAxis categoryField="Month" />
        </horizontalAxis>
        <series>
            <Area2DSeries yField="Profit" form="curve" displayName="Profit">
                <showDataEffect>
                    <SeriesInterpolate />
```

```
                    </showDataEffect>
                </Area2DSeries>
                <Area2DSeries yField="Cost" form="curve" displayName="Cost">
                    <showDataEffect>
                        <SeriesInterpolate />
                    </showDataEffect>
                </Area2DSeries>
                <Area2DSeries yField="Revenue" form="curve" displayName="Revenue">
                    <showDataEffect>
                        <SeriesInterpolate />
                    </showDataEffect>
                </Area2DSeries>
        </series>
    </Area2DChart>
</KoolChart>
```

<Example 26 Example: Area Chart>



<Figure 14 Output: Area Chart>

6.11. Plot Charts

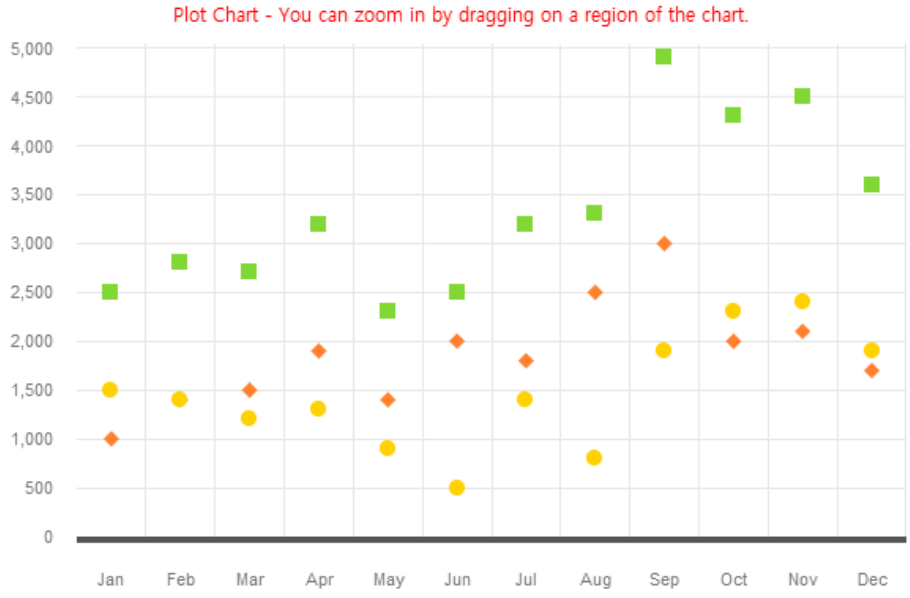
The Plot chart is used to represent data in Cartesian coordinates where each data point has one value that determines its position along the x-axis, and one value that determines its position along the y-axis. The definition of the Plot2D chart starts with <Plot2Dchart> and ends with </Plot2Dchart>. The series of the Plot2D chart is <Plot2DSeries>.

Note: Both of *xField* and *yField* are required for each PlotSeries in the Plot2D chart.

The following is a sample layout and its output for the Plot chart.

```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
        <Options>
                <Legend useVisibleCheck="true" />
        </Options>
        <Plot2DChart showDataTips="true"> // Creating a Plot chart
                <verticalAxis>
                    <LinearAxis maximum="3500" />
                </verticalAxis>
                <horizontalAxis>
                    <LinearAxis maximum="2800" />
                </horizontalAxis>
                <series>
                    <Plot2DSeries xField="Cost" yField="Profit" radius="5" displayName="Cost/Profit">
                    </PlotSeries>
                    <Plot2DSeries xField="Revenue" yField="Profit" radius="5" displayName="Revenue/Profit">
                    </Plot2DSeries>
                    <Plot2DSeries xField="Cost" yField="Revenue" radius="5" displayName="Cost/Revenue">
                    </Plot2DSeries>
                </series>
        </Plot2DChart>
</KoolChart>
```

< Example 27 Example: Plot Chart>



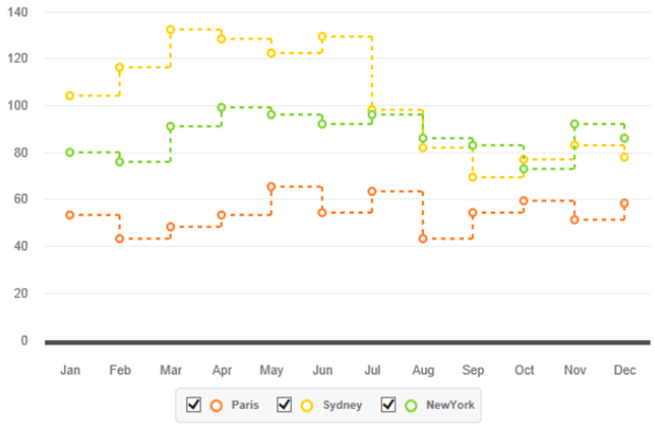
< Figure 15 Output: Plot Chart>

6.12. Line Charts

Unlike other Cartesian charts, the Line chart does not have the *type* property. It only has the overlaid type. The series of the Line chart is <Line2DSeries>. Like <Area2DSeries>, <Line2Dseries> has the *form* property that specifies the way in which the series data is shown in the chart. The valid values of the *form* property are segment, curve, step and reverseStep.

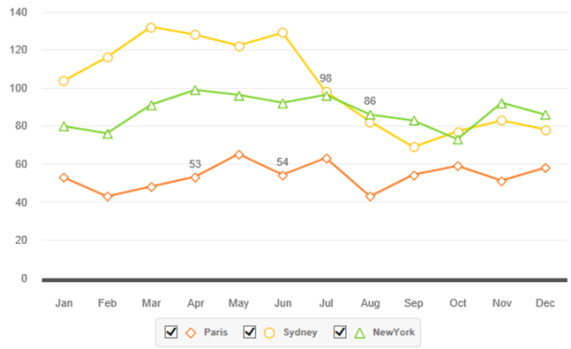
```
<KoolChart cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Anual Report" />
        </Options>
        <Line2DChart showDataTips="true">
                <horizontalAxis>
                        <CategoryAxis categoryField="Month" />
                </horizontalAxis>
                <series>
                        <Line2DSeries yField="Profit" form="step">
                                <showDataEffect>
                                        <SeriesInterpolate />
                                </showDataEffect>
                        </Line2DSeries>
                        <Line2DSeries yField="Cost" form="step">
                        </Line2DSeries>
                        <Line2DSeries yField="Revenue" form="step">
                        </Line2DSeries>
                </series>
        </Line2DChart>
</KoolChart>
```

< *Example 28 Example: Line Chart*>



< *Figure 16 Output: Line Chart*>

In the Line chart, you can define the shape (circle, triangle, rectangle, etc.) displayed at the data points (x, y coordinates) by using itemRenderer for the series data.



< *Figure 17 Output: Line Chart with itemRenderer*>

```
<KoolChart cornerRadius="12" borderStyle="solid" paddingTop="10" paddingBottom="20" paddingRight="20"
paddingLeft="20">
        <Options>
                <Caption text="Anual Report" />
        </Options>
<Line2DChart showDataTips="true">
        <horizontalAxis>
        <CategoryAxis categoryField="Month"/>
    </horizontalAxis>
    <series>
        No. 1 <Line2DSeries yField="Profit"  radius="10" fill="0xFF0000"  itemRenderer="
DiamondItemRenderer"/>

        No. 2 <Line2DSeries yField="Cost"  radius="10" fill="0x00FF00" itemRenderer="CircleItemRenderer"/>

        No. 3 <Line2DSeries yField="Revenue"  radius="10" fill="0xFFFF00"
itemRenderer="TriangleItemRenderer"/>
    </series>
</Line2DChart>
</KoolChart>
```

<Example 29 Example: Line Chart with itemRenderer>

You can use the following properties for itemRenderer.

- radius: Specifies the size of the shape (circle, triangle, rectangle, etc). The center of the radius is the x, y coorinates of the data point.
- fill: Fills the shape with color.
- stroke: Specifies the border thickness of the shape.

The line thickness and the color of the Line chart are determined by lineStroke. Please refer to <Example 49 Example: Layout and Output of Line Chart with Stroke> for further information.

The followings are the values for the *itemRenderer* property of the Line chart.

- DiamondItemRenderer – No. 1 in the example, <Example 29 Example: Line Chart with itemRenderer>.
- CircleItemRenderer – No. 2 in the example, <Example 29 Example: Line Chart with itemRenderer>.
- TriangleItemRenderer – No. 3 in the example, <Example 29 Example: Line Chart with itemRenderer>.
- CrossItemRenderer - cross-shaped
- XShapeItemRenderer - X-shaped
- IShapeItemRenderer – I-shaped
- RectangleItemRenderer – Square-shaped

6.13. Dashed-Line Charts

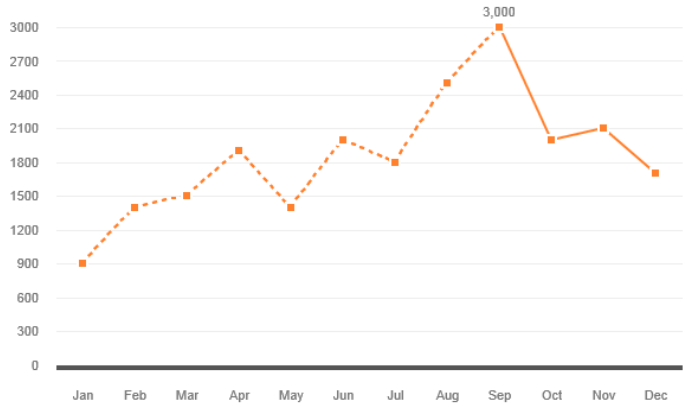
The Dashed-Line chart has all the features of the Line chart, and the format of the layout is identical to that of the Line chart. The following table shows the properties of the Dashed-Line chart.

Properties	Values	Descriptions
lineStyle	normal, dashLine	Specifies the line type. To create the Dashed-Line chart, set the value of the *lineStyle* property to dashLine.
dashLinePlacement,	before, after	If the dashLineSeperatePos properity is set, dashLinePlacement determines the place of the dashed-line (before or after dashLineSeperatePos).
dashLineSeperatePos	The index of data (uint)	If you want to use the dashed-line and the normal line at the same time in the chart, dashLineSeperatePos determines the position of the boundary between the dashed-line and the normal line. The position of dashLineSeperatePos is the index of data.
dashLinePattern	Number (pixel) (default: 10)	Specifies the length of each dash. For example, if you set the value to 5, both dashe and space will be 5 pixels in size. The larger the value, the better performance you can have. If you want to improve chart rendering performance, use the larger value of dashLinePattern.

<Table 9 Properties: Dashed-Line Chart>

```
<KoolChart backgroundColor="0xFFFFFF"  cornerRadius="12" borderStyle="solid">

        <Line2DChart showDataTips="true" >
                <horizontalAxis>
                        <CategoryAxis categoryField="Month" padding="0.5" />
                </horizontalAxis>
                <series>
                        <Line2DSeries labelPosition="up"
                yField="Profit" radius="4
                lineStyle="dashLine"          // Creating a Dashed-Line chart
                dashLineSeperatePos="5"     // The position of the boundary (The 6th data)
                dashLinePlacement="before" // Displaying dashed-line first
                itemRenderer=" RectangleItemRenderer">
                        </Line2DSeries>
                </series>
        </Line2DChart>
</KoolChart>
```

<Example 30 Example: Dashed-Line Chart>



<Figure 18 Output: Dashed-Line Chart>

6.14. Combination Charts

The Combination chart can be created by using two different chart types. The following example describes how to create the Combination chart using the Column chart and the Line chart.

- The Column chart represents series data using <Column2DSeries> or <Column3DSeries>. However, since this example chart uses two series of the Column chart and one series of the Line chart, <Column2DSet> or <Column3DSet> must be defined to combine the series of the Column chart.

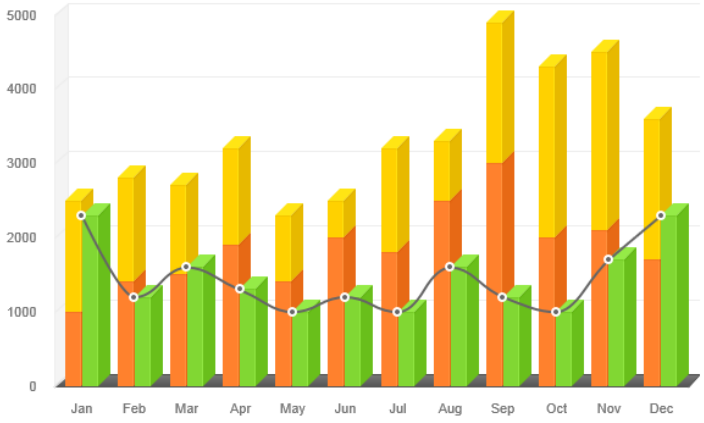
- To create the Combination 2D chart, define <Column2DSet> and then create <Column2DSeries> as the child of <Column2DSet>.
- The type of <Column2DSet> or <Column3DSet> is the same to those of other charts. The default value is overlaid.
- Create <Line2DSeries> as the sibling of <Column2DSet>.

The following layout shows how to create the Combination chart using the Column chart and the Line chart as described above.

```
<KoolChart>
    <Combination2DChart showDataTips="true">
        <horizontalAxis>
            <CategoryAxis categoryField="Month"/>
        </horizontalAxis>

        <series>
            <Column2DSet type="clustered">
                <series>
                    <Column2DSeries yField="Profit" displayName="Profit"/>
                    <Column2DSeries yField="Revenue" displayName="Revenue"/>
                </series>
            </Column2DSet>
            <Line2DSeries  yField="Cost" displayName="Cost">
                <lineStroke> // Specifies the thickness and the color of the line.
                    <Stroke color="0xFFCC00" weight="4"/>
                </lineStroke>
            </Line2DSeries>
        </series>
    </Combination2DChart>
</KoolChart>
```

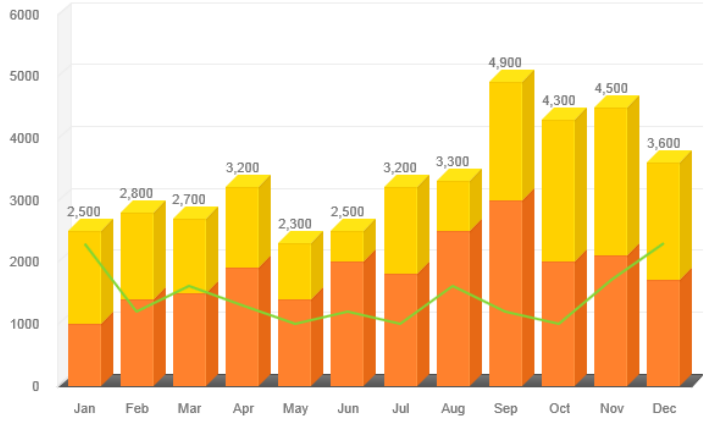
<Example 31 Example: Combination Chart>

The following outputs are the results of running the above layouts.



<Figure 19 Output: Combination Chart with type="clustered">

If you want to display the stacked column, set the value of the *type* property to "stacked".



<Figure 20 Output: Combination Chart with type="stacked">

6.15. Real-Time Charts

The Real-time chart is useful when you need to monitor real-time data that is changing continuously.

The definition of the Real-time chart starts with <RealTimeChart> and ends with </RealTimeChart>. For the series of the Real-time chart, you can specify the Column series, the Line series or the Area series. <HttpsServiceRepeater> has the role of initiating RPC (Remote Procedure Call), which is the protocol used to read data from the server and should be defined in the layout.

To create the Real-time chart, you should keep in mind the following rules.

1. <HttpServiceRepeater> must be created as the first-descendant of <KoolChart>.
2. Set the value of the *target* property to the ID of the Real-time chart.
3. The URL of <HttpServiceRepeater> is the path of the XML data. The format of the XML data must be written as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<items>                          // Starts with <items>.
    <item>                       // Each data item should be enclosed between <item> and </item>.
        <Time>13:8:27</Time>
        <Volume>5527</Volume>
        <Price>309</Price>
    </item>
</items>
```

<Example 32 Example: Real-time Chart>

When you create the Real-time chart in your web page, you don't have to feed the initial data.

* Note: If the value of the *interval* property in the Real-time chart is 5 (5 seconds), KoolChart will pull data from the server every 5 seconds but in the real service environment, several factors (network link status, communication status, traffic, etc.) may cause some delays. The server-side programmer should consider those factors to develop the program that pull the real-time data.

KoolChart supports two data types to present the real-time chart.

1. Category-based data: As soon as the Real-time chart receives the data from the server, it displays the predefined number of data immediately and then displays the next data one by one. For example, if you set the value of displayDataSize to 20, it will display 20 data at once and then refresh the chart with the next data.

2. Time-based data: The Real-time chart displays data based on the time. For example, if you set the value of timePeriod to 3600 seconds (1 hour) and set the value of interval to 3, the Real-time chart will display data every 3 seconds for 1 hour. After 1 hour, it will refresh the chart with the next data (The value of interval should be the same to that of <HttpServiceRepeater>).

6.15.1. Creating Real-Time Charts Based on the Number of Data (Using <CategoryAxis>)

If you want to display the Real-time chart to represent the category-based data, you must use <CategoryAxis>.

<CategoryAxis> is not continuous, and it recognizes each category field as a separate field. For example, if you have the date or time related data, <CategoryAxis> will recognize it as a simple string. The

drawback of <CategoryAxis> is that it may display duplicate data along the axis as it is not continuous.

```
<KoolChart cornerRadius="12" borderStyle="solid">
        <RealTimeChart id="chart" dataDisplayType="dataSize" displayDataSize="15" showDataTips="true">
                <horizontalAxis>
                        <CategoryAxis id="hAxis" categoryField="Time"/>
                </horizontalAxis>
                <series>
                        <Column2DSeries yField="Volume" displayName="Trading Volume"
itemRenderer="GradientColumnItemRenderer">
                                <fill> // fills the column with the color
                                        <SolidColor color="0xB0C759" />
                                </fill>
                        </Column2DSeries>
                </series>
        </RealTimeChart>
        <HttpServiceRepeater url="http://www.koolchart.net/realtimeSample /data.php" target="{chart}" interval="3"
method="get" /> // Performs data refreshes every 3 seconds
</KoolChart>
```

<Example 33 Example: Real-time Chart Using <CategoryAxis>>

6.15.2. Creating Real-Time Charts Based on Time (Using <DateTimeAxis>)

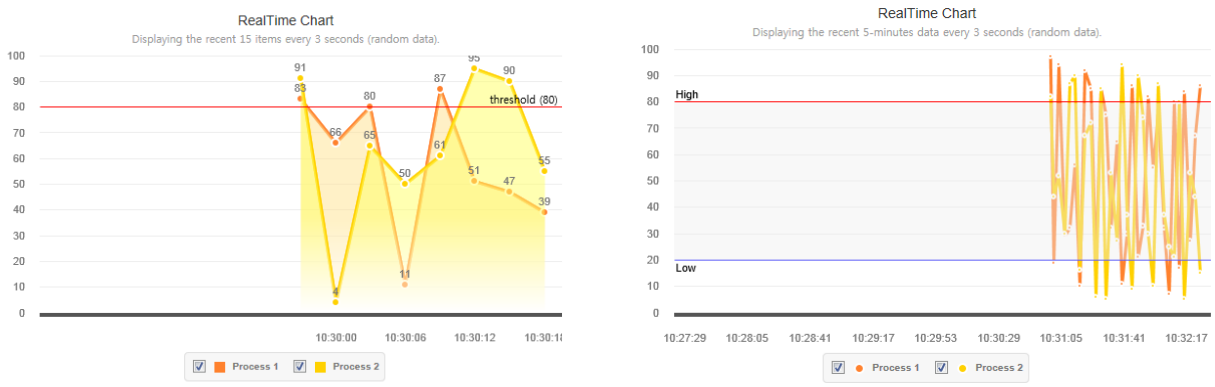
If you want to display the Real-time chart based on time, you have to use <DataTimeAxis>. <DateTimeAxis> is continuous and discards duplicate data. For example, if KoolChart receive the previously missing data (duplicated data) caused by some error, the data will be ignored. And if KoolChart cannot receive the data in the correct order, DateTimeAxis will display a blank for the missed data. You can use <DateTimeAxis> for the typical Rea-time chart.

```
<KoolChart cornerRadius="12" borderStyle="solid">
        <RealTimeChart id="chart" dataDisplayType="time" timePeriod="60" interval="3" showDataTips="true"> //
Performs data refreshes every 3 seconds for 60 seconds.
                <horizontalAxis>
              // Data refresh interval is 3 seconds, and the label interval is 9 seconds.
                        <DateTimeAxis dataUnits="seconds" labelUnits="seconds" dataInterval="3" interval="9"
displayLocalTime="true"/>
                </horizontalAxis>
                <series> // xField should be set because DateTimeAxis is used.
                        <Column2DSeries xField="Time" yField="Volume" displayName="Trading Volume"
```

```
itemRenderer="GradientColumnItemRenderer">
                                        <fill>
                                                <SolidColor color="0xB0C759" />
                                        </fill>
                                </Column2DSeries>
                        </series>
        </RealTimeChart>
        <HttpServiceRepeater url="http://www.koolchart.com/realtimeSample/data4.php" target="{chart}"
interval="3" method="get" />
</KoolChart>
```

<Example 34 Example: Real-time Chart Using <DateTimeAxis>>

*** Note: The value of the interval property of <RealTimeChart>, the value of the dataInterval property of <DateTimeAxis>, and the value of the interval property of <HttpServiceRepeater> should be same.**



<Figure 21 Outputs: Real-time Chart (CategoryAxis: Left, DateTimeAxis: Right)>

6.15.3. Using HttpServiceRepeater in Charts

Not just the Real-time chart, HttpServiceRepeater can be used in other types of charts.

For example, if you have created a Column3D chart and want to perform data refreshes every 1 minute, HttpServiceRepeater is useful. Use the layout of the Column3D chart and just add HttpServiceRepeater to the layout as follows:

```
<KoolChart cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Anual Report" />
        </Options>
```

```
        <Column3DChart id="chart" showDataTips="true">
                <horizontalAxis>
                        <CategoryAxis categoryField="Month" />
                </horizontalAxis>
                <series>
                        <Column3DSeries yField="Profit" displayName="Profit">
                                <showDataEffect>
                                        <SeriesInterpolate />
                                </showDataEffect>
                        </Column3DSeries>
                </series>
        </Column3DChart>

        <HttpServiceRepeater url="http://www.koolchart.com/ realtimeSample /singleData.php" target="{chart}"
        interval="60"/>
</KoolChart>
```

<Example 35 Example: Using HttpServiceRepeater in Column3D Chart>

6.16. Radar Charts

The definition of the Radar chart starts with <RadarChart> and ends with </RadarChart>. The series of the Radar chart is <RadarSeries>. Two kinds of axes (radialAxis and angularAxis) are used in the Radar chart. <radialAxis> corresponds to the radius of the chart, which maps numeric data to a distance between the center and the perimeter of the chart. <angularAxis> corresponds to the perimeter of the chart, and it takes values that are categories rather than numerics.

The following examples show three data formats (Table, Array, XML) for the annual household expenditure.

1. Table

	2005	2006	2007	2008
Food	100	100	180	150
Health Care	80	120	200	210
Transportation	70	115	100	240
Clothing	80	120	140	210
Education	90	160	130	200
Shelter	100	180	165	140
Leisure	76	120	130	170
Others	80	140	140	190

2. Array

```
var chartData = [{"catName":"Food", "year2005":100, "year2006":100, "year2007":180, "year2008":150}
                ,{"catName":"Health Care", "year2005":80, "year2006":120, "year2007":200, "year2008":210}
                ,{"catName":"Transportation", "year2005":70, "year2006":115, "year2007":100,
"year2008":240}
                ,{"catName":"Clothing", "year2005":80, "year2006":120, "year2007":140, "year2008":210}
                ,{"catName":"Education", "year2005":90, "year2006":160, "year2007":130, "year2008":200}
                ,{"catName":"Shelter", "year2005":100, "year2006":180, "year2007":165, "year2008":140}
                ,{"catName":"Leisure", "year2005":76, "year2006":120, "year2007":130, "year2008":170}
                ,{"catName":"Others", "year2005":80, "year2006":140, "year2007":140, "year2008":190}];
```

3. XML

```
<items>
        <item>
                <catName>Food</catName>
                <year2005>100</year2005>
                <year2006>100</year2006>
                <year2007>180</year2007>
                <year2008>150</year2008>
        </item>
        <item>
                <catName>Health Care</catName>
                <year2005>80</year2005>
                <year2006>120</year2006>
                <year2007>200</year2007>
                <year2008>210</year2008>
        </item>
          .
          .
          .
        <item>
                <catName>Others</catName>
                <year2005>80</year2005>
                <year2006>140</year2006>
                <year2007>140</year2007>
                <year2008>190</year2008>
        </item>
</items>
```

Based on the above data, the following layout can be created to generate a Radar chart.

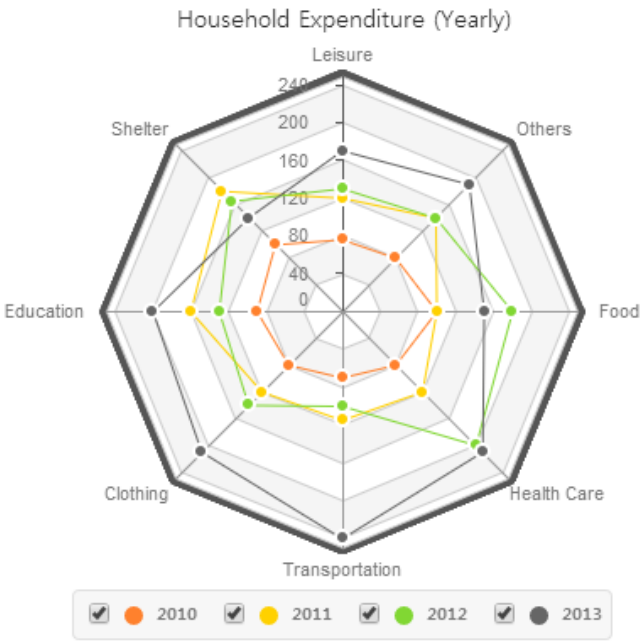
```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Household Expenditure (Yearly)"/>
                <SubCaption text="(2010~2013)"/>
                <Legend useVisibleCheck="true"/>
        </Options>
        <RadarChart id="chart1"
                    isSeriesOnAxis="false"
                    type="polygon"
                    showAllDataTips="false"
                    showDataTips="true">
                    <radialAxisRenderers>
                        <AxisRenderer horizontal="true" visible="true" styleName="hangingCategoryAxis" >
                        </AxisRenderer>
                        <AxisRenderer horizontal="false" visible="false" styleName="hangingCategoryAxis" >
                        </AxisRenderer>
                    </radialAxisRenderers>
                    <radialAxis>
                        <LinearAxis/>
                    </radialAxis>
                    <angularAxis>
                                        <CategoryAxis id="aAxis" categoryField="catName"
displayName="Category" />
                                </angularAxis>
                    <angularAxisRenderers>
                        <AngularAxisRenderer axis="{aAxis}" axisTitleStyleName="axisTitle"
styleName="hangingCategoryAxis"/>
                    </angularAxisRenderers>
                <series>
                        <RadarSeries field="year2010"
                                displayName="2010">
                                <showDataEffect>
                                        <SeriesInterpolate/>
                                </showDataEffect>
                                <stroke> <!-- Sets the line style of the shape displayed at the data point  -->
                        <Stroke color="0xE48701" weight="2"/>
                                </stroke>
                                <lineStroke> <!-- Sets the line style of the series -->
                                        <Stroke color="0xE48701" weight="2"/>
                                </lineStroke>
                                <areaFill> <!--  Fills the inner space with color  -->
                                        <SolidColor color="0xE48701" alpha="0.1"/>
                                </areaFill>
                                <fill> <!--  Fills the shape displayed at the data point with color -->
```

```
                        <SolidColor color="0xFFFFFF"/>
                </fill>
        </RadarSeries>

        <RadarSeries field="year2011"
                displayName="2011">
                <showDataEffect>
                        <SeriesInterpolate/>
                </showDataEffect>
                <stroke>
                        <Stroke color="0xB0C759" weight="2"/>
                </stroke>
                <lineStroke>
                        <Stroke color="0xB0C759" weight="2"/>
                </lineStroke>
                <areaFill>
                        <SolidColor color="0xB0C759" alpha="0.1"/>
                </areaFill>
                <fill>
                        <SolidColor color="0xFFFFFF"/>
                </fill>
        </RadarSeries>
        <RadarSeries field="year2012"
                displayName="2012">
                <showDataEffect>
                        <SeriesInterpolate/>
                </showDataEffect>
                <stroke>
                        <Stroke color="0x0A80C4" weight="2"/>
                </stroke>
                <lineStroke>
                        <Stroke color="0x0A80C4" weight="2"/>
                </lineStroke>
                <areaFill>
                        <SolidColor color="0x0A80C4" alpha="0.1"/>
                </areaFill>
                <fill>
                        <SolidColor color="0xFFFFFF"/>
                </fill>
        </RadarSeries>
        <RadarSeries field="year2013"
                displayName="2013">
                <showDataEffect>
                        <SeriesInterpolate/>
```

```
                                    </showDataEffect>
                                    <stroke>
                                            <Stroke color="0xCC99CC" weight="2"/>
                                    </stroke>
                                    <lineStroke>
                                            <Stroke color="0xCC99CC" weight="2"/>
                                    </lineStroke>
                                    <areaFill>
                                            <SolidColor color="0xCC99CC" alpha="0.1"/>
                                    </areaFill>
                                    <fill>
                                            <SolidColor color="0xFFFFFF"/>
                                    </fill>
                            </RadarSeries>
                    </series>
                </RadarChart>
</KoolChart>
```

<*Example 36 Example: Radar Chart*>

The following is the output produced by the layout and the data above.



<*Figure 22 Output: Radar Chart*>

6.17. Target vs Actual Charts

The Target vs Actual chart displays the target value and actual value at the same time in the chart. It has the 2D linear type and the 3D cylinder type. The definition of the 2D linear type starts with <Combination2Dchart> and ends with <Combination2Dchart/>. For the 3D cylinder type, it starts with <Combination3Dchart> and ends with <Combination3Dchart/>. Only two types of series, the series for the target and the series for the actual, can be used in the Target vs Actual chart.

The properties of the Target vs Actual chart are the same to those of the Cartesian chart. Please refer to <Table 6 Properties: Cartesian Chart>.

Keep in mind that you should define the series of the actual first and then define the series of the target.

	2D Linear Type	3D Cylinder Type	3D Cylinder-Bar Type
Chart	Combination2DChart	Combination3DChart	Combination3DChart
Series for Actual	Target2DResultSeries	Target3DResultSeries	HTarget3DResultSeries
Series for Target	Target2DGoalSeries	Target3DGoalSeries	HTarget3DResultSeries

<Table 10 Node Names of Target vs Actual Chart>

The following is an example layout for the 2D Target vs Actual chart.

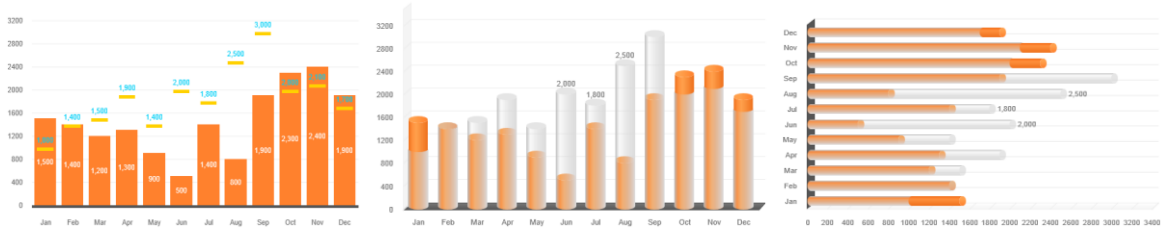
```
<KoolChart backgroundColor="0xFFFFEE"  cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Target vs Actual Chart"/>
                <SubCaption text="Linear Type"/>
        </Options>
        <Combination2DChart showDataTips="true">
                <horizontalAxis>
                        <CategoryAxis categoryField="Month" />
                </horizontalAxis>
                <series>
                        <!-- The series for the actual value should be defined first -->
                        <Target2DResultSeries yField="Result" displayName="Result">
                                <showDataEffect>
                                        <SeriesInterpolate />
                                </showDataEffect>
                        </Target2DResultSeries>
                        <!-- The target series -->
                        <Target2DGoalSeries yField="Goal" displayName="Goal">
                                <showDataEffect>
                                        <SeriesInterpolate />
```

```
                                    </showDataEffect>
                                </Target2DGoalSeries>


                            </series>
                        </Combination2DChart>
</KoolChart>
```

<Example 37 Example: 2D Target vs Actual Chart>

You can create the 3D Target vs Actual chart from the above example by changing Combination2DChart, Target2DResultSeries and Target2DResultSeries to Combination3DChart, Target3DResultSeries and Target3DResultSeries respectively.

The followings are the outputs of the Target vs Actual chart



<Figure 23 Outputs: Target vs Actual Chart>

6.18. Scroll Charts

The Scroll chart is useful when the chart has too many data to show but has a little space. Instead of accommodating all data in the constrained space, you can use the Scroll chart to increase the space and to dispay data better. It displays a specified number of items in the chart with the scrollbar.

Because <CategoryLinearAxis> is inherited from <LinearAxis>, you can use all the properties of <LinearAxis> in <CategoryLinearAxis>. To create a Scroll chart, please follow the steps below.

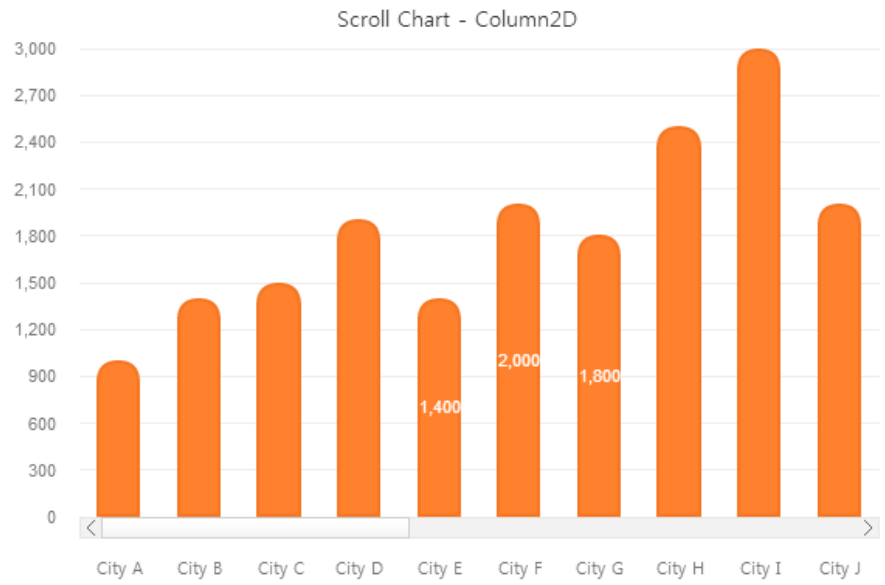
- Create the layout of any 2D chart (e.g. Column 2D chart).
- As the Column chart needs to be scrolled horizontally, set <CategoryLinearAxis> as a child of <horizontalAxis> (You can define <verticalAxis> for the Bar chart).
- Set <ScrollableAxisRenderer> as a child of <horizontalAxisRenderers>.

The scrollbar is created as a CSS style. The image files for the scrollbar are located in the Assets directory.

```
<KoolChart backgroundColor="0xFFFFEE"   cornerRadius="12" borderStyle="solid">
        <Options>
```

```
                <Caption text="Scroll Chart – Column 2D"/>
        </Options>
        <Column2DChart showDataTips="true" gutterRight="10">
            <series>
                <Column2DSeries id="cs1" yField="Data1" displayName="Data1" itemRenderer="
SemiCircleColumnItemRenderer"/>
            </series>
            <horizontalAxis>
                <CategoryLinearAxis id="hAxis" categoryField="Gu"/>
            </horizontalAxis>
            <horizontalAxisRenderers>
                <ScrollableAxisRenderer axis="{hAxis}" visibleItemSize="10" scrollSensitivity="10"/>
            </horizontalAxisRenderers>
        </Column2DChart>
</KoolChart>
```

<Example 38 Example: Scroll 2D Column Chart>



<Figure 23 Output: Scroll 2D Column Chart>

- Scroll Images for CSS

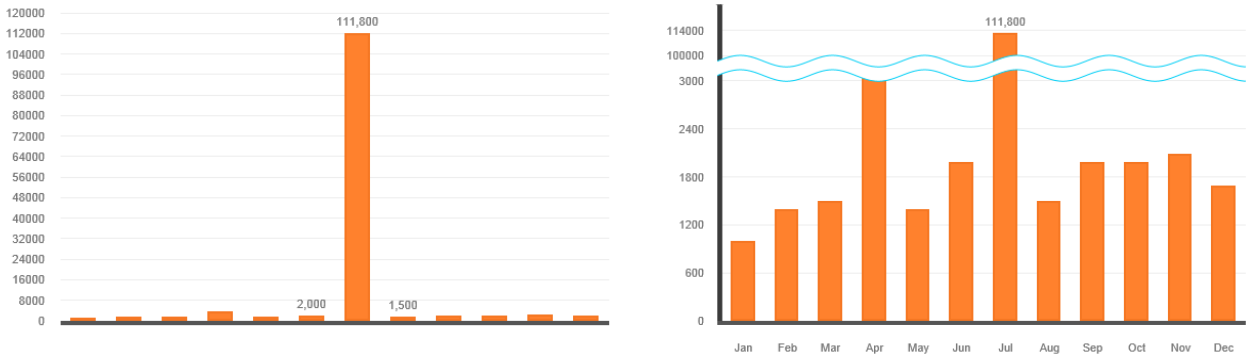
```
<style>
/* For the scrollbar DIV */
.KoolChart__ScrollBar {
        border : 1px solid #78CBDF;
}
```

```
/* For the horizontal track DIV */
.KoolChart__HScrollTrack {
	background : url('./Assets/h_scroll_track.png') repeat-x;
}

/* For the thumb DIV */
.KoolChart__HScrollThumb {
	background:url('./Assets/h_scroll_thumb.png') repeat-x;
	border : 1px solid #78CBDF;
	cursor : pointer;
}

/* For the thumb - hover DIV */
.KoolChart__HScrollThumb:hover{
	background:url('./Assets/h_scroll_thumb_hover.png') repeat-x;
}

/* For the thumb - line DIV */
.KoolChart__HScrollThumbHeader{
	background:url('./Assets/h_scroll_line.png') no-repeat;
}

/* For the right (or up) arrow DIV */
.KoolChart__HScrollUpArrow {
	background:url('./Assets/right_scroll_arrow.png') no-repeat;
	cursor : pointer;
}

/* For the left (or down) arrow DIV */
.KoolChart__HScrollDownArrow {
	background:url('./Assets/left_scroll_arrow.png') no-repeat;
	cursor : pointer;
}

/* For the vertical track DIV */
.KoolChart__VScrollTrack {
	background : url('./Assets/v_scroll_track.png') repeat-y;
```

```
}

.KoolChart__VScrollThumb {
	background:url('./Assets/v_scroll_thumb.png') repeat-y;
	border : 1px solid #78CBDF;
	cursor : pointer;
}

.KoolChart__VScrollThumb:hover{
	background:url('./Assets/v_scroll_thumb_hover.png') repeat-y;
}

.KoolChart__VScrollThumbHeader{
	background:url('./Assets/v_scroll_line.png') no-repeat;
}

.KoolChart__VScrollUpArrow {
	background:url('./Assets/up_scroll_arrow.png') no-repeat;
	cursor : pointer;
}

.KoolChart__VScrollDownArrow {
	background:url('./Assets/down_scroll_arrow.png') no-repeat;
	cursor : pointer;
}
</style>
```

6.19. Broken Axis Charts

The Broken Axis chart is often used when a few data points greatly exceed the others. If you have a dataset which is that the most data points are smaller than 3,000, and a few data points are bigger than 100,000, you may define the Y-axis which has the scale to accommodate 100,000. Although you can see all the data points in the chart, you can hardly analyze the trend of the data points which are smaller than 3,000.



Column Chart Broken Axis Chart

< Figure 24 Outputs: Using Broken Axis Chart >

As you can see the figures above, the Broken Axis chart is very useful when a few data points greatly exceed the others.

The following is an example layout for the Broken Axis chart.

```
<?xml version="1.0" encoding="utf-8"?>
<KoolChart backgroundColor="0xFFFFFF" borderStyle="solid" cornerRadius="5">
        <Options>
                <Caption text="Anual Report"/>
        </Options>
        <Column2DChart showDataTips="true">
                <horizontalAxis>
                        <CategoryAxis categoryField="Month"/>
                </horizontalAxis>
                <verticalAxis>
                <!—Defines <BrokenAxis> to create a broken axis chart. -->
                        <BrokenAxis id="vAxis" brokenMinimum="3000" brokenMaximum="110000"
brokenRatio="0.8"/>
<!-- brokenMinimum – the start point of the broken axis -->
<!-- brokenMaximum – the end point of the broken axis -->
```

```
<!-- brokenRatio – the location where the broken axis is drawn. The valid value range is 0 ~ 1 -->
<!—if the value reaches 1, the location will be closer to the brokenMaximum. -->
                </verticalAxis>
                <verticalAxisRenderers>
                        <BrokenAxis2DRenderer axis="{vAxis}"/>
                        <!-- You must define <BrokenAxis2DRenderer>, otherwise BrokenAxis cannot be
rendered. -->
                </verticalAxisRenderers>
                <series>
                        <Column2DSeries yField="Profit" displayName="Profit">
                                <showDataEffect>
                                        <SeriesInterpolate/>
                                </showDataEffect>
                        </Column2DSeries>
                </series>
                <annotationElements>
                        <CrossRangeZoomer enableZooming="false"/>
                        <!—<BrokenAxis> does not support zooming. -->
                </annotationElements>
        </Column2DChart>
</KoolChart>
```

<Example 39 Example: Broken Axis Chart>

Please notice the following:

The Broken Axis chart supports only the following charts: Line Chart, Area Chart, Column Chart, Bar Chart

The Broken Axis chart does not support <CrossRangeZoomer>, <HistoryChart>.

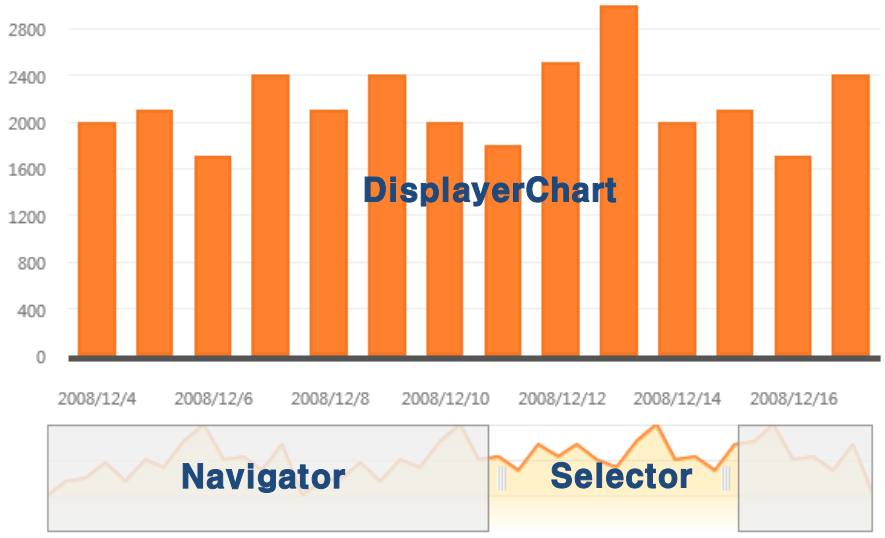
6.20. History Charts

The History chart has the same characteristics to the Scroll chart. But the difference between the two charts is that the History chart scrolls through the navigator (or overview). The navigator allows users to control the number of items displayed in the chart dynamically. The definition of the History chart starts with <HistoryChart> and ends with <HistoryChart/>. The History chart is composed of three different parts, which are displayer, navigator and selector.

- DisplayerChart
  It displays the selected items you want to display in the chart. You can select the items you want to see through Navigator.

- Navigator
  It displays the overview of the entire items.

- Selector
  It displays the selected area of Navigator. The items in the selected area are displayed in Displayer.



<Figure 25 Three Parts of History Chart>

The following table shows the properties of the History chart.

Properties	Values	Descriptions
displayerChart	Displayer	Defines <displayerChart>.
navigator	Navigator	Defines <navigator>.
selector	HistoryRangeSelector	Defines <selector>.

<Table 11 Properties: History Chart>

Displayer and Navigator are valid values for <displayerChart> and <navigator>. They are only available to <HistoryChart>. Displayer has all the properties of the Cartesian chart and Navigator has all the properties of the Area2D chart.

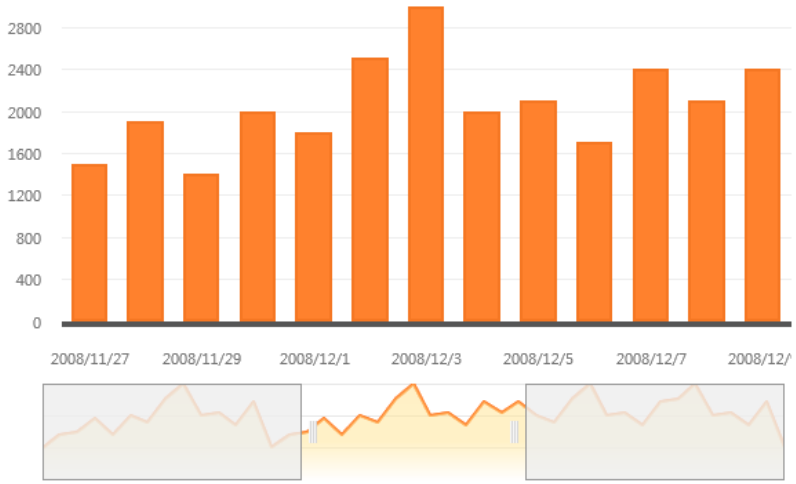
The following is a layout of the History chart.

```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
    <Options>
```

```
        <Caption text="History 2D Chart"/>
    </Options>
    <HistoryChart>
        <displayerChart> <!-- Sets displayerChart -->
            <Displayer id="chart1" showDataTips="true" width="100%" height="100%">
                <horizontalAxis>
                    <CategoryAxis id="mainHAxis" categoryField="Date"/>
                </horizontalAxis>
                <series>
                    <Column2DSeries id="columnSeries"
                        yField="Data1" fill="0xB0C759"
                        displayName="Data1">
                        <showDataEffect>
                            <SeriesInterpolate duration="1000"/>
                        </showDataEffect>
                    </Column2DSeries>
                </series>
            </Displayer>
        </displayerChart>
        <navigator>
            <Navigator id="navi"   width="100%" height="100">
                <horizontalAxis>
                    <CategoryAxis categoryField="Date" id="naviHAxis"/>
                </horizontalAxis>
                <horizontalAxisRenderers>
                    <AxisRenderer axis="{naviHAxis}" visible="false"/>
                </horizontalAxisRenderers>
                <verticalAxis>
                 <LinearAxis id="vAxis"/>
                </verticalAxis>
                <verticalAxisRenderers>
                 <AxisRenderer axis="{vAxis}" visible="false"/>
                </verticalAxisRenderers>
                    <backgroundElements>
                 <GridLines direction="horizontal"/>
                </backgroundElements>
                    <series>
                    <Area2DSeries name="A" yField="Data1"/>
                </series>
            </Navigator>
        </navigator>
        <selector><!-- Displays 30% of the center of the total data when the initial rendering is finished. -->
            <HistoryRangeSelector width="100%" startingRange="center" visibleItemSize="30"/>
        </selector>
```

```
  </HistoryChart>
</KoolChart>
```

<*Example 39 Example: History Chart*>



<*Figure 26 Output: History Chart*>

- The thumb of the navigator is created as a CSS style.

```
<style>
/* For the thumb of the navigator */
.KoolChart__ChartRangeSelector_Thumb {
        background:url('./Assets/selector.png') no-repeat;
}
</style>
```

6.21. From-To Charts

The From-To chart displays the data area by using the start point and the end point. The Waterfall chart and the Step chart can be created from the From-To chart. The column series (Column2DSeries, Column3DSeries), bar series (Bar2DSeries, Bar3DSeries) and area series (Area2DSeries) are used as the series of the From-To chart. The *minField* property is added for the start point and all the other properties of the From-To chart are the same to those of the Column, Bar and Area chart.

Now we will create a Waterfill chart by using a Column chart. The following is the array form of the numeric data.

```
var data= [
            {"cat":"Jan", "from":1000, "to":5000},
           {"cat":"Feb", "from":5000, "to":12000},
           {"cat":"Mar", "from":12000, "to":16000},
           {"cat":"Apr","from":16000, "to":10000},
            {"cat":"May", "from":10000, "to":100},
            {"cat":"Jun", "from":100, "to":-1000},
            {"cat":"Jul", "from":-1000, "to":3000},
            {"cat":"Aug", "from":3000, "to":8000},
            {"cat":"Sep", "from":8000, "to":12000},
            {"cat":"Oct", "from":12000, "to":14000},
            {"cat":"Nov", "from":14000, "to":7500},
            {"cat":"Dec", "from":7500, "to":2500},
            {"cat":"Next Year", "from":0, "to":20000}
            ];
```

<Example 40 Example: Data of From-To Chart>

The above data represents the monthly revenue variance, and the following is the layout to create the Waterfill chart using the above data.

```
<KoolChart backgroundColor='0xFFFFEE'  cornerRadius='12' borderStyle='solid'>
        <Options>
                <Caption text='2013 Revenue Variation (Monthly)'/>
                <SubCaption text='Unit: Thousand Dollars' textAlign='right' fontSize='11' paddingRight='20'/>
        </Options>
        <NumberFormatter id='fmt'/>
        <Column2DChart showDataTips='true' dataTipJsFunction='dataTipFunc' fontSize='11'>
                <horizontalAxis>
                        <CategoryAxis categoryField='cat'/>
                </horizontalAxis>
                <verticalAxis>
                        <LinearAxis formatter='{fmt}'/>
                </verticalAxis>
                <series>
                        <Column2DSeries id='series1' minField='from' yField='to' fillJsFunction='fillFunc'
itemRenderer='BoxItemRenderer'>
                                <showDataEffect>
                                        <SeriesSlide direction='up' duration='1000'/>
                                </showDataEffect>
                                <stroke>
```

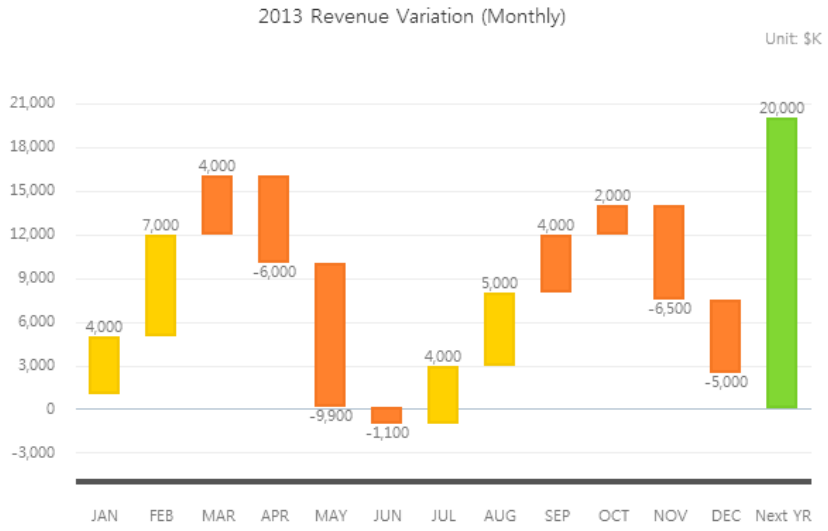
```
                                        <Stroke weight='1' color='0x999999' alpha='0.7'/>
                                    </stroke>
                                </Column2DSeries>
                            </series>
                        </Column2DChart>
</KoolChart>
```

<*Example 41 Example: Layout of From-To Chart*>

As shown in the layout above, the *minField* property with the value of "from" is added in <Column2DSeries>.

The fillJsFunction and dataTipJsFunction are used to create the user-defined tooltips and colors. Please refer to < 7.9 Setting User-Defined Functions> for further information.

The output of the above layout is as follows:



<*Figure 27 Output: From-To Chart*>

6.22. Matrix Charts

The Matrix chart represents data using images or figures of which position is determined by the X, Y coordinates, and the size is determined by the Z value. The types of the Matrix chart are as follows:

Types of the Matrix chart	Descriptions	Outputs
renderer	The position is determined by the X, Y coordinates. The size of the item is determined by the Z value.	
image	The position is determined by the X, Y coordinates. The size of the image is determined by the Z value.	
fill	The position is determined by the X, Y coordinates. The color of the area is determined by the Z value.	
plot	The position is determined by the X, Y coordinates. The type of the plot is determined by the Z value	

<Table 12 Types of Matrix2D Chart>

The following example shows how to use the *type* property of the Matrix chart.

```
<Matrix2DChart showDataTips="true" type="renderer" selectionMode="single">
```

The following table shows the properties of the Matrix2D chart.

Properties	Values	Descriptions
type	renderer, image, fill, plot (default : renderer)	Defines the data representation of the Matrix2D chart.
drawType	radius, area (default : radius)	Specifies the size criteria of the item. radius: size = 2*πr, area: size = area.

		(If the value of the type is fill (or plot), the value of drawType will be ignored.)

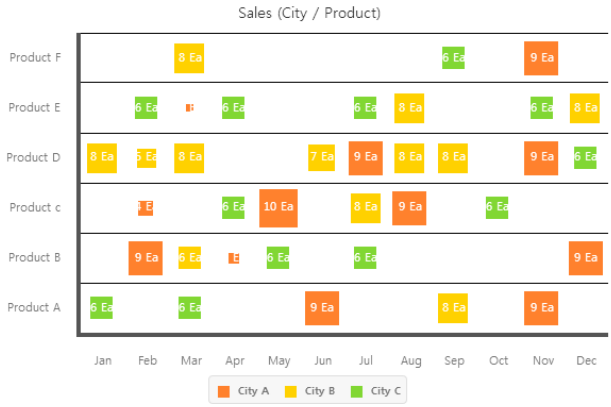
<Table 13 Properties: Matrix2D Chart>

The following example describes how to use the *drawType* property when the type is renderer (or image) in the Matrix2D chart.

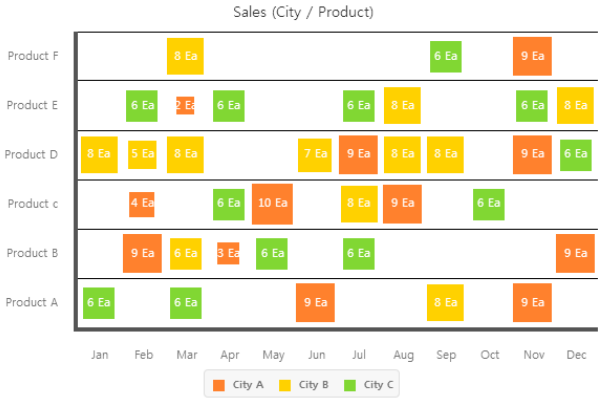
```
<Matrix2DChart showDataTips="true" type="renderer" drawType="radius" selectionMode="single">
```

The followings show the outputs of the Matrix2D chart.

drawType = "radius"



drawType = "area"



If you want to change the shape of the item, you need to set the *renderer* property as follows:

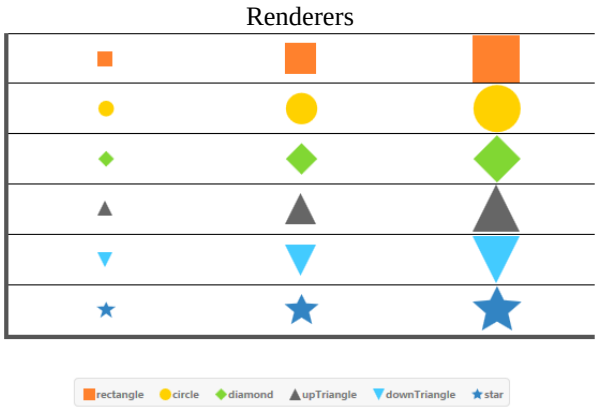
```
<Matrix2DSeries ….. renderer="circle" // renderer="star" …..>
```

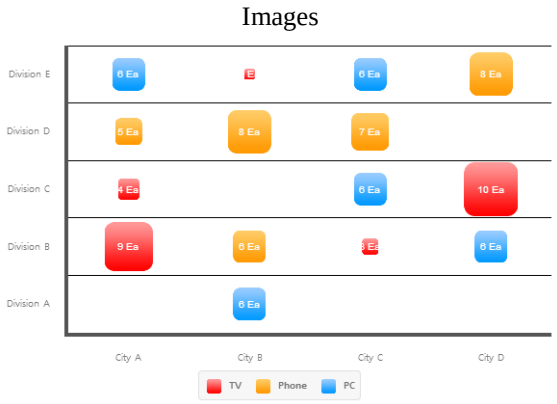
If you want to display images for the items, you need to set the *imageSource* property as follows:

The value of the *imageSource* property should be the URL of the image file.

```
<Matrix2DSeries ….. imageSource="./Images/icon1.png" …..>
```

The followings show the outputs of the above examples.



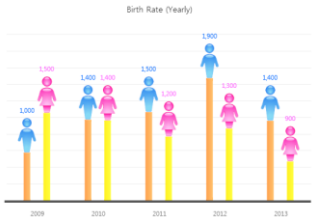
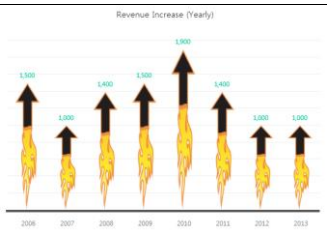


6.23. Image Charts

The Image chart is a kind of the Column chart. It represents data as images whose size is determined by the value of the numeric data. The following factors determine the types of the Image chart:

- First, the aspect ratio of the original image
- Second, whether to use a single image or multiple images to represent the data items

The following table describes the types of the Image chart.

Types (imageDisplayType)	Aspect Ratio (true,false)	Descriptions	Samples
single	True	Represents data by using a single image. Uses the same aspect ratio for items. If the extra space remains unused, the space will be filled with a bar.	
	False	Represents data by using a single image. The image size is adjusted automatically (mostly vertical).	

singleRepeat	True	Represents data by using a single image. Uses the same aspect ratio for items. If the extra space remains unused, the space will be filled with the same images.	
	False	n/a	X
multiple	True	n/a	X
	False	Represents data by using multiple images. Each image has a unique value. The size of an image is the unit value.	

<Table 14 Types of Image Chart>

The *imgSource* property is added for the Image chart and all the other properties of the Image chart are the same to those of the Column chart.

The *imgSource* property determines the path of the image file and the way to represent the numeric data.

```
<ImageSeries yField="Data1" imageDisplayType="singleRepeat" displayName="Number" styleName="seriesStyle"
formatter="{numFmt}">
    <imgSource>
        <ImageSourceItem url="../Samples/Images/10000.png"/>
    </imgSource>
        <showDataEffect>
            <SeriesSlide duration="1000" direction="up"/>
        </showDataEffect>
</ImageSeries>
```

In the above example, a single image is repeated because the value of imageDisplayType is singleRepeat. The path (URL) of the image file is defined in <ImageSourceItem>. It is a child node of <ImgSource>. The properties of <ImageSourceItem> are as follows:

Properties	Values	Descriptions
maintainAspectRatio	True | false (default: true)	Whether or not the aspect ratio is the same

		to the original image.
url	The URL of images	The path (URL) of the image file.
Value	Number	Specifies the unit value of each image. (only available to multiple)

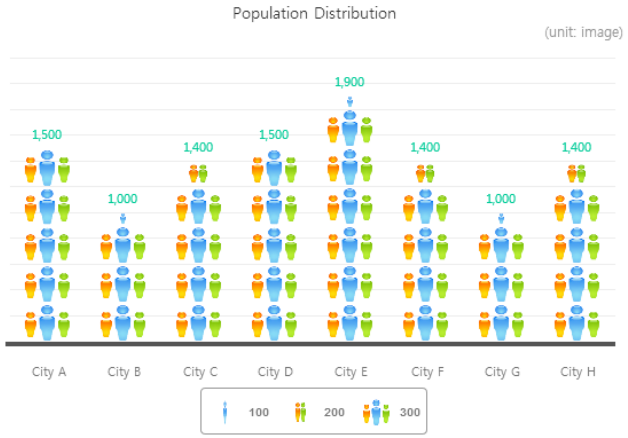
<Table 15 Properties: The properties of <ImageSourceItem>>

The following is an example layout for the Image chart whose type is multiple, and each image has a unique value.

```
<ImageChart id="chart" showDataTips="true" gutterLeft="20" gutterRight="20" showLabelVertically="true">
                    <!--defines the X axis -->
                      <horizontalAxis>
                              <CategoryAxis id="hAxis" categoryField="Region"/>
                      </horizontalAxis>
                    <!--defines the Y axis -->
                      <verticalAxis>
                              <LinearAxis id="vAxis"/>
                      </verticalAxis>
                      <series>
                     <!--Defines the series -->
                      <ImageSeries yField="Data1" imageDisplayType="multiple" styleName="seriesStyle"
formatter="{numFmt}">
                          <imgSource>

                          <!-- The unit value of the first image is 100 -->
                            <ImageSourceItem maintainAspectRatio="false" url="../Samples/Images/3-1.png"
value="100"/>

                          <!--The unit value of the second image is 200 -->
                            <ImageSourceItem maintainAspectRatio="false" url="../Samples/Images/3-2.png"
value="200"/>

                          <!--The unit value of the third image is 300 -->
                            <ImageSourceItem maintainAspectRatio="false" url="../Samples/Images/3-3.png"
value="300"/>

                          </imgSource>
                      </ImageSeries>
                      </series>
                      <horizontalAxisRenderers>
                              <AxisRenderer axis="{hAxis}" fontSize="11"/>
                      </horizontalAxisRenderers>
                <-- visible="fasle" for the Y axis -->
                      <verticalAxisRenderers>
```

```
                    <AxisRenderer axis="{vAxis}" visible="false" includeInLayout="false"/>
                </verticalAxisRenderers>
            </ImageChart>
```

The output of the above layout is as follows:

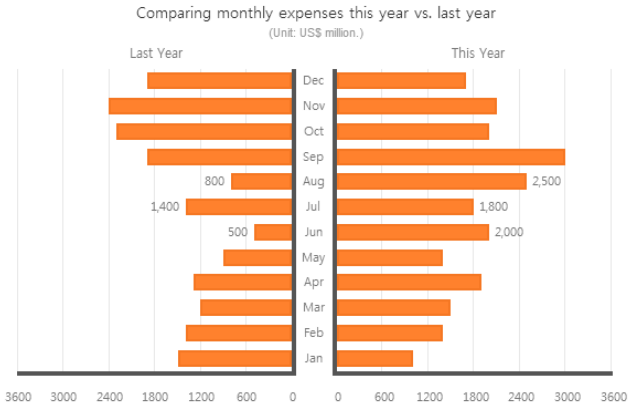


<*Figure 28 Output: Image Chart*>

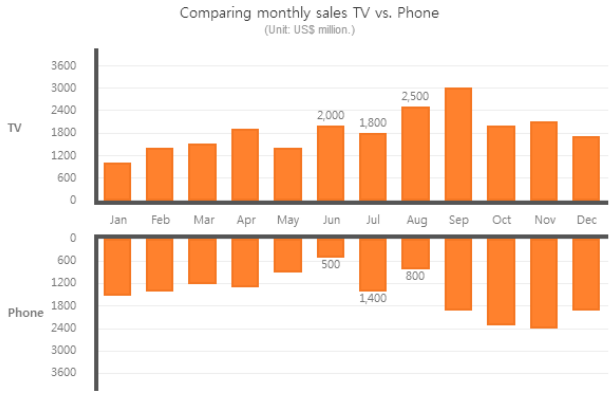
6.24. Wing Charts

The Wing chart (or Tornado chart) is a special type of the Bar chart (or Column chart), where two bars (or columns) are spread over both sides of the axis. The Wing chart is useful to compare two categories of data and to provide decision makers with a quick overview.

KoolChart supports two types of Wing chart, which are the Bar type and the Column type.



Bar Type



Column Type

<Figure 29 Output: Two Types of Wing Chart>

The following is an example layout for the Wing chart.

```
<?xml version="1.0" encoding="utf-8"?>
<KoolChart backgroundColor="0xFFFFFF" borderStyle="solid" cornerRadius="5">
        <Options>
                <Caption text="Anual Report"/>
        </Options>
        <!—Defines <Column2DWingChart> to create a Wing chart. -->
        <Column2DWingChart showDataTips="true">
                <horizontalAxis>
                        <CategoryAxis categoryField="Month"/>
                </horizontalAxis>
                <series>
                <!-- <Column2DWingSeries> must be defined for the series of <Column2DWingChart>. -->
                <!— Column2D (or 3D) chart has only one data field, yField. -->
                <!—But in Wing chart, an additional field, yFieldOpp, must be defined in the layout. -->
                <!—yField: draw to the up, yFieldOpp: drawn to the bottom -->
                        <Column2DWingSeries yField="Profit" yFieldOpp="Cost" labelPosition="inside">
                                <showDataEffect>
                                        <WingSeriesInterpolate/>
                                </showDataEffect>
                        </Column2DWingSeries>
                </series>
        </Column2DWingChart>
</KoolChart>
```

6.25. Real-Time Premium Charts

The Real-time premium chart is the advanced version of the Real-time chart, <6.15 Real-Time Charts>. Like the Real-time chart, it uses HTTPService and RPC for data communication. The unique features of the Real-time premium chart are as follows:

1. You can use different data refresh cycles for each series. For example, if you have 3 series, you can set 10 seconds for the first and 5 seconds for the second and 1 minute for the third.
2. The Real-time chart displays data from the right to the left based on the unit (time or amount of data), but the Real-time premium chart displays data from the left to the right based on the time defined by servers. The cycle of each series of the Real-time premium chart can be different because each series can have the different time unit.
3. If the initial data of the Real-time chart need to be displayed in the client browser, they will be displayed as soon as the chart-loading is finished.

4. <DateTimeAxis> must be defined for the X-axis in the Real-time premium chart. The Real-time premium chart does not support <CategoryAxis>.

Unlike the Real-time chart, the definition of the Real-time premium chart does not start with <RealTimeChart>. You can create the Real-time premium chart using any type of KoolChart by defining <HttpMultiServiceRepeater> in the layout. <HttpMultiServiceRepeater> has the properties of RPC for each series.

The following table describes the properties of <HttpMultiServiceRepeater>.

Properties	Values	Descriptions
baseURL	URL	Specifies the base URL for the *url* property of RPCItem. (full path = baseURL + url of RPCItem)
method	get | post (default: get)	Specifies the HTTP method.
requestTimeout	Second	The maximum waiting time for the request.
targetController	Chart ID	Specifies the controller of the target of RPCItem. (It must be the chart.)
showErrorMessage	true|false (default: true)	Whether or not to show the error message box when RPC error occurrs.

<Table 16 Properties: HttpMultiServiceRepeater Node>

The following table describes the properties of <RPCItem>.

Properties	Values	Descriptions
name	String	Specifies the name of RPCItem.
url	String	Specifies the url of RPCItem. (full path = baseURL + url of RPCItem)
method	get | post (default: get)	Specifies the HTTP method.
target	Series ID	Specifies the ID of the series received from the RPC request. Each RPCItem is a series.
interval	Seconds	Specifies the interval of the RPC requests. The first RPC request is sent when the initial chart loading is finished. If the value of interval is not specified, the request will not be sent periodically.
concurrency	multiple | single | last	Specifies how to handle the duplicated HTTP

	(default: multiple)	requests. multiple: Sends all requests without cancelling the previous requests. single: Sends one request at a time. If the duplicated requests occur, the error message will be generated. last: Cancels all the previous requests and sends the last request
retryCount	Number (default: 30)	Specifies the retry count, when RPC error occurs.

<Table 17 Properties: RPCItem Node>

The following example shows how to create a Real-time premium chart by using a Combination 2D chart.

```
<KoolChart backgroundColor="0xFFFFFF"  cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Real-time chart - different cycle data"/>
                <SubCaption text="Random data is used" textAlign="right" fontSize="11"/>
                <Legend fontSize="11" useVisibleCheck="true"/>
        </Options>

        // Defines the formatters
        <DateFormatter id="dateOrgFmt" formatString="YYYY/MM/DD HH:NN:SS"/>
        <DateFormatter id="dateFmt" formatString="HH:NN:SS"/>
        <DateFormatter id="dateFmt2" formatString="HH:NN"/>
        <NumberFormatter id="numFmt"/>

// Creates the Combination 2D chart with one column series and two line series
        <Combination2DChart id="chart" showDataTips="true" dataTipMode="multiple">
                <series>

                        <!-- Line series with 5 seconds cycle  -->
                        <Line2DSeries id="lineSeries" xField="date" yField="data5"  displayName="Data(5
Sec)">
                                <lineStroke>
                                        <Stroke color="0xFF6666" weight="2" alpha="1"/>
                                </lineStroke>
                                <horizontalAxis>
                                        <DateTimeAxis id="hAxis2" displayLocalTime="true"
labelUnits="minutes" dataUnits="seconds" interval="1" formatter="{dateOrgFmt}" displayName="Time"/>
                                </horizontalAxis>
                                <verticalAxis>
                                        <LinearAxis id="vAxis2" minimum="0" maximum="150"/>
```

```
                                    </verticalAxis>
                                </Line2DSeries>

                                <!-- Line series with 3 seconds cycle -->
                                <Line2DSeries id="lineSeries2" xField="date" yField="data3" displayName="Data(3 Sec)"
verticalAxis="{vAxis2}" horizontalAxis="{hAxis2}">
                                    <lineStroke>
                                        <Stroke color="0x339966" weight="1" alpha="1"/>
                                    </lineStroke>
                                </Line2DSeries>

                                <!-- Defines the accumulated sum-->
                                <Column2DSeries id="columnSeries" xField="date" yField="data60"
displayName="Accumulated sum" itemRenderer="BoxItemRenderer">
                                    <horizontalAxis>
                                        <DateTimeAxis id="hAxis" displayLocalTime="true"
labelUnits="hours" dataUnits="minutes" interval="3" dataInterval="10" formatter="{dateOrgFmt}"
displayName="Time"/>
                                    </horizontalAxis>
                                    <verticalAxis>
                                        <LinearAxis id="vAxis" minimum="0" maximum="999"/>
                                    </verticalAxis>
                                    <fill>
                                        <SolidColor color="0x6666FF"/>
                                    </fill>
                                </Column2DSeries>

                            </series>

                            <horizontalAxisRenderers>
                                <AxisRenderer axis="{hAxis}" placement="bottom" formatter="{dateFmt2}"
tickLength="30" minorTickLength="0" tickPlacement="inside" showLine="false">
                                    <axisStroke>
                                        <Stroke weight="1" color="0x999999"/>
                                    </axisStroke>
                                    <tickStroke>
                                        <Stroke weight="1" color="0x6666FF" alpha="0.5"/>
                                    </tickStroke>
                                </AxisRenderer>
                                <AxisRenderer axis="{hAxis2}" placement="bottom" formatter="{dateFmt}">
                                    <axisStroke>
                                        <Stroke weight="1" color="0x999999"/>
                                    </axisStroke>
                                </AxisRenderer>
```

```
                </horizontalAxisRenderers>

                <verticalAxisRenderers>
                        <AxisRenderer axis="{vAxis}" placement="right" formatter="{numFmt}"/>
                        <AxisRenderer axis="{vAxis2}" placement="left" formatter="{numFmt}"/>
                </verticalAxisRenderers>
        </Combination2DChart>


// Defines HttpMultiServiceRepeater for a Real-time premium chart.
        <HttpMultiServiceRepeater baseURL="http://www.koolchart.com/realtimeSample/" targetController="{chart}"
requestTimeout="30">
                <RPCList>
                        <RPCItem name="rpc1" url="data3Interval.php" target="{lineSeries2}" interval="3"
concurrency="last" retryCount="30"/>
                        <RPCItem name="rpc2" url="data5Interval.php" target="{lineSeries}" interval="5"
concurrency="last" retryCount="30"/>
                        <RPCItem name="rpc3" url="data23ToCurrent2.php" target="{columnSeries}"
interval="600" concurrency="last" retryCount="30"/>
                </RPCList>
        </HttpMultiServiceRepeater>
</KoolChart>
```

<Example 42 Example: Real-time Premium Chart>

The above sample layout generates a Real-time premium chart which has two line series and one column series and their intervals are 3 seconds, 5 seconds and 10 minutes repectively. The RPC request scenario is that the first RPC request is sent after the initial chart-loading is finished and then the next RPC request is sent at the interval specified for each series. In this example, two line series have no initial data and the column series has the initial data. It displays data after the initial loading for 10 minutes then it refreshes the chart with the new data based on the interval of each series for the next 10 minutes. The accumulated sum for 10 minutes is displayed as a column.

You can see the examples of the server-side scripts for each case. Please see the following folder of the installation CD.

root of CD(or ZIP file)/Samples/RealtimeServerSamples/

The following is the descriptions of the server-side scripts for the above example.

The server-side example of the Real-time premium chart.

http://demo.koolchart.com/realtimeSample/hourDataToday.php

The descriptions of how it works

// The data in this example is based on arbitrary random data.
requestAllData = true, generates data every 5 seconds from "hour:00" until "hour:minute".
requestAllData = false, generates only one data corresponing to the time "hour:minute".
As the value of the *interval* property of RPCItem is 5, the chart receives data from the server every 5 seconds and adds the newly received data to the existing data.
At the time when <nextInitDate>, all data will be deleted and requested again. (refresh)

Copy the following URLs into your browser and try to print out.

http:demo.koolchart.com/realtimeSample/hourDataToday.php?requestAllData=true

http:demo. koolchart.com /realtimeSample/hourDataToday.php?requestAllData=false

---

The descriptions of the parameters.

---

requestAllData: Whether or not the chart requests all data.
This parameter is used to determine the data-refresh time.
If the value of requestAllData is true, RPC occurrs when the following conditions are met.

1. When the initial loading is finished, the first RPC request occurrs.
2. At the time when <nextInitDate>.

If "requestAllData=true", you should set the value of the *isInitData* property in the infoMsg node to true at the server side. (<isInitData>true</isInitData>).
If "requestAllData=false", <isInitData>false</isInitData>

index: Indicates the index of the recently received data.
By using the index, you can minimise the duplication of data.
The client chart maintains the index of the last data received from the server. By passing the index as a parameter when the next data request, the server can easily find the next data to send.

dummy: This value is for avoding the cache problem of IE.
It indicates the milliseconds time from 00:00, January 1st, 1970 until the current time. (This time is based

on the client time).

XML writing rules.

The following rules must be followed to write the XML sent to the client.

Rule 1. The infoMsg node and its sub-node information must be required.
Rule 2. The data should be enclosed in the item node.

The descriptions of the sample XML.

The data in this example is based on arbitrary random data (not the data from database)

The display results of the XML are different based on the value of requestAllData.

1. requestAllData = false,
   Generates the XML for one data item.
   The newly received data is added to the existing data.

2. requestAllData = true,
   on the initial loading of the chart, The client requests the data from the server with the parameter, "requestAllData=true"
   In this sample, the server will send the cumulative data from the current time, "hour:00" until the time, "hour:minute".
   At the time when <nextInitDate>, the client sends the request again with the parameter, "requestAllData = true".

3. The descriptions of the infoMsg node.

   1) index node
      The index parameter is not used in this example. But this parameter is useful by passing the unique ID of the most recently received index.
      The server can easily find the next data to send by using this parameter.

   2) timeNow node
      It indicates the current time of the server. This parameter is used to determine the data-refresh time.
      If you omit this parameter, the data-refresh time will be determined by comparing the client time with endDate.
      In other words, if you do not set the timeNow node, the refresh time of each client could be different.

3) isInitData node

If you have the request with the parameter, "requestAllData = true", you set the value of the isInitData node to true.

4) startDate node

The initial time displayed in the chart.

If you omit this parameter, the chart will display the initial time based on the time of the initial data-loading.

5) endDate node

The last time displayed in the chart.

6) nextInitDate node

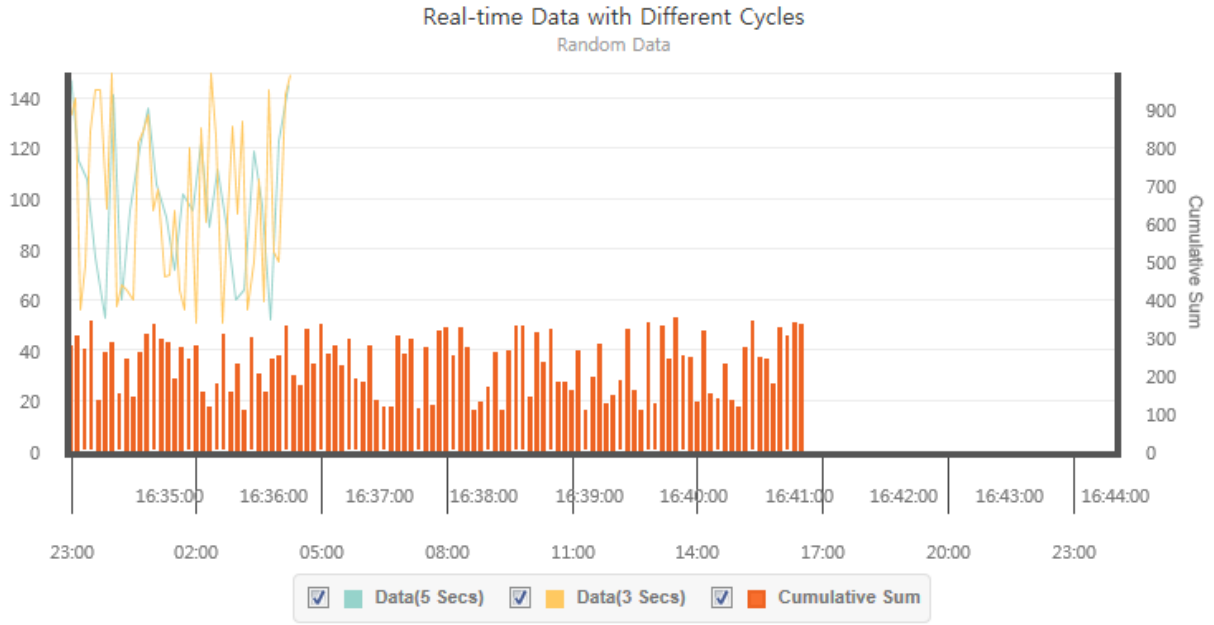
At the time when <nextInitDate>, the data is refreshed. The request is sent with the parameter, "requestAllData=true".

This value is related with the current time and the timeNow node.

For example, If the chart displays the data created one month ago every 10 minutes and it has finished displaying the endDate data at a certain time, the data request will be sent based on this time with the parameter, "requestAllData=true"

<Example 43 Example: Server-Side of Real-Time Premium Chart>

The following is the output produced by the layout above.



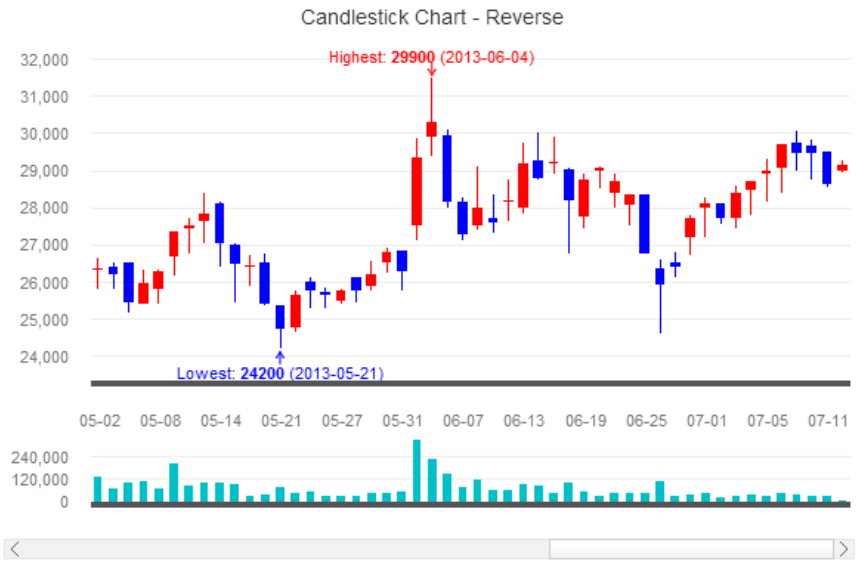
<Figure 30 Output: Real-time Premium Chart>

6.26. Candlestick Charts

The Candlestick chart in KoolChart is a subset of professional stock charts. You can define a dataset that contains open, high, low and close values for each time period you want to display. But functionalities supported by the professional stock chart such as drawing a line or adding a specific symbol at a specific point are not supported by the Candlestick chart in KoolChart.

The functionalities supported by the Candlestick chart in KoolChart are as follows:

- Showing the maximum and minimum values in the chart.
- Showing the maximum and minimum values using the user-defined function.
- Showing the disclosure symbol.
- The disclosure symbol can be represented by <itemRenderer>, and special characters or images also can be added by using the element.
- The disclosure symbol represented by the element can get the mouse event (click).
- The line color, the outer line color of the bar and the background color of the bar can be set by the open and close values.
- The number of items shown in the chart can be customized.



<Figure 31 Output: Basic Candlestick Chart>

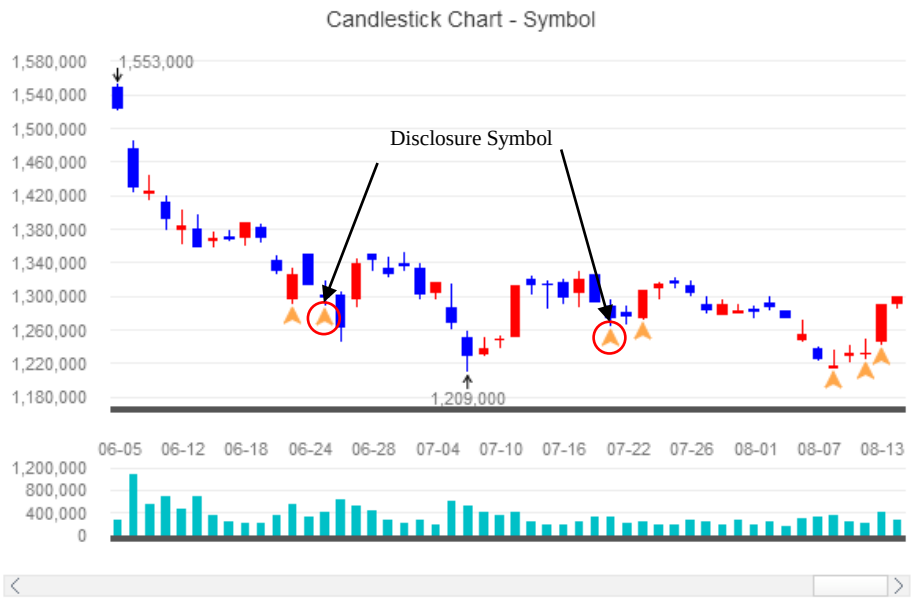
```
<KoolChart backgroundColor="0xFFFFFF"  cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text=" Candlestick Chart Reverse"/>
        </Options>
        // Defines the formatter
        <NumberFormatter id="nft" precision="0"/>
```

```
        // Sets DualChart
        <DualChart leftGutterSyncEnable="true" rightGutterSyncEnable="true">
        // leftGutterSyncEnalbe - Adjusts the left margin of mainChart and subChart
        // rightGuttersSyncEnable - Adjusts the right margin of mainChart and subChart
                <mainChart>
                        <Candlestick2DChart showDataTips="true" paddingBottom="0">
                                <horizontalAxis>
                                        <CategoryAxis id="hAxis" categoryField="date"/>
                                </horizontalAxis>
                                <verticalAxis>
                                        <LinearAxis baseAtZero="false"/>
                                </ verticalAxis>
                                <series>
// openField - Sets the field name for the opening price.
// showMaxValueLabel - Shows the maximum value of the data points currently displayed in the chart.
// showMinValueLabel - Shows the minimum value of the data points currently displayed in the chart.
                                        <Candlestick2DSeries openField="openprc" closeField="closeprc"
highField="high"   lowField="low" showMinValueLabel="true" showMaxValueLabel="true"
maxLabelJsFunction="maxLabelFunc" minLabelJsFunction="minLabelFunc">
                        // Sets the fill color when the closing price is higher than the opening price.
                                                <fill>
                                                        <SolidColor color="#ff0000" alpha="0.5"/>
                                                </fill>
                        // Sets the color of the line connecting the highest price and the lowest price when the closing
price is higher than the opening price.
                                                <stroke>
                                                        <Stroke color="#ff0000"/>
                                                </stroke>
                        // Sets the color of the outer line of the bar when the closing price is higher than the opening
price.
                                                <boxStroke>
                                                        <Stroke color="#ff0000"/>
                                                </boxStroke>
                        // Sets the fill color for the bar when the opening price is higher than the closing price.
                                                <declineFill>
                                                        <SolidColor color="#0000ff" alpha="0.5"/>
                                                </declineFill>
                        // Sets the color of the line connecting the highest price and the lowest price when the opening
price is higher than the closing price.
                                                <declineStroke>
                                                        <Stroke color="#0000ff"/>
                                                </declineStroke>
                        // Sets the color of the outer line of the bar when the opening price is higher than the closing
price.
```

```
                                                <declineBoxStroke>
                                                        <Stroke color="#0000ff"/>
                                                </declineBoxStroke>
                                        </Candlestick2DSeries>
                                </series/>
                                <horizontalAxisRenderers>
                                        <Axis2DRenderer placement="bottom" axis="{hAxis}"
tickLength="0"/>
                                </horizontalAxisRenderers>
                                <annotationElements>
// syncCrossRangeZoomer - Draws crosshairs in the sub-chart by referencing the id of CrossRangeZoomer and using
the data in the main chart.
                                        <CrossRangeZoomer id="candleCRZ" enableZooming="false"
syncCrossRangeZoomer="{columnCRZ}" zoomType="both" horizontalLabelFormatter="{nft}"/>
                                </annotationElements>
                        </Candlestick2DChart>
                </mainChart>
                <subChart>
                        <Column2DChart showDatatips="true" height="20%" paddingTop="0"
paddingBottom="0" gutterTop="4">
                                <horizontalAxis>
                                        <CategoryAxis id="hAxis2" categoryField="date"/>
                                </horizontalAxis>
                                <verticalAxis>
                                        <LinearAxis baseAtZero="false"/>
                                </ verticalAxis>
                                <series>
                                        <Column2DSeries yField="trdvolume"
itemRenderer="BoxItemRenderer">
                                                <fill>
                                                        <SolidColor color="#eca614"/>
                                                </fill>
                                        </Column2DSeries>
                                </series>
                                <horizontalAxisRenderers>
                                        <Axis2DRenderer axis="{hAxis2}" showLabels="false"
tickLength="0"/>
                                </horizontalAxisRenderers>
                                <annotationElements>
// syncCrossRangeZoomer - Draws crosshairs in the main chart by referencing the id of CrossRangeZoomer and using
the data in the sub-chart.
                                        <CrossRangeZoomer id="columnCRZ" enableZooming="false"
syncCrossRangeZoomer="{candleCRZ}" horizontalLabelFormatter="{nft}" verticalLabelPlacement="top"/>
                                </annotationElements>
```

```
                    </Column2DChart>
                </subChart>
                <dataSelector>
                    <DualScrollBar inverted="true" visibleItemSize="50"/>
                </dataSelector>
            </DualChart>
</KoolChart>
```

<Example 44 Example: Basic Candlestick Chart>



<Figure 30 Output: Candlestick Chart with Disclosure>

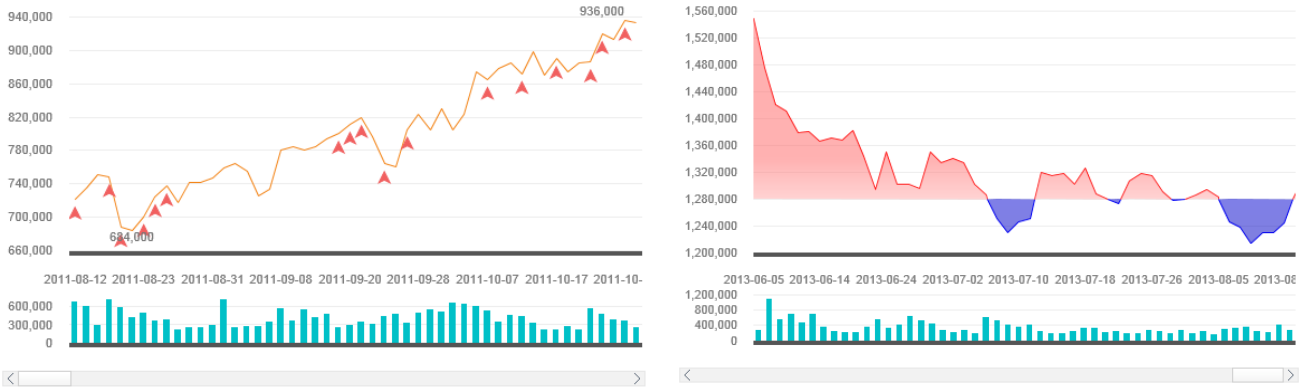
```
<KoolChart backgroundColor="0xFFFFFF"  cornerRadius="12" borderStyle="solid">
        <Options>
            <Caption text="Riamore CandleChart"/>
        </Options>
        // Defines the formatter
        <NumberFormatter id="nft" precision="0"/>
        // Sets DualChart
        <DualChart leftGutterSyncEnable="true" rightGutterSyncEnable="true">
        // leftGutterSyncEnalbe - Adjusts the left margin of mainChart and subChart
        // rightGuttersSyncEnable - Adjusts the right margin of mainChart and subChart
            <mainChart>
                <Candlestick2DChart showDataTips="true" paddingBottom="0">
                    <horizontalAxis>
                        <CategoryAxis id="hAxis" categoryField="date"/>
                    </horizontalAxis>
                    <verticalAxis>
                        <LinearAxis baseAtZero="false"/>
```

```
                                        </ verticalAxis>
                                        <series>
// openField - Sets the field name for the opening price.
// showMaxValueLabel - Shows the maximum value of the data points currently displayed in the chart.
// showMinValueLabel - Shows the minimum value of the data points currently displayed in the chart.
// symbolField - Sets the field name for the disclosure.
// symbolRenderer - Sets the disclosure symbol.
                                                <Candlestick2DSeries openField="openprc" closeField="closeprc"
highField="high"   lowField="low" showMinValueLabel="true" showMaxValueLabel="true"
maxLabelJsFunction="maxLabelFunc" minLabelJsFunction="minLabelFunc" symbolField="disclosure"
symbolRenderer="UpArrowItemRenderer">
                        // Sets the fill color when the closing price is higher than the opening price.
                                                        <fill>
                                                                <SolidColor color="#ff0000" alpha="0.5"/>
                                                        </fill>
                        // Sets the color of the line connecting the highest price and the lowest price when the closing
price is higher than the opening price.
                                                        <stroke>
                                                                <Stroke color="#ff0000"/>
                                                        </stroke>
                        // Sets the color of the outer line of the bar when the closing price is higher than the opening
price.
                                                        <boxStroke>
                                                                <Stroke color="#ff0000"/>
                                                        </boxStroke>
                        // Sets the fill color for the bar when the opening price is higher than the closing price.
                                                        <declineFill>
                                                                <SolidColor color="#0000ff" alpha="0.5"/>
                                                        </declineFill>
                        // Sets the color of the line connecting the highest price and the lowest price when the opening
price is higher than the closing price.
                                                        <declineStroke>
                                                                <Stroke color="#0000ff"/>
                                                        </declineStroke>
                        // Sets the color of the outer line of the bar when the opening price is higher than the closing
price.
                                                        <declineBoxStroke>
                                                                <Stroke color="#0000ff"/>
                                                        </declineBoxStroke>
                                                </Candlestick2DSeries>
                                        </series/>
                                        <horizontalAxisRenderers>
                                                <Axis2DRenderer placement="bottom" axis="{hAxis}"
tickLength="0"/>
```

```
                        </horizontalAxisRenderers>
                        <annotationElements>
// syncCrossRangeZoomer - Draws crosshairs in the sub-chart by referencing the id of CrossRangeZoomer and using the data in the main chart.
                            <CrossRangeZoomer id="candleCRZ" enableZooming="false" syncCrossRangeZoomer="{columnCRZ}" zoomType="both" horizontalLabelFormatter="{nft}"/>
                        </annotationElements>
                    </Candlestick2DChart>
            </mainChart>
            <subChart>
                    <Column2DChart showDatatips="true" height="20%" paddingTop="0" paddingBottom="0" gutterTop="4">
                        <horizontalAxis>
                            <CategoryAxis id="hAxis2" categoryField="date"/>
                        </horizontalAxis>
                        <verticalAxis>
                            <LinearAxis baseAtZero="false"/>
                        </ verticalAxis>
                        <series>
                            <Column2DSeries yField="trdvolume" itemRenderer="BoxItemRenderer">
                                <fill>
                                    <SolidColor color="#eca614"/>
                                </fill>
                            </Column2DSeries>
                        </series>
                        <horizontalAxisRenderers>
                            <Axis2DRenderer axis="{hAxis2}" showLabels="false" tickLength="0"/>
                        </horizontalAxisRenderers>
                        <annotationElements>
// syncCrossRangeZoomer - Draws crosshairs in the main chart by referencing the id of CrossRangeZoomer and using the data in the sub-chart.
                            <CrossRangeZoomer id="columnCRZ" enableZooming="false" syncCrossRangeZoomer="{candleCRZ}" horizontalLabelFormatter="{nft}" verticalLabelPlacement="top"/>
                        </annotationElements>
                    </Column2DChart>
            </subChart>
            <dataSelector>
                    <DualScrollBar inverted="true" visibleItemSize="50"/>
            </dataSelector>
            </DualChart>
</KoolChart>
```

<Example 45 Example: Candlestick Chart with Disclosure>

In addition to the basic type of the Candlestick chart, the Candleline and Candlearea charts can be created. To create Candleline and Candlearea chart, you need to define <CandleLine2D> and <CandleArea2D> instead of <Candlestick2D>. The outputs of the Candleline and Candlearea charts are like as below figures, <Figure 31 Output: Candleline and Candlearea Charts>, and please refer to the API document and sample charts for further information.



<Figure 33 Output: Candleline and Candlearea Charts>

6.27. Gauge Charts

6.27.1. Circular Gauge

The definition of the Circular gauge starts with <CircularGauge> and ends with </CircularGauge>. It has a range of 0 to 360 degrees. The default starting angle, 0 degrees, is toward 3 o'clock (or east), and positive angle measurement is clockwise. The Circular gauge is composed of three parts: the background frame, the needle and the needle cover. You can make various shapes by using the properties of the three parts. The Circular gauge has the default property values but the values of the following properties should be defined by users.

- startAngle
- minimumAngle
- maximumAngle
- value
- minimum
- maximum

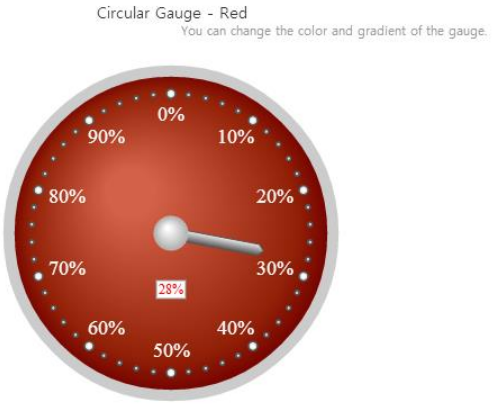
```
<KoolChart backgroundColor="0xEEEEEE" cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text=" Circular Gauge - Red"/>
```

```
                    <SubCaption text="You can change the color and gradient of the gauge" textAlign="right"
paddingRight="10" fontSize="11" />
            </Options>
            <CurrencyFormatter id="numFmt" precision="0" currencySymbol="%" alignSymbol="right"/>
            <CircularGauge width="300" height="300"   valueChangeFunction="valueChangeFunc"
                                    startAngle="270" minimumAngle="0" maximumAngle="360"
                                    minimum="0" maximum="100" value="28"
                                    interval="10" minorInterval="2"
                                    formatter="{numFmt}"
                                    padding="10"
                                    labelGap="10"
                                    tickLabelStyleName="tickText"
                                    valueLabelStyleName="valueText"
                                    editMode="true" liveDragging="true"
                                    showDataTip="true"
                                    tickColor="0xCCCCCC"
                                    minorTickColor="0x932108"
                                    coverRadiusRatio="0.1"
                                    hideTickLabel="first"
                                    needleThickness="10"
                                    pointThickness="5"
                                    needleLengthRatio="0.6"
                                    needlePointStyle="steeple"
                                    needleBackLengthRatio="0"
                                    isValueTop="false"
                                    valueYOffset="50">
                        <frameStroke>
                                <Stroke color="0xCCCCCC" weight="10"/>
                        </frameStroke>
                        <frameFill>
                                <SolidColor color="0x932108"/>
                        </frameFill>
                        <needleFill>
                                <LinearGradient angle="90">
                                        <entries>
                                                <GradientEntry color="0xEEEEEE" ratio="0" alpha="1"/>
                                                <GradientEntry color="0x555555" ratio="1" alpha="1"/>
                                        </entries>
                                </LinearGradient>
                        </needleFill>
                        <needleCoverFill>
                                <RadialGradient>
                                        <entries>
                                                <GradientEntry color="0xFFFFFF" ratio="0" alpha="1"/>
```

```
                                        <GradientEntry color="0xB0B0B0" ratio="1" alpha="1"/>
                                    </entries>
                                </RadialGradient>
                            </needleCoverFill>
        </CircularGauge>
        <Style>
                .valueText {
                        fontSize:12;
                        fontFamily:Myriad;
                        textAlign:center;
                        borderColor:0x999999;
                        backgroundColor:0xFFFFFF;
                        backgroundAlpha:1;
                        paddingTop:2;
                        borderThickness:1;
                        borderAlpha:1;
                        borderStyle:solid;
                        color:0xFF0000;
                }
                .tickText {
                        fontFamily:Myriad;
                        fontSize:18;
                        color:0xFFFFFF;
                }
        </Style>
</KoolChart>
```

<*Example 46 Example: Circular Gauge*>

The following is the output produced by the above layout.



<*Figure 34 Output: Circular Gauge*>

6.27.2. Half-Circular Gauge

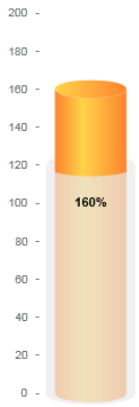
The definition of the Half-Circular gauge starts with <HalfCircularGauge> and ends with </HalfCircularGauge>. The *startAngle* property is not available for the Half-Circular gauge. The initial value of the *minimumAngle* property is 180 degrees (9 o'clock). If the value of the minimumAngle property is smaller than 180, the chart will set the value of minimumAngle to 180 automatically, and minimumAngle can not be larger than 360. So the range of the Half-Circular gauge is from 9 o'clock to 3 o'clock. And other styles and properties are the same to those of the Circular gauge.

6.27.3. Cylinder Gauge

There are two types of the Cylinder gauge: the vertical type and the horizontal type. The Vertical Cylinder gauge starts with <VCylinderGauge> and ends with <VCylinderGauge/>, and the Horizontal Cylinder gauge starts with <HCylinderGauge> and ends with <HCylinderGauge/>.
The following is an example layout for the Vertical Cylinder gauge.

```
<KoolChart  backgroundColor="0xFFFFFF"  cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Gauge"/>
        </Options>
        <VCylinderGauge id="cy1" width="30%" height="180"
                                minimum="0" maximum="160"
                                labels="[0, 20, 40, 60, 80, 100, 120, 140, 160]"
                                value="50"
                                targetMark="150"
                                snapInterval="1"
                                tickThickness="2"
                                tickLength="2"
                                tickInterval="20"
                                liveDragging="true"
                                tickColor="0x000000"
                                valueLabelXOffset="10"
                                valueLabelStylename="valueLabel"
        />
        <Style>
            .valuelabel{
                fontWeight:bold;
             }
        </Style>
</KoolChart>
```

<Example 47 Example: Vertical Cylinder Gauge>



<Figure 35 Output: Vertical Cylinder Gauge>

6.27.4. Linear Gauge

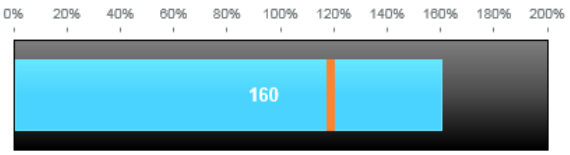
There are two types of the Linear gauge, the vertical type and the horizontal types. The Vertical Linear gauge starts with <VLinearGauge> and ends with <VLinearGauge/>, and the Horizontal Linear gauge starts with <HLinearGauge> and ends with <HLinearGauge/>.

The following is an example layout for the Horizontal Linear gauge.

```
<KoolChart backgroundColor="0xFFFFFF"  cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Gauge"/>
        </Options>
        <HLinearGauge id="hl" width="300" height="100"
                                minimum="0" maximum="100"
                                liveDragging="true"
                                labels="[0, 25, 50, 75, 100]"
                                value="50"
                                targetMark="90"
                                tickThickness="1"
                                tickLength="4"
                                tickInterval="5"
                                snapInterval="1"
                                linearColumnWidth=".6"
                                tickColor="#000000"
                                valueLabelFontWeight="bold"
        />
        <Style>
            .valuelabel {
                fontWeight:bold;
```

```
            }
        </Style>
</KoolChart>
```

<*Example 48 Example: Horizontal Linear Gauge*>



<*Figure 36 Output: Horizontal Linear Gauge*>

6.28. Slide Charts

You can use the Slide chart as a way of presenting data. After creating several charts regardless of the chart types, you can combine them as a single chart.

The following rules must be followed to create the Slide chart.

1. The layouts for each chart should be added by calling the setSlideLayoutSet() function.
2. The data for each chart should be added by calling the setSlideDataSet() function.

For example, if you want to create a Slide chart which has 5 different chart types, then you need to prepare five pairs of dataset and layout. The type for the layout and the dataset should be Array.

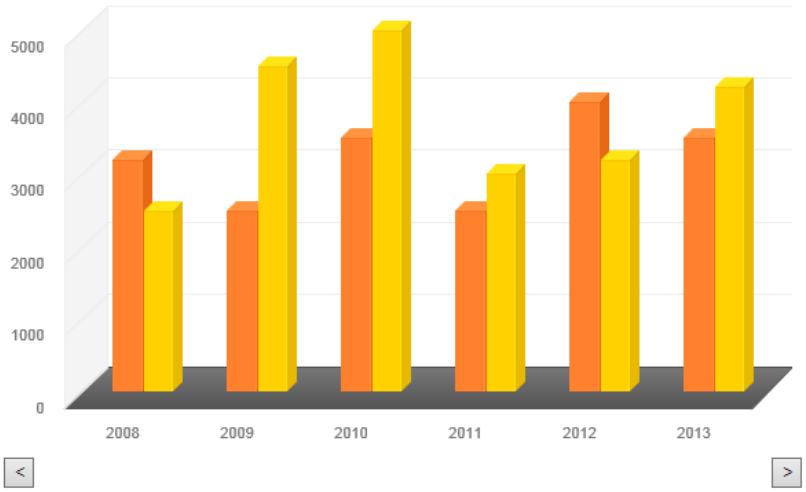
The following is an example layout for the Slide chart.

```
<script language="JavaScript" type="text/javascript">

var chartVars = "KoolOnLoadCallFunction=KoolChartOnLoad";

function KoolChartOnLoad()
{
        var layout1 = getCartesianLayout("Column2D","Column Char ",["Profit"]);
        var layout2 = getCartesianLayout("Line2D","Line Chart",["Profit"]);
        var layout3 = getCartesianLayout("Column3D","Column 3D Chart with Multi Dataset",["Profit","Cost"]);
```

```
        // Creating five layouts and five datasets.
        layoutSet = [layout1, layout2, layout3, layout1, layout2];
        dataSet = [chartData, chartData,chartData, chartData, chartData];

        // Adding the layouts.
        document.getElementById("chart").setSlideLayoutSet(layoutSet);

        // Adding the dataset.
        document.getElementById("chart").setSlideDataSet(dataSet);
}

// This function returns the layouts as the string type.
// This function can be customized.
// Parameters
// type: chart type
// title: caption
// dataField: data field name
function getCartesianLayout(type, title, dataField)
{
        var layout="<KoolChart borderStyle='solid'>"
                                +"<Options><Caption text='" + title +"'/></Options>"
                                +"<" + type + "Chart showDataTips='true'>"
                                +"<series>";

                for(var i=0; i<dataField.length; ++i) {
                        layout += "<" + type +"Series yField='" + dataField[i] + "' displayName='" + dataField[i]
+ "'/>"
                }

                layout +="</series>"
                                +"<horizontalAxis>"
                                +       "<CategoryAxis categoryField='Month'/>"
                                +"</horizontalAxis>"
                                +"</" + type + "Chart>"
                                +"</KoolChart>";
        return layout;
}
// Creating the dataset as an array type
var chartData = [{"Month":"Jan", "Profit":13000},
        {"Month":"Feb", "Profit":12000},
        {"Month":"Mar", "Profit":15000},
        {"Month":"Apr", "Profit":22200},
        {"Month":"May", "Profit":18000},
        {"Month":"Jun", "Profit":15000},
```

```
{"Month":"Jul", "Profit":22000},
{"Month":"Aug", "Profit":14000},
{"Month":"Sep", "Profit":26000},
{"Month":"Oct", "Profit":22000},
{"Month":"Nov", "Profit":28000},
{"Month":"Dec", "Profit":34000}];
```

<Example 49 Example: Slide Chart>

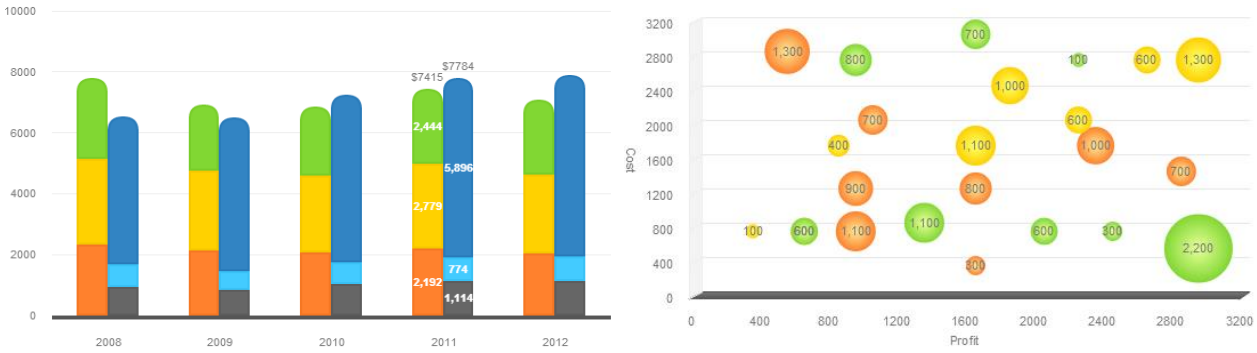


<Figure 37 Output: Slide Chart>

7. Using Layouts for Advanced Users

7.1. Displaying Numeric Values in Charts

You can display the numeric values in charts as follows:



The *labelPosition* property of the series is used to display the numeric values in the chart. The valid values of labelPosition are different based on the type of the series. The following table shows the valid values for each series.

Series	Values	Descriptions
ColumnSeries(2D, 3D), BarSeries(2D, 3D), ImageSeries	inside, outside, both, none	Specifies the position at which the lables are displayed.
Line2DSeries, Area2DSeries	up, down, both, none	Specifies the position at which the lables are displayed.
Bubble3DSeries	inside,none	Specifies the position at which the lables are displayed..
PieSeries(2D,3D)	inside, outside, callout, insideWithCallout	Specifies the position at which the lables are displayed. (callout: Displays the numeric values outside the chart with the line. insideWithCallout: Displays the numeric value inside the chart. But if not enough space in the slice, the "callout" method is used.)

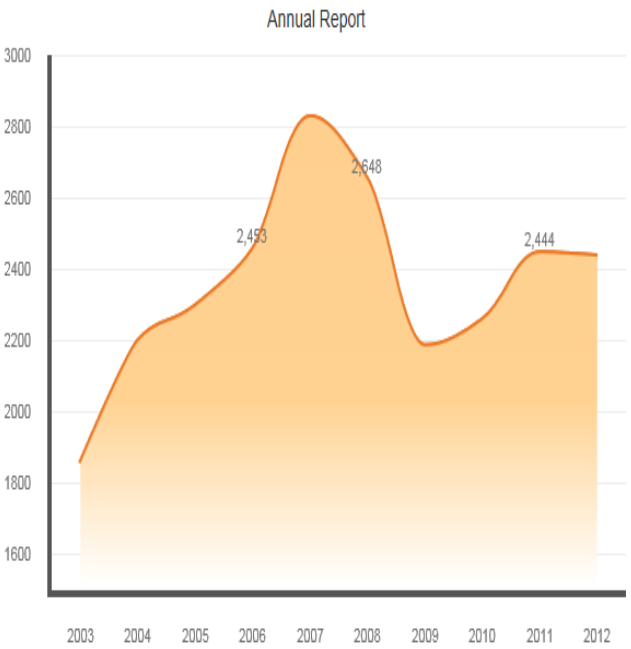
<Table 18 Properties: labelPosition>

7.2. Setting Colors in Series Items

The data item has two types of the color, the line (or stroke) color and the fill color.

```
<series>
   <Area2DSeries yField="Profits"
         form="curve"
         displayName="Profits">

      <areaFill>
            <SolidColor
 color="0x00FF99"/>
     </areaFill>
     <areaStroke>
        <Stroke color="0xFF0000"
weight="3"/>
     </areaStroke>
   </Area2DSeries>
</series>
```

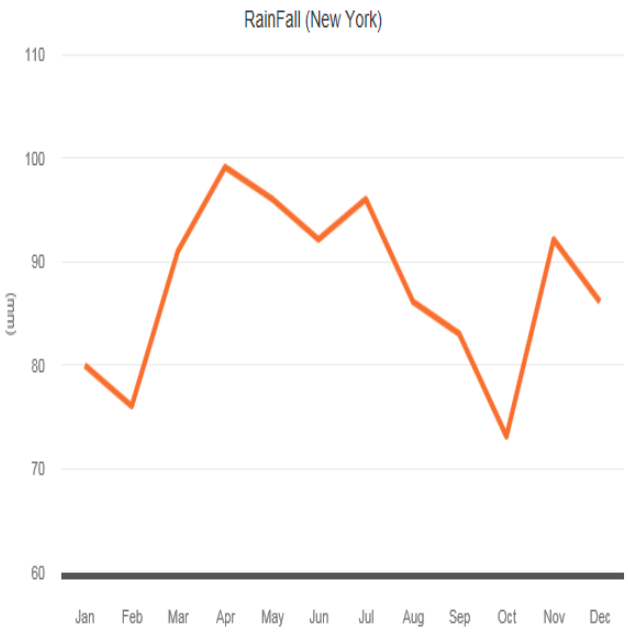
- Important: The property names for the Area chart are areaFill and areaStroke.



<Example 50 Example: Layout and Output of Area Chart with Fill and Stroke>

```
<Line2DChart>
    <horizontalAxis>
        <CategoryAxis
categoryField="Month"/>
    </horizontalAxis>
    <series>
        <Line2DSeries yField="Profit"
form="curve" displayName="Profit">
           <lineStroke>
              <Stroke color="0xF0FF00"
weight="4"/>
// color: line color.
// weight: line thickness.
           </lineStroke>
        </Line2DSeries>
    </series>
</Line2DChart>
```

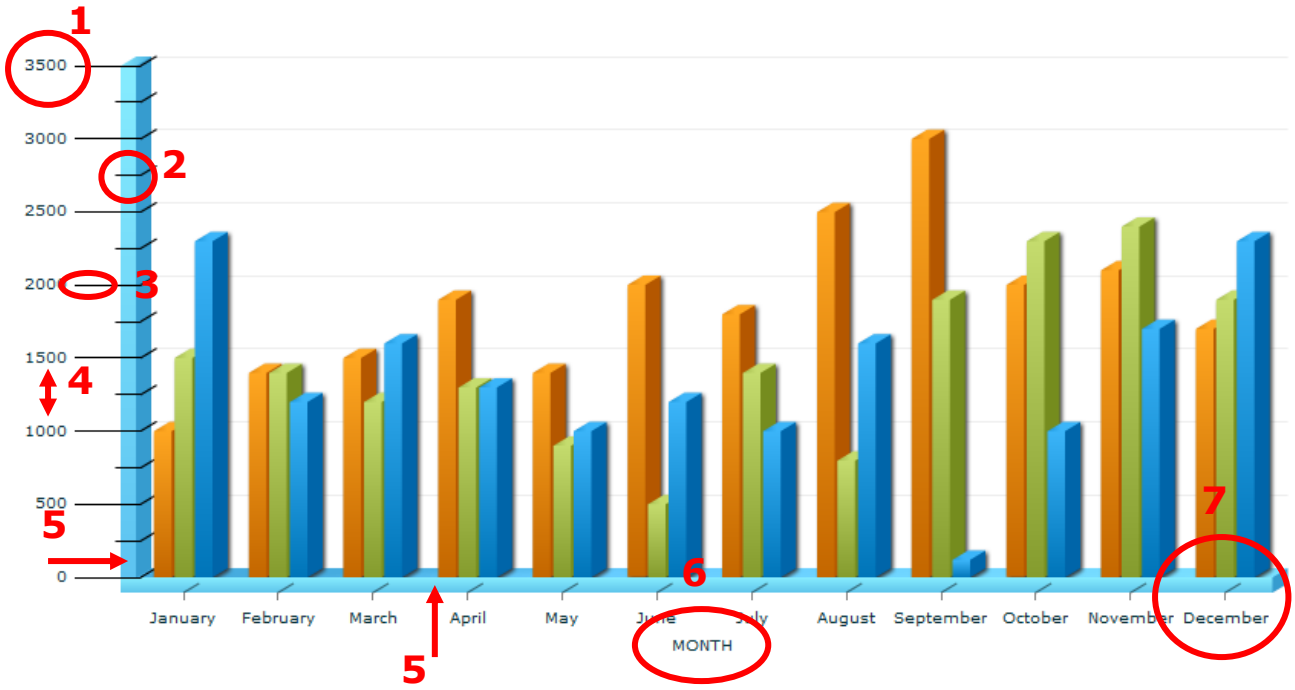
- Important: The property name for the Line chart is lineStroke.



<Example 51 Example: Layout and Output of Line Chart with Stroke>

7.3. Applying axis styles

7.3.1. About Axis Styles



<Figure 38 Axis Styles>

<Figure 38 Axis Styles> shows an example of the chart that uses the axis style applied to the horizontal and the vertical axes. Each number of the above example indicates:

1. The maximum value of the vertical axis. It is valid only for <LinearAixs>.
2. It is the minor tick. You can define the length and style of the minor tick (line color, line thickness, etc).
3. It is tick. You can define the position, length and style of the tick.
4. It is the gap of two consecutive ticks. In the above figure, the gap is 500. It is valid only for <LinearAixs>.
5. The styles of the vertical axis line and the horizontal axis line. In the above figure, the sky-blue color is defined for the line of the axes.
6. The title of the horizontal axis.
7. The position of the tick. The tick can be located between two consecutive labels or at the top of the label. The above figure shows that the tick is located at the top of the label.

The layout for the above example is as follows:

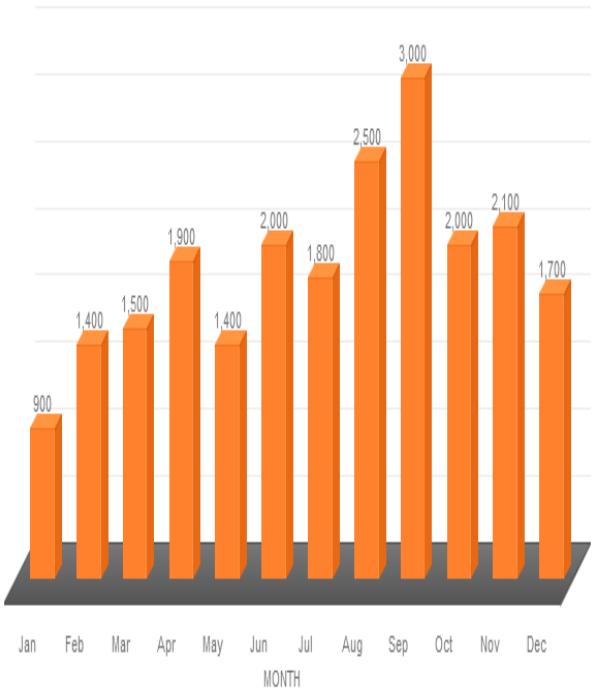
```
<Column3DChart>
    <horizontalAxis>                                                    7
6       <CategoryAxis categoryField="Month" ticksBetweenLabels="false"
```

```
title="MONTH" displayName="Month"/>
        </horizontalAxis>
        <verticalAxis>                                                  1
                <LinearAxis interval="500" baseAtZero="true" maximum="3500"/>
        </verticalAxis>
        <horizontalAxisRenderers>
                <Axis3DRenderer visible="true"
                        tickLength="5"
                        minorTickLength="5"
                        tickPlacement="outside"
                        placement="bottom"
                        canDropLabels="false"
                        showLabels="true"
                        labelAlign="center">
                        <axisStroke>
                                <Stroke weight="10" color="0x66CCFF" caps="none"/>
                        </axisStroke>
                        <tickStroke>
                                <Stroke weight="1" color="0x000000" alpha="0.5" />
                        </tickStroke>
                        <minorTickStroke>
                                <Stroke weight="1" color="0x000000" caps="square"/>
                        </minorTickStroke>
                </Axis3DRenderer>
        </horizontalAxisRenderers>

          <verticalAxisRenderers>
                <Axis3DRenderer visible="true"
                        tickLength="30"
                        minorTickLength="3"
                        tickPlacement="left"
                        placement="left"
                        canDropLabels="false"
                        showLabels="true"
                        labelAlign="center">
                        <axisStroke>
                                <Stroke weight="10" color="0x66CCFF" caps="none"/>
                        </axisStroke>
                        <tickStroke>
                                <Stroke weight="1" color="0x000000" />
                        </tickStroke>
                        <minorTickStroke>
                                <Stroke weight="1" color="0x000000" caps="square"/>
                        </minorTickStroke>
                </Axis3DRenderer>
        </verticalAxisRenderers>

        ......

</Column3DChart>
```

<Example 52 Example: Layout for Axis Styles>

7.3.2. Showing and Hiding Axes Using Properties

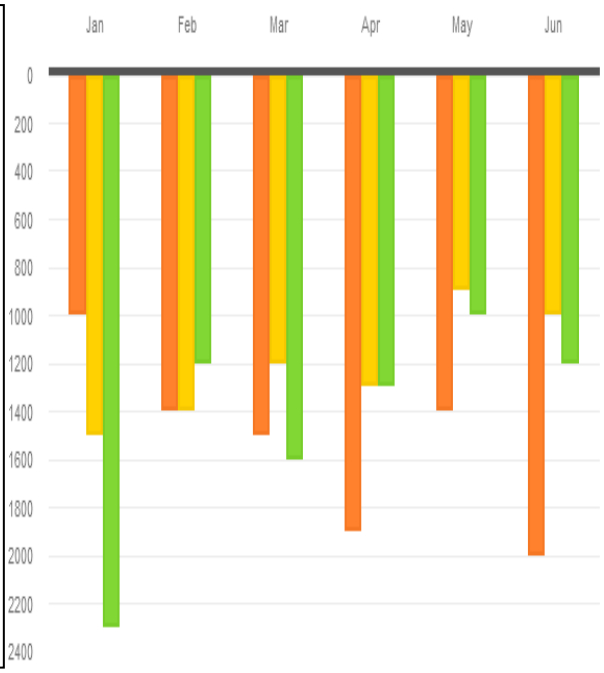
```
<Column3DChart>

        ......


    <horizontalAxisRenderers>
        <Axis3DRenderer
              placement="bottom"
              showLabels="true"
              labelAlign="center">
        </Axis3DRenderer>
    </horizontalAxisRenderers>

    <verticalAxisRenderers>
         <Axis3DRenderer
               visible="false">
         </Axis3DRenderer>
    </verticalAxisRenderers>

        ......

</Column3DChart>
```



<Example 53 Example: Layout and Output for Hiding Vertical Axis>

7.3.3. Changing the Position of Axes

```
<KoolChart>
  <Column3DChart>

        ......

     <horizontalAxisRenderers>
        <Axis2DRenderer
              placement="top"
              showLabels="true”>
        </Axis3DRenderer>
     </horizontalAxisRenderers>

     <verticalAxisRenderers>
        <Axis2DRenderer
              placement="left"
              showLabels="true"
              labelAlign="center">
        </Axis3DRenderer>
     </verticalAxisRenderers>

        ......
```

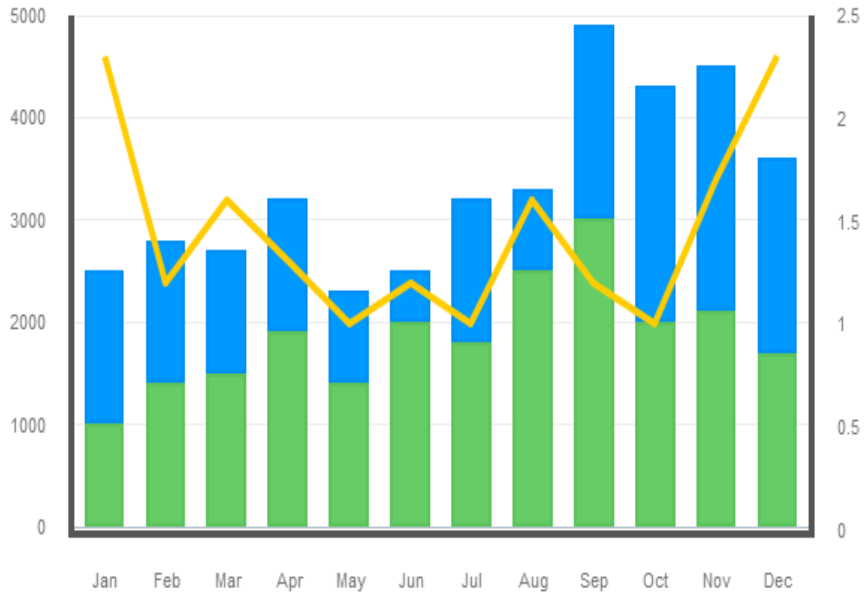


<Example 54 Example: Layout and Output for Changing Axis Position>

7.3.4. Creating Dual Y-Axes

```
<KoolChart cornerRadius="12" borderStyle="solid">
    <Options>
        <Caption text="Annual Report" />
        <SubCaption text="2008" />
        <Legend />
    </Options>
    <Combination2DChart showDataTips="true">
        <horizontalAxis>
            <CategoryAxis categoryField="Month" />
        </horizontalAxis>
        <verticalAxis>   // Creating a vertical axis for a column series.
            <LinearAxis id="axis1"/>
        </verticalAxis>
        <series>
            <Column2DSeries yField="Profit" displayName="Profit">
                <showDataEffect>
                    <SeriesInterpolate />
                </showDataEffect>
                <fill>
                    <SolidColor color="0x66CC66" />
                </fill>
            </Column2DSeries>
            <Line2DSeries selectable="true" yField="Cost" displayName="Cost">
                <verticalAxis> // Creating a vertical axis for a line series.
                    <LinearAxis id="axis2"/>
                </verticalAxis>
                <showDataEffect>
                    <SeriesInterpolate />
                </showDataEffect>
                <lineStroke>
                    <Stroke color="0xFFCC00" weight="3" />
                </lineStroke>
            </Line2DSeries>
        </series>
        <verticalAxisRenderers> // Creating verticalAxisRenderers for each series.
            <Axis2DRenderer axis="{axis1}"/>
            <Axis2DRenderer axis="{axis2}"/>
        </verticalAxisRenderers>
    </Combination2DChart>
</KoolChart>
```

<Example 55 Example: Layout for Two Vertical Axes>

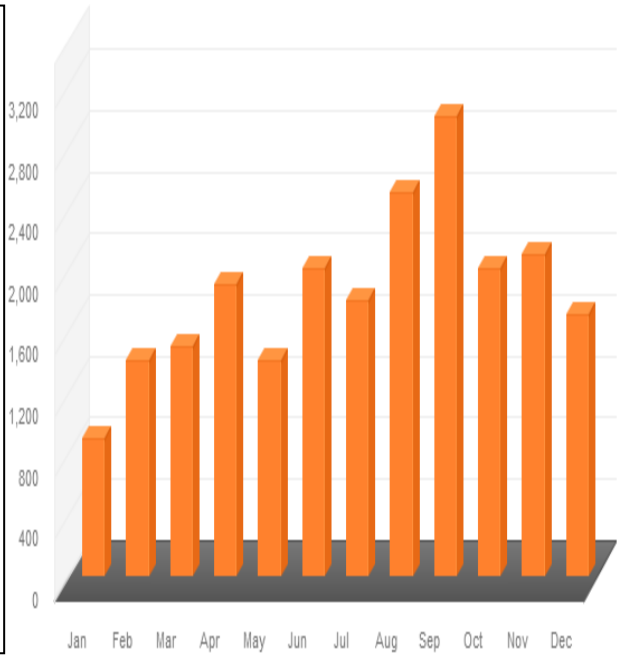


<*Figure 37 Output: Chart with Two Vertical Axes*>

7.3.5. Adding Thousands Separators

<LinearAxis> represents labels as numeric values. You can use the thousands separator to specify a delimiter character that will be used to break a value into groups of three digits, counting left from the decimal mark.

```
<KoolChart cornerRadius="12">
  <Options>
     <Caption text="Anual Report"/>
  </Options>
  <NumberFormatter id="numfmt"
     useThousandsSeparator="true"/>
  <Column3DChart>
    <horizontalAxis>
       <CategoryAxis
          categoryField="Month"/>
    </horizontalAxis>
    <verticalAxis>
      <LinearAxis
           formatter="{numfmt}"/>
    </verticalAxis>
    <series>
       <Column3DSeries yField="Profit"
          displayName="Profit"/>
    </series>
  </Column3DChart>
</KoolChart>
```



<*Example 56 Example: Layout and Output for Adding Thousands Separator*>

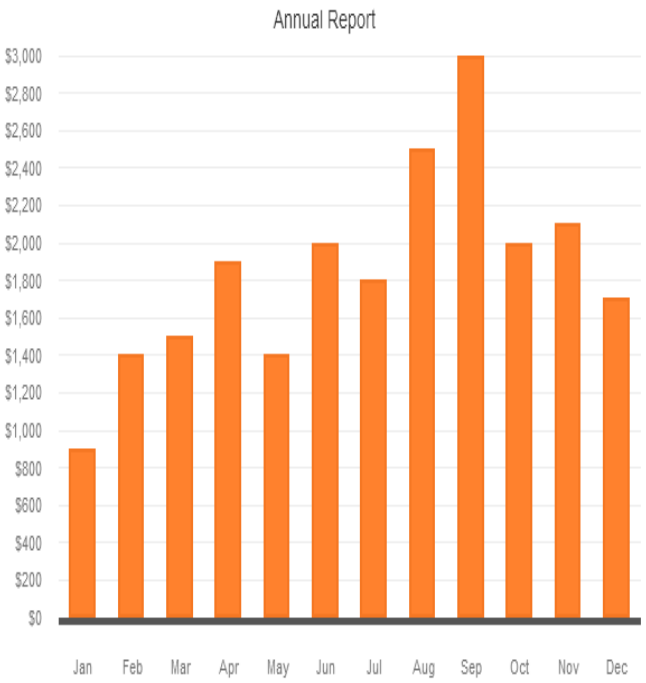
Properties	Values	Descriptions
useThousandsSeparator	True | false (default: true)	Whether or not to add the thousands separator.
useNegativeSign	true | false (default: true)	Indicates how to display the negative sign. If the value of useNegativeSign is true, displays negative values with a leading minus sign; otherwise, negative values are enclosed in parentheses
precision	Number (default: -1)	Specifies the decimal precision. If the value of precision is -1, the precision will be ignored.
thousandsSeparatorTo	Separator (default: comma (,))	Specifies the delimiter as a thousands separator.
decimalSeparatorTo	Separator (default: dot (.))	Specifies the delimiter as a decimal point.
rounding	down, nearest, up, none (default: none)	Specifies the rounding methods.

<Table 19 Properties: NumberFormatter>

7.3.6. Adding Currency Symbols

You can add the currency symbol to the numeric values. The following example shows how to add the currency symbol ($) to the numeric values.

```
<KoolChart>
    <Options>
      <Caption text="Anual Report"/>
    </Options>
    <CurrencyFormatter id="fmt"
           currencySymbol="$"
           alignSymbol="left"/>
    <Column2DChart showDataTips="true"
              fontSize="11">
        <horizontalAxis>
            <CategoryAxis
          categoryField="Month"/>
        </horizontalAxis>
        <verticalAxis>
          <LinearAxis
formatter="{fmt}"/>
        </verticalAxis>
        <series>
            <Column3DSeries
              yField="Profit"/>
        </series>
        </Column3DChart>
</KoolChart>
```



<Example 57 Example: Layout and Output for Adding Currency Symbol>

Properties	Values	Descriptions
currencySymbol	Symbol (default: $)	Specifies the currency symbol.
alignSymbol	left | right (default: left)	Specifies the position of the currency symbol.
useThousandsSeparator	True | false (default: true)	Whether or not to use the thousands separator.
useNegativeSign	true | false (default: true)	Indicates how to display negative values. If the value of useNegativeSign is true, displays negative values with a leading minus sign otherwise, negative values are enclosed in parentheses
precision	Number (default: -1)	Specifies the decimal precision. If the value of precision is -1, the precision will be ignored.
thousandsSeparatorTo	Separator (default: comma	Specifies the delimiter as a thousands

	(,))	separator.
decimalSeparatorTo	Separator (default: dot (.))	Specifies the delimiter as a decimal point.

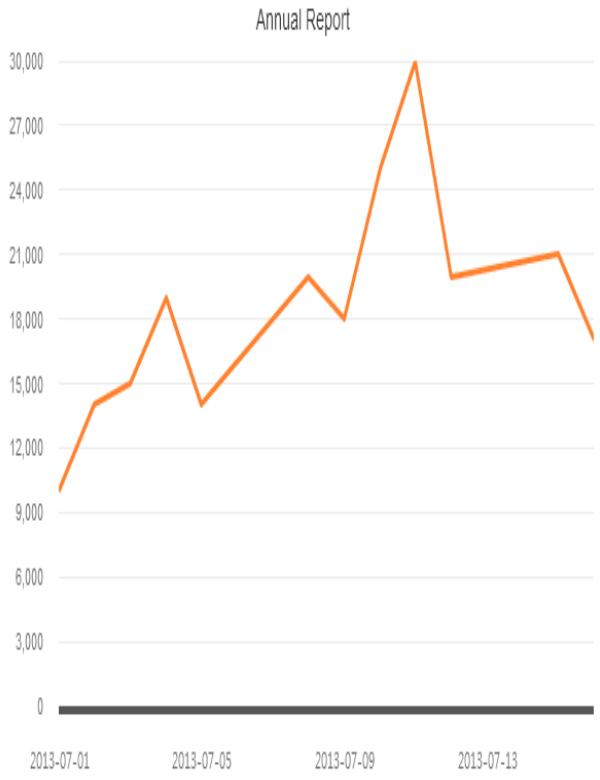
<*Table 20 Properties: CurrencyFormatter*>

7.3.7. Using Date Formats in DateTimeAxis

If <DateTimeAxis> is defined for the horizontal axis of the Line chart, the data will be displayed along the horizontal axis by using the date format. For example, if you set the value of labelUnits to "days", the labels will be displayed by using the default date format of your computer (e.g. "MM/DD/YYYY"). You can change the date format of the data label by using <DateFormatter>.

The following example shows how to use <DateFormatter>.

```
<KoolChart>
  <Options>
     <Caption text="Anual Report"/>
  </Options>
  <DateFormatter id="dateFmt"
  formatString="YYYY-MM-DD" />
  <NumberFormatter id="curFmt" />
  <Line2DChart showDataTips="true"
  fontSize="11">
    <horizontalAxis>
       <DateTimeAxis dataUnits="days"
       labelUnits="days" interval="4"
       formatter="{dateFmt}"
       alignLabelsToUnits="false"
       displayLocalTime="true" />
    </horizontalAxis>
    <verticalAxis>
      <LinearAxis formatter="{curFmt}"/>
    </verticalAxis>
    <series>
      <Line2DSeries xField="Date"
      yField="Profit"
      displayName="Profit"/>
    </series>
  </Line2DChart>
</KoolChart>
```



<Example 58 Usage Example: DateFormatter>

If you want to change the default date format, you need to define <DateFormatter> and assign the unique ID to the <DateFormatter>. Once the <DateFormatter> is defined with the unique id, you need to set the value of the *formatter* property of <DateTimeAxis> to the ID of the <DateFormatter>.

Note: Because the ID is a unique object, it must be enclosed in the curly braces. (Please refer to <Example 8 Example: How to Create the Property Value Using the ID>

The following table shows how to use formatString. For example, if you want to display a date string as "2009/12/01", you must specify the pattern string as "YYYY/MM/DD".

Pattern Characters	Descriptions	Examples
Y	Year	YY = 05 YYYY = 2005 YYYYY = 02005
M	Month	M = 7 MM= 07 MMM=Jul

		MMMM= July
D	Day	D = 4 DD = 04 DD = 10
H	Hour	H = 1 HH = 01
N	Minute	N = 1 NN = 01
S	Second	SS = 30

<Table 21 Descriptions: FormatString>

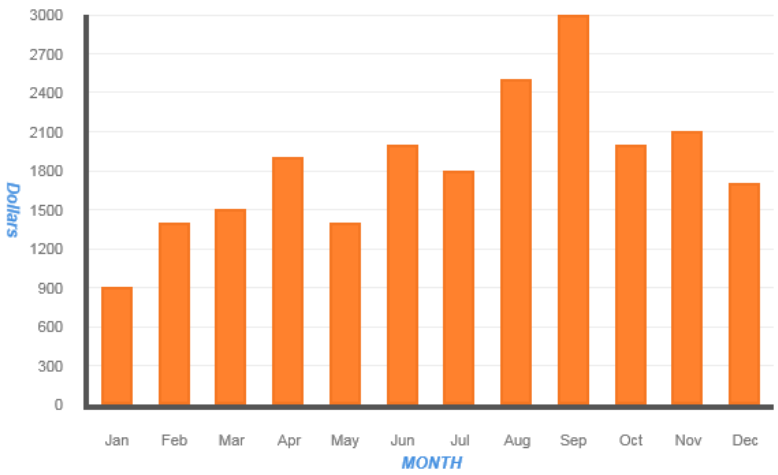
7.3.8. Adding Titles

You can add the title to the axis. For Example, you can add "Profit" as a title for the vertical axis and "Month" for the horizontal axis. The following example shows how to add the title to the axis.

```
<KoolChart>
        <Options>
                <Caption text="Anual Report"/>
        </Options>
        <Column2DChart showDataTips="true">
        <horizontalAxis>
           <CategoryAxis categoryField="Month" title="Month"/>// "Month" is the title for the horizontal axis
        </horizontalAxis>
        <verticalAxis>
            <LinearAxis title="Profit"/> // "Profit" is the title for the vertical axis
        </verticalAxis>
        <horizontalAxisRenderers>              // <horizontalAxisRenderers> is defined
                        <Axis2DRenderer   axisTitleStyleName="title"/>
        </horizontalAxisRenderers>
        <verticalAxisRenderers>                // <verticalAxisRenderers> is defined
                        <Axis2DRenderer   axisTitleStyleName="title"/>
        </verticalAxisRenderers>
        <series>
           <Column2DSeries yField="Profit"/>
        </series>
    </Column2DChart>
    <Style> // The Style node is defined for the style of the axis title.
        .title
```

```
        {
                fontSize:12;
                color:0x0000FF;
                fontWeight:"bold";
        }
    </Style>
</KoolChart>
```

<Example 59 Usage Example: Axis Title>

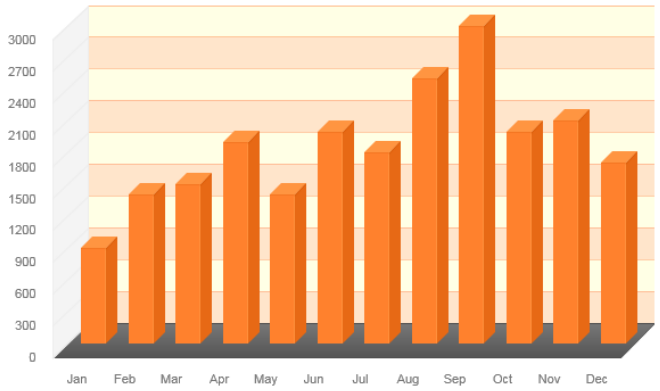


<Figure 40 Output: Axis Title>

7.4. Designing Chart Background

7.4.1. Adding Gridlines

All charts which have X and Y axes can have <backgroundElements>. <backgroundElements> can be used to decorate the background area of the chart.



<Figure 39 Output: backgroundElements>

In the above example, only horizontal lines are used to decorate the background area. If you want to add the grid lines (vertical or horizontal), you need to keep in mind the following options.

- direction = "horizontal", "vertical", "both".
- horizontalChangeCount = "1" : Changes the colors of the are between the lines. The color defined in <horizontalFill> (or <verticalFill>) and the color define in <horizontalAlternateFill> (or verticalAlternateFill>) are used alternately.
- horizontalStroke: Defines the color of the line.
- horizontalFill: The initial color. In the above example, the initial color is white.
- horizontalAlternateFill: The alternate color. In the above example, the alternate color is orange.

The following layout shows how to add the horizontal lines to the background of the chart. If you want to add the vertical lines instead of the horizontal lines to the background of the chart, you must set the value of the direction property to "vertical".

```
<KoolChart>
    <Column3DChart>
        <backgroundElements>
            <GridLines direction="horizontal" verticalChangeCount="1" horizontalChangeCount="1">
                <horizontalStroke>
                    <Stroke color="0xFF9966" alpha="0.5" weight="1"/>
                </horizontalStroke>
                <horizontalFill>
                    <SolidColor color="0xFFCC99" alpha="0.5"/>
                </horizontalFill>
                <horizontalAlternateFill>
                    <SolidColor color="0xFFFFCC" alpha="0.5"/>
                </horizontalAlternateFill>
                <verticalStroke>
                    <Stroke color="0xFF9966" alpha="0.5" weight="1"/>
                </verticalStroke>
                <verticalFill>
                    <SolidColor color="0xFFCC99" alpha="0.5"/>
                </verticalFill>
                <verticalAlternateFill>
                    <SolidColor color="0xFFFFCC " alpha="0.5"/>
                </verticalAlternateFill>
            </GridLines>
        </backgroundElements>
    </Column3DChart>
</KoolChart>
```

<Example 60 Example: backgroundElements>

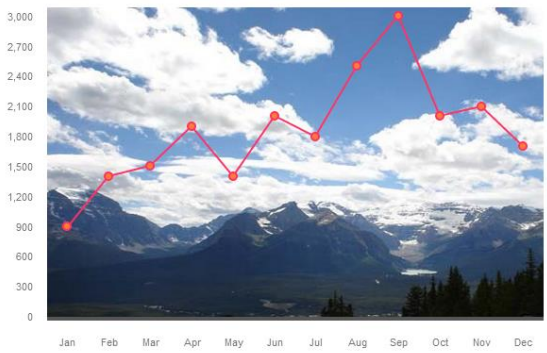
7.4.2. Adding Images

In order to add images to the background of the chart, you can use <backgroundElements> or <annotationElements>. The background elements are displayed in the backgound of the chart and the annotation elements are displayed in the foregound of the chart.

The following example shows how to add an image using <backgroundElements>.

```
<KoolChart backgroundColor="0xFFFFEE"   cornerRadius="12" borderStyle="solid">
        <Line2DChart showDataTips="true">
                <horizontalAxis>
                        <CategoryAxis categoryField="Month"/>
                </horizontalAxis>
                <series>
                        <Line2DSeries yField='Profit' displayName='Profit'
itemRenderer='mx.charts.renderers.CircleItemRenderer' radius='5'>
                                <showDataEffect>
                                        <SeriesInterpolate />
                                </showDataEffect>
                                <lineStroke><Stroke color='0xFF3366' weight='2'/></lineStroke>
                                <stroke><Stroke color='0xFF3366' weight='2'/></stroke>
                        </Line2DSeries>
                </series>
                <backgroundElements>
                        <Image source="http://www.koolchart.com/image/koolbg.jpg"
maintainAspectRatio="false" alpha="1"/>
                </backgroundElements>
        </Line2DChart>
</KoolChart>
```

<Example 61 Example: Adding Image to Background>



<Figure 42 Output: Adding Image to Background>

The following example shows how to add an image to the foreground of the chart.

```
<annotationElements>
                        <Image source="http://www.koolchart.com/image/koolbg.jpg"
maintainAspectRatio="false" alpha="0.25"/>
</ annotationElements>
```

<Example 62 Example: Adding Image to Foreground>

If you add an image to the foreground of the chart and set the alpha value (transparency) of the image to 1, the chart will be invisible. To avoid this problem, you need to set the proper value of the alpha property in <Image>.

The following table shows the properties of <Image>.

Properties	Descriptions
source	Specifies the URL of the image.
maintainAspectRation	Whether or not the aspect ratio is the same to the original image. If it is true, the chart will try to keep the ratio of the image otherwise, the chart will adjust the image to fit the size of the chart
alpha	The transparency of the image (Effective value : 0~1)

<Table 22 Properties: Image>

7.5. Creating Effects in Charts

KoolChart supports the three types of effects, which are SeriesSlide, SeriesInterpolate and SeriesZoom.

- SeriesInterpolate: The SeriesInterpolate effect gradually moves the data points from the start value (0) to the actual values.
- SeriesSlide: The SeriesSlide effect offers the sliding effect while data rendering.
- SeriesZoom: The SeriesZoom effect starts at the reference point in the chart and gradually expands

The following table shows the properties of the effect.

Properties	Values	Descriptions	Effects
duration	millisecond (default: 500)	Specifies the time required to	Common

		complete the whole effect.	
elementOffset	millisecond (default: 20)	Specifies the delay time of the effect.	
minimumElementDuration	millisecond (default: 0)	Specifies the minimum amount of the time which is taken to complete the effect for each element.	
offset	millisecond(default: 0)	Specifies the delay time of the start of the effect.	
perElementOffset	Millisecond	Specifies the delay time of the start of each element.	
direction	left | right | up | down (default: left)	Specifies the direction of the sliding effect.	SeriesSlide
horizontalFocus	center | left | right (default: center)	Specifies the horizontal focal point of the zoom.	SeriesZoom
verticalFocus	center | top | bottom (default: center)	Specifies the vertical focal point of the zoom.	
relativeTo	chart | series (default: chart)	Specifies the boundary of the focal point of the zoom.	

<Table 23 Properties: Effect>

The following example shows how to create the effect using <SeriesSlide>.

```
<KoolChart>
    <Column3DChart>
        <Column3DSeries yField="Profit">
            <showDataEffect>
                <SeriesSlide duration="1000" // Uses <SeriesSlide> with specific options.
                    direction="down"
                    minimumElementDuration="20"
                    perElementOffset="0"
                    elementOffset="30"/>
            </showDataEffect>
        </Column3DSeries>
        <Column3DSeries yField="Cost">
            <showDataEffect>
                < SeriesSlide /> // Uses <SeriesSlide> with default options.
            </showDataEffect>
        </Column3DSeries>
```

```
    ……
    ……
</KoolChart>
```

<Example 63 Example: Effects Using <SeriesSlide>>

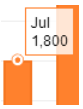
The effect of the above example runs as follows:

Each element starts its slide after 30 miliseconds of the previous element's slide, and the effect takes at least 20 miliseconds to complete its slide. The entire effect takes at least 1 second (1,000 miliseconds) to slide down the series data.

7.6. Showing DataTips (ToolTips) on Charts

Showing Tooltips on the chart (column, bar, area, line, etc.) is achieved by using the showDataTips property. The Tooltip text will be shown when the mouse cursor moves over the chart element. To display Tooltips on the chart you must set the value of the *showDataTips* property to "true". The following example shows how to display Tooltips when mouseover event occurs in the Column 3D chart. The *showDataTips* property is the common property of all charts.

```
<KoolChart>
    <Options>
        <Caption text="Annual Report"/>
    </Options>
    <Column3DChart showDataTips="true">
        <horizontalAxis>
            <CategoryAxis categoryField="Month" title="Month"/>
        </horizontalAxis>
        <series>
            <Column3DSeries yField="Profit" displayName="Profit">
                <showDataEffect>
                    <SeriesInterpolate/>
                </showDataEffect>
            </Column3DSeries>
            ……
```



< Tooltip displayed when mouseover event occurs >

```
</KoolChart>
```

<Example 64 Usage Example: Tooltips>

Once you set the value of the showDataTips property to "true" and set the value of the *displayName* property of Series to "Profit", you can have the Tooltip text like above example.

7.7. Creating Lines between Stacked DataSets in Column Charts

You may want to add the line which connects each data point in the chart in order to provide further information regarding the relationship between the data points. You can add the connecting line to the stacked column chart by setting the value of the *lineToEachItems* property to "true".

alwayShowLines: Indicates whether or not to connect the line for 0-valued data.

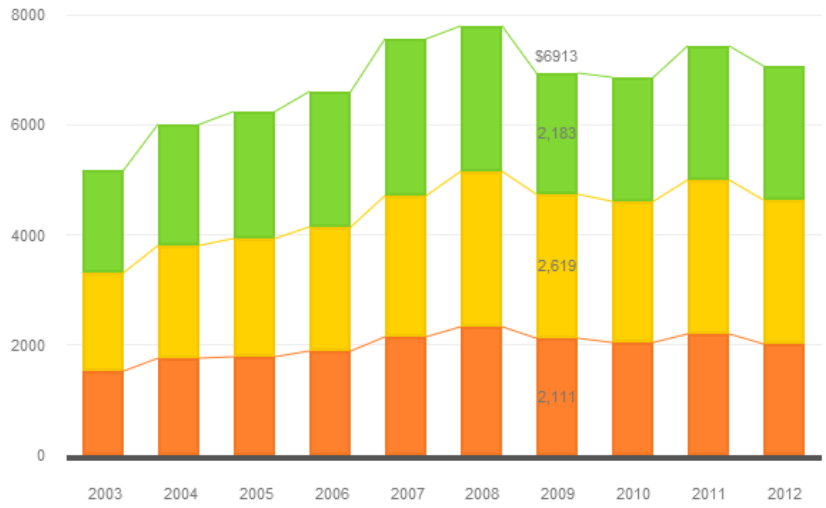
True - Connectss the line for all data including 0-valued data.

False – Connects the line except 0-valued data.

```
<Column2DChart showDataTips="true" type="stacked"> // Defines as a stacked type
        <horizontalAxis>
                <CategoryAxis categoryField="Month" />
        </horizontalAxis>
        <series>                                // Sets the value of <lineToEachItems> to true
                <Column2DSeries yField="Profit" lineToEachItems="true" alwayShowLines="true">
                        <linkLineStroke> // Defines the line styles
                                <Stroke weight="2" color="0x000000" caps="none" />
                        </linkLineStroke>
                        <showDataEffect>
                                <SeriesInterpolate />
                        </showDataEffect>
                </Column2DSeries>
                <Column2DSeries yField="Cost" lineToEachItems="true" alwayShowLines="true">
                        <linkLineStroke>
                                <Stroke weight="2" color="0x000000" caps="none" />
                        </linkLineStroke>
                        <showDataEffect>
                                <SeriesInterpolate />
                        </showDataEffect>
                </Column2DSeries>
                <Column2DSeries yField="Revenue" lineToEachItems="true" alwayShowLines="true">
                        <linkLineStroke>
                                <Stroke weight="2" color="0x000000" caps="none" />
                        </linkLineStroke>
                        <showDataEffect>
                                <SeriesInterpolate />
                        </showDataEffect>
```

```
                </Column2DSeries>
            </series>
</Column2DChart>
```

<*Example 65 Example: Lines between Data Points*>



<*Figure 41 Output: Lines between Data Points* >

7.8. Setting Functions for Click Events on Items

The itemClickJsFunction property enables you to pass information of the data item to your JavaScript function when the data item is clicked. To pass the information of the data item to the JavaScript function, you should create the JavaScript function first and then set the value of the *itemClickJsFunction* property to the function name. The *itemClickJsFunction* property should be defined in the <Chart> node (e.g. <Column3DChart>, <Bar3DChart>, etc).

The following example shows how to define the itemClickJsFunction property in the Pie chart. When users click on the slice of the Pie chart, the function named chartClick will be called.

```
<KoolChart  cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Anual Report" />
        </Options>
        <Pie2DChart showDataTips="true" itemClickJsFunction="chartClick">
                <series>
                        <Pie2DSeries field="Profit" nameField="Month" displayName="Profit">
                                <showDataEffect>
                                        <SeriesInterpolate/>
```

```
                                    </showDataEffect>
                                </Pie2DSeries>
                            </series>
                        </Pie2DChart>
</KoolChart>
```

<Example 66 Example: Call Functions>

The following is an example of the chartClick function.

```
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script src="KoolChart.js" language="javascript"></script>

<script language="JavaScript" type="text/javascript">

var chartVars;
var layoutURL = encodeURIComponent("chartLayout.xml");
chartVars += "&layoutURL="+layoutURL;

var dataURL =encodeURIComponent("singleData.xml");
chartVars += "&dataURL="+dataURL;

KoolChart.create("chart1", "chartHolder", chartVars, "100%", "100%");


// ------------------ Create the function called when the item is clicked ---------------

/*
// Parameters:
// seriesId: Series ID
// displayText: The tooltips shown when the mouse over the item.
// index: The index of the clicked item in the series.
// data: The record of the dataset declared for creating the chart.
// values: The value of the item. The value is an array and each element in the array is different depending on the chart type.
                        BarSeries       0: X-axis value, 1: Y-axis value
                        ColumnSeries    0: X-axis value, 1: Y-axis value
                        AreaSeries      0: X-axis value, 1: Y-axis value
                        BubbleSeries    0: X-axis value, 1: Y-axis value, 2: radius
                        LineSeries      0: X-axis value, 1: Y-axis value
                        PieSeries       0: value
*/
```

```
function chartClick(seriesId, displayText, index, data, values)
{
        alert("seriesId:"+seriesId+"₩ndisplayText:"+displayText+"₩nindex:"+index+"₩ndata:"+data+"₩nvalues:"+valu
es);
}

</script>
</head>

<body>
<div class="content">
        <div id="chartHolder" style="width:600px; height:400px;">
        </div>
</div>
</body>
</html>
```

<Example 67 Example: JavaScript Function Called When Item Click Occurs>

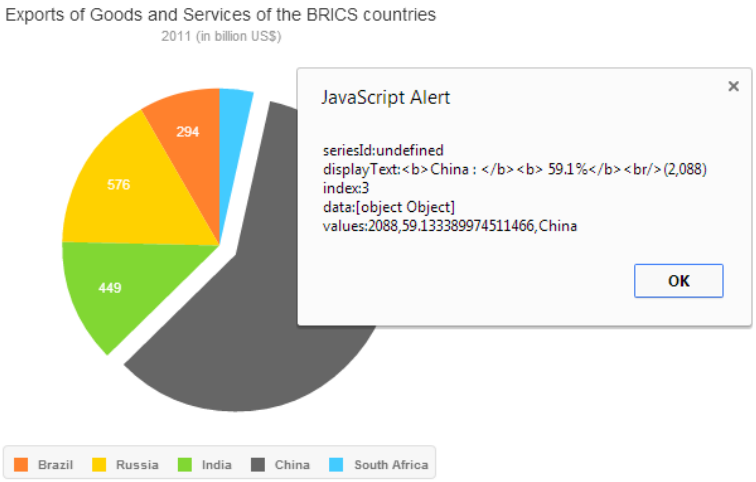
The following table describes the parameters used in the above example.

Parameters	Descriptions
seriesId	Series ID
displayText	The tooltips shown when the mouse over the item.
index	The index of the clicked item in the series. (The index of the leftmost item or the top item is 0.)
data	The record of the dataset declared for creating the chart.
values	The value of the item. This parameter is passed as an array data, and the elements of the array are determined based on the type of the chart as follows:

Bar2DSeries(Bar3DSeries)	0: X-axis value, 1: Y-axis value
Column2DSeries(Column3DSeries)	0: X-axis value, 1: Y-axis value
Area2DSeries	0: X-axis value, 1: Y-axis value
Bubble3DSeries	0: X-axis value, 1: Y-axis value 2: radius
Line2DSeries	0: X-axis value, 1: Y-axis value
Pie2DSeries(Pie3DSeries)	0: value

<Table 24 Parameters of the Function Called When Item Click Occurs>

The following is the output produced by the above layout and the function call.



<Figure 42 Output: Function Call When the Item Click Occurs>

7.9. Setting User-Defined Functions

You can use user-defined functions in the following cases.

1. Tooltips
2. Axis labels (axis or axis renderer)
3. Numeric data fields for Column, Bar or Pie charts

The user-defined function can be used by creating the JavaScript function and setting the function name to the value of the corresponding property in the layout. The *Formatter* property cannot be used with the user-defined function for the axis label or numeric data field (except Tooltips). If <Formatter> and <labelJsFunction> are defined at the same time, the chart will not be generated successfully.

7.9.1. DataTips (ToolTips) Functions

You can create user-defined tooltips using the dataTipJsFunction property. When the mouseover event occurs, the function name which is set to the value of dataTipsFunction property will be called. You can create the JavaScript function which returns the string value that is shown as Tooltips when users move the mouse point over the chart.

The following example shows how to define the dataTipJsFunction property.

```
<KoolChart backgroundColor="0xFFFFEE"   cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Anual Report"/>
        </Options>
        <Column3DChart showDataTips="true" dataTipJsFunction="dataTipFunc">
                <horizontalAxis>
                        <CategoryAxis categoryField="Month" displayName="Date"/>
                </horizontalAxis>
                <verticalAxis>
                        <LinearAxis displayName="Amount"/>
                </verticalAxis>
                <series>
                        <Column3DSeries id="series1" yField="Profit" displayName="Profit">
                                <showDataEffect>
                                        <SeriesInterpolate />
                                </showDataEffect>
                        </Column3DSeries>
                        <Column3DSeries id="series2" yField="Cost" displayName="Cost">
                                <showDataEffect>
                                        <SeriesInterpolate />
                                </showDataEffect>
                        </Column3DSeries>
                        <Column3DSeries id="series3" yField="Revenue" displayName="Revenue">
                                <showDataEffect>
                                        <SeriesInterpolate />
                                </showDataEffect>
                        </Column3DSeries>
                </series>
        </Column3DChart>
</KoolChart>
```

<Example 68 Usage Example: dataTipJsFunction>

The following example shows the user-defined function defined as the value of the dataTipJsFunction property.

```
/*
// Parameters
// seriesId: Series ID
// seriesName: The name of the series, which is set as the value of the displayName property.
// index: The index of the item in the series.
// xName: The value of the displayName property in the horizontalAxis node.
// yName: The value of the displayName property in the verticalAxis node.
// data: The record of the dataset declared for creating the chart.
// values: The value of the item. The value is an array and each element in the array is different depending on the chart type.
                    Bar2D(3D)Series          0: X-axis value, 1: Y-axis value
                    Column2D(3D)Series       0: X-axis value, 1: Y-axis value
                    Area2DSeries             0: X-axis value, 1: Y-axis value
                    Bubble3DSeries           0: X-axis value, 1: Y-axis value 2: radius
                    Line2DSeries             0: X-axis value, 1: Y-axis value
                    Pie2D(3D)Series          0: value, 1: percentage, 2: nameField */

// User-defined tooltips.
function dataTipFunc(seriesId, seriesName, index, xName, yName, data, values)
{
        return "<font color='#CC3300'>User-defined data tips</font>₩nseriesId:"+seriesId
                        +"<br/><font color='#0000FF'>Current Data : </font>"
                        +"<b>"+seriesName+"</b>"+"₩nitemIndex:"+index
                        +"<br/><font color='#0000FF'>xDisplayName : </font>"+"<b><font
size='11'>"+xName+"</font></b>"
                        +"<br/><font color='#0000FF'>yDisplayName : </font>"+"<b><font
size='11'>"+yName+"</font></b>"
                        +"<br/>data:"+data+"<br/>values:"+values;
}
```

<Example 69 Example: JavaScript Function Called for User-defined Tooltips>

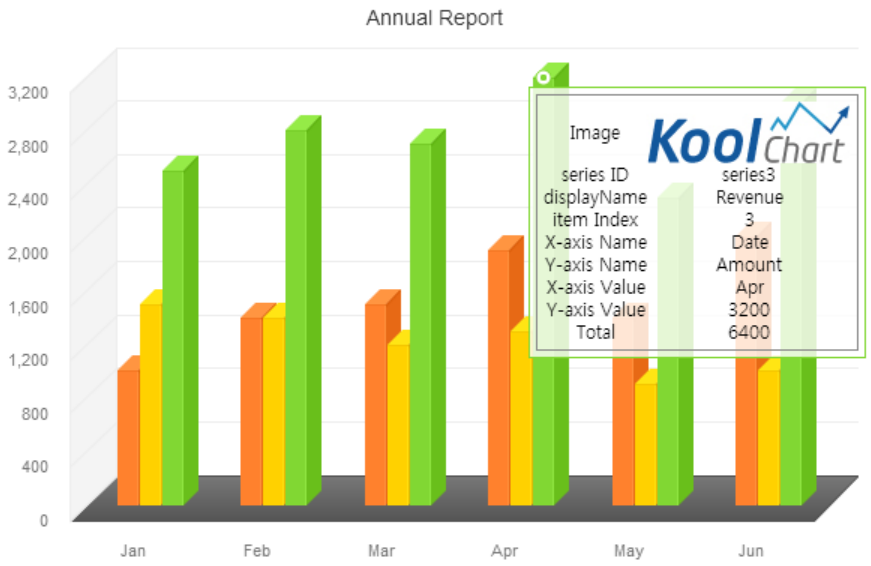
- **HTML tags can be used for user-defined tooltips.**

The following table describes the parameters used in the above example.

Parameters	Descriptions
seriesId	Series ID
seriesName	The name of the series, which is set as the value of the displayName property.
index	The index of the clicked item in the series. (The index of the leftmost item or the top

<table>
<tr><td></td><td>item is 0.)</td></tr>
<tr><td>xName</td><td>The name of the X-axis when “displayName” is defined in the X-axis.</td></tr>
<tr><td>yName</td><td>The name of the Y-axis when “displayName” is defined in the Y-axis</td></tr>
<tr><td>data</td><td>The record of the dataset declared for creating the chart.</td></tr>
<tr><td>values</td><td>The value of the item. This parameter is passed as an array data and the elements of the array are determined based on the type of the chart as follows:
<table>
<tr><td>Bar2DSeries(Bar3DSeries)</td><td>0: X-axis value, 1: Y-axis value</td></tr>
<tr><td>Column2DSeries(Column3DSeries)</td><td>0: X-axis value, 1: Y-axis value</td></tr>
<tr><td>Area2DSeries</td><td>0: X-axis value, 1: Y-axis value</td></tr>
<tr><td>Bubble3DSeries</td><td>0: X-axis value, 1: Y-axis value 2: radius</td></tr>
<tr><td>Line2DSeries</td><td>0: X-axis value, 1: Y-axis value</td></tr>
<tr><td>Pie2DSeries(Pie3DSeries)</td><td>0: value, 1: percentage, 2: nameField</td></tr>
</table></td></tr>
</table>

<Table 25 Parameters: Function Called for Tooltips>



<Figure 43 Output: JavaScript Function Called for User-defined Tooltips>

7.9.2. Axis Labels

You can create the user-defined axis label by using the labelJsFunction property. To create the user-defined axis label, you need to create a callback function and then set the value of the *labelJsFunction* property to the function name. You can define the labelJsFunction property in the axis or the axis renderer. (The function defined in the axis runs first.)

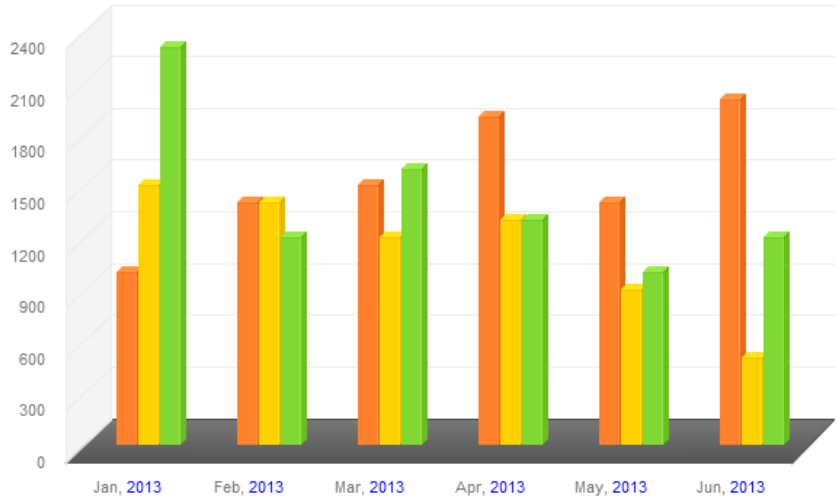
The following example shows how to define the labelJsFunction property in <horizontalAxis>.

```
<KoolChart backgroundColor="0xFFFFEE"   cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Anual Report"/>
        </Options>
        <Column3DChart showDataTips="true">
                <horizontalAxis>
                        <!-- axisLabelFunc is the name of the JavaScript function. -->
                        <CategoryAxis categoryField="Month" displayName="Date"
labelJsFunction="axisLabelFunc"/>
                </horizontalAxis>
                <verticalAxis>
                        <LinearAxis displayName="Amount"/>
                </verticalAxis>
                ...
                ...
                ...
        </Column3DChart>
</KoolChart>
```

<Example 70 Usage Example: labelJsFunction>

```
/*
// ------------------------- A user-defined function defined for the X-axis label -----------------------------
//
// Parameters:
// id: The ID of the axis
// value: The value of the data item
*/
function axisLabelFunc(id, value)
{
        return   value + ", 2013";
}
```

<Example 71 Example: JavaScript Function Called for User-defined Label>



<Figure 44 Output: JavaScript Function Called for User-defined Label>

7.9.3. Numeric Values

If you want to create the user-defined numeric value in Column, Bar or Pie charts, you need to define the labelJsFunction property in <series>.

The following example shows how to define the labelJsFunction property to create the user-defined numeric value.

```
<series>
        <!-- seriesLabelFunc is a name of the JavaScript function -->
        <Column2DSeries yField="Profit" displayName="Profit" labelPosition="outside" styleName="seriesStyle"
outsideLabelJsFunction="seriesLabelFunc">
                <showDataEffect>
                        <SeriesInterpolate/>
                </showDataEffect>
        </Column2DSeries>
</series>
```

<Example 72 Usage Example: labelJsFunction for User-defined Numeric Value>

The following example shows how to set the user-defined function (seriesLabelFunc) to display numeric values.

```
/*
// A user-defined function for numeric data fields.
//
// Parameters
// seriesId: Series ID
// index: The index of the item in the series.
```

```
// data: The record of the dataset declared for creating the chart.
// values: The value of the item. The value is an array and each element in the array is different depending on the chart type.
                Bar2D(3D)Series         0: X-axis value, 1: Y-axis value
                Column2D(3D)Series    0: X-axis value, 1: Y-axis value
                Pie2D(3D)Series         0: X-axis value, 1: Y-axis value 2: radius
                Bubble3DSeries         0: value, 1: percentage

// Important: HTML tags can not be used in labelJsFunction.
*/
function seriesLabelFunc(seriesId, index, data, values)
{
        return "$"+values[1];
}
```

<Example 73 Example: JavaScript Function Called for User-define Numeric Value>

Series	labelJsFunction
Column2DSeries Column3DSeries, Bar2DSeries, Bar3DSeries	insideLabelJsFunction, outsideLabelJsFunction
Line2DSeries, Area2DSeries	upLabelJsFunction, downLabelJsFunction
Bubble3DSeries	insideLabelJsFunction
Pie2DSeries, Pie3DSeries,	labelJsFunction

<Table 26 labelJsFunction >

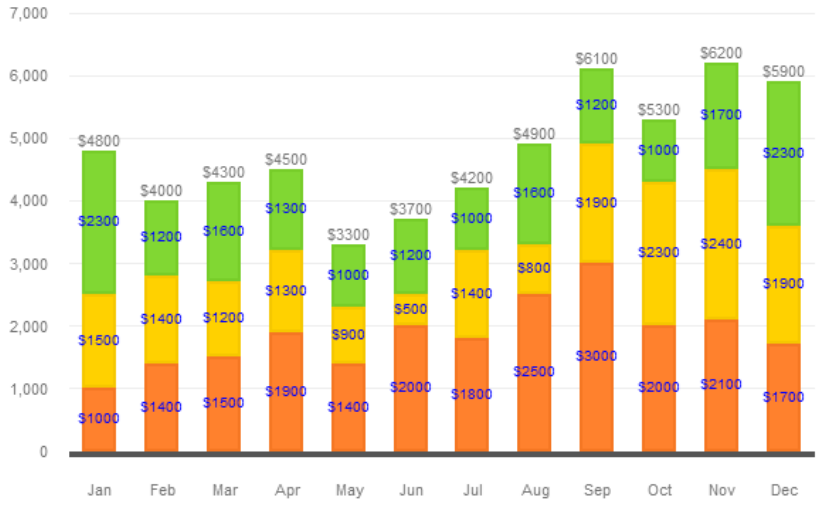
labelJsFunction is related with labelPosition.

- **HTML tags can not be used for labelJsFunction.**

The following table describes the parameters used in the above example.

<table>
<tr><th>Parameters</th><th>Descriptions</th></tr>
<tr><td>seriesId</td><td>Series ID</td></tr>
<tr><td>index</td><td>The index of the data item. (The index of the leftmost item or the top item is 0).</td></tr>
<tr><td>data</td><td>The record of the dataset declared for creating the chart.</td></tr>
<tr><td>values</td><td>The value of the item. This parameter is passed as an array data and the elements of the array are determined based on the type of the chart as follows:
<table>
<tr><td>BarSeries(Bar3DSeries)</td><td>0: X-axis value, 1: Y-axis value</td></tr>
<tr><td>ColumnSeries(Column3DSeries)</td><td>0: X-axis value, 1: Y-axis value</td></tr>
<tr><td>Bubble3DSeries</td><td>0: X-axis value, 1: Y-axis value 2: radius</td></tr>
<tr><td>Pie2DSeries(Pie3DSeries)</td><td>0: value, 1: percentage</td></tr>
</table></td></tr>
</table>

<Table 27 Parameters: JavaScript Function Called for User-defined Numeric Value>



<Figure 45 Output: JavaScript Function Called for User-defined Numeric Value>

7.9.4. Filling Colors

If you want to fill the data item with the user-defined color based on the numeric value, you need to define the fillJsFunction property in <series>.

The following is an example layout to define the fillJsFunction property in the Column 3D chart.

```
<Column3DChart showDataTips="true" gutterLeft="0" showLabelVertically="true">
                <series>
                        <Column3DSeries yField="Profit"
```

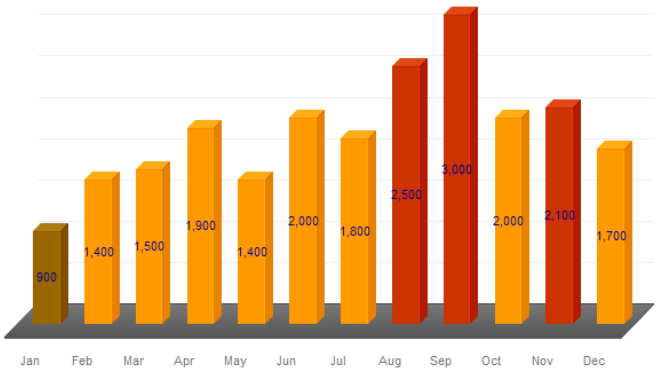
```
                    fillJsFunction="fillJsFunc"   styleName="seriesStyle">
                        <showDataEffect>
                            <SeriesSlide direction="up" duration="1000"/>
                        </showDataEffect>
                    </Column3DSeries>
                </series>
...
...
...
</Column3DChart>
```

<*Example 74 Usage Example: fillJsFunction*>

The following example shows the user-defined function (fillJsFunc) set for fillJsFunction.

```
/*
//-------------------- A user-defined function for fillJsFunction ----------------------------------
//
// Parameters:
// seriesId: Series ID
// index: The index of the item in the series.
// data: The record of the dataset declared for creating the chart.
// values: The value is an array and each element in the array is different depending on the chart type.
                Bar2D(3D)Series         0: X-axis value, 1: Y-axis value
                Column2D(3D)Series      0: X-axis value, 1: Y-axis value
                Area2DSeries            0: X-axis value, 1: Y-axis value
                Bubble3DSeries          0: X-axis value, 1: Y-axis value 2: radius
                Line2DSeries            0: X-axis value, 1: Y-axis value
                Pie2D(3D)Series         0: value, 1: percentage, 2: nameField

// If you define minField in the From-To chart, the last index of the value is minField. */
function fillJsFunc(seriesId, index, data, values)
{
        if(values[1] > 2000)
                return "0xFF3366";
        else if(values[1] > 1000)
                return "0xFFFF33";
        else
                return "0xFF9999";
}
```

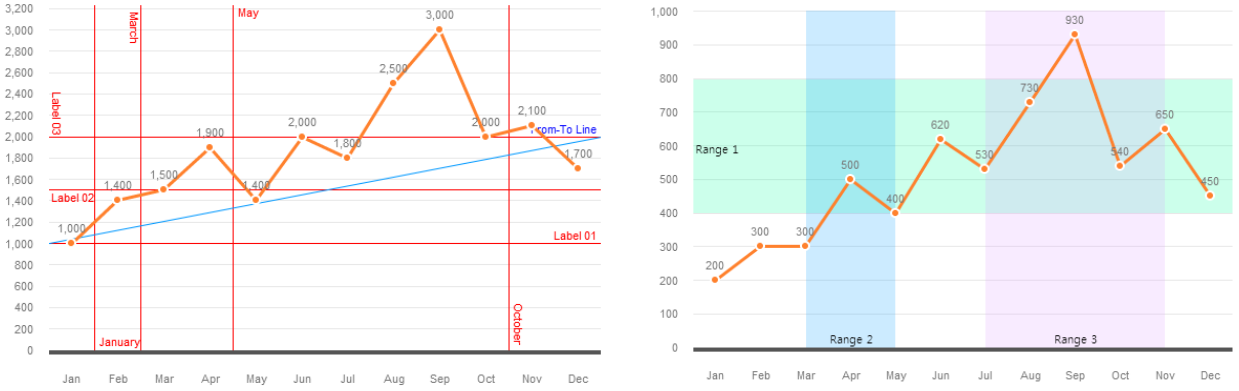
<*Example 75 Example: JavaScript Function Called for fillJsFunction*>



<Figure 46 Output: JavaScript Function Called for fillJsFunction>

7.10. Creating Areas and Lines

You can create the upper and lower limit line or the data range as the following examples show:



<Figure 47 Outputs: Charts with Lines and Data Ranges>

Parameters	Descriptions
The properties of <backgroundElements> or <annotationElements>	Displays the data ranges or lines in the background (backgroundElements) or foreground (annotationElements).
AxisMarker	Displays AxisLine and AxisRange.
AxisLine	Line
AxisRange	Range

The following layout shows how to display the data range and line.

```
<KoolChart backgroundColor="0xFFFFFF">
        <Stroke id="stroke1" color="0xFF0000" weight="1"/>
        <Line2DChart showDataTips="true">

                ......
                  // Displays the data range and line in the background of the chart.
        <backgroundElements>
                <GridLines/>   // Defines <GridLines/>
                <AxisMarker> // Creates <AxisMarker> to draw the data range and line

                <lines> // Draws the line
                // Draws three lines by creating three <AxisLines>.
                <AxisLine value="1000" label="Label 01" stroke="{stroke1}" labelUpDown="up"
color="0xFF0000"/>

                <AxisLine value="1500" label="Label 02" stroke="{stroke1}" labelAlign="left"
labelUpDown="down" color="0xFF0000"/>

                <AxisLine value="2000" label="Label 03" stroke="{stroke1}" labelUpDown="up" labelAlign="left"
labelRotation="90" color="0xFF0000"/>
                </lines>

                <ranges> // Draws the data range.
                // Draws two data ranges by creating two <AxisRanges>
                <AxisRange startValue="1000" endValue="2000" label="Range 1" fontSize="11"
                labelHorizontalAlign="left" color="0xFF00FF">
                        <fill>
                                <SolidColor color="0x00FF99" alpha="0.2"/>
                        </fill>
                </AxisRange>
                <AxisRange startValue="Mar" endValue="May" label="Range 2" fontSize="11"
labelVerticalAlign="bottom" color="0x0066FF" horizontal="false">
                        <fill>
                                <SolidColor color="0x0099FF" alpha="0.2"/>
                        </fill>
                </AxisRange>
                </ranges>

                </AxisMarker>
                </backgroundElements>
        </Line2DChart>
</KoolChart>
```

<Example 76 Example: Lines and Ranges>

The following table shows the properties of the data range and line.

Properties	Values	Descriptions
<lines>	<AxisLine>	Defines <lines> to draw the line in charts.
<ranges>	<AxisRange>	Defines <ranges> to draw the data range.

<Table 28 <AxisLine> and <AxisRange> >

All the properties of <Caption> can be used in <AxisRange> and <AxisLine>. The following table shows the properties of <AxisRange> and <AxisLine>. (The properties of <Caption> are not shown in the following table. Please refer to <5.3.1 Creating <Caption> and <SubCaption> for the properties of <Caption>.)

AxisLine -

Properties	Values	Descriptions
labelAlign	center,left,right (default: right)	Specifies the horizontal alignment of the label. If the value of the *horizontal* property is false, "left" is the same to "top", and "right" is the same to "bottom".
labelUpDown	up,down (default: up)	Specifies the position from the line where the label is placed. If the value of the *horizontal* property is false, "up" is the same to "right" and "down" is the same to "left".
labelRotation	0~360 (default: 0)	Rotates the label. If the embeded fonts are used, then any value of 0~360 can be available, but if the system fonts are used, then only 0 is available.
linePosition	center, left, right	Specifies the position where the line is displayed for <CategoryAxis>. In <CategoryAxis>, multiple positions are allowed for one value.
stroke	Stroke	Defines the style of the line.
startValue	Number,String	If you want to draw the diagonal line, you need to specify the position where the line starts. The *startValue* property is ignored when the *value* property is set.
endValue	Number,String	If you want to draw the diagonal line, you need to specify the position where the line ends. The *endValue* property is ignored when the *value* property is set.
Value	Number,String	If you want to draw the normal line (not diagonal

		line), you need to specify the position of the line.
label	String	Specifies the text displayed on the line.

<Table 29 Properties: AxisLine>

AxisRange -

Properties	Values	Descriptions
labelHorizontalAlign	String (center,left,right)	Specifies the horizontal alignment of the label.
labelVerticalAlign	String (top,middle,bottom)	Specifies the vertical alignment of the label.
labelRotation	Number	Rotates the label. Any value of 0~360 can be available for the embedded fonts, but only 0 or 90 are available for the system fonts.
fill	Uint	Specifies the colors to fill the data range.
startValue	Number, String	Specifies the position where the data range starts.
endValue	Number, String	Specifies the position where the data range ends.
horizontal	Boolean (true, false)	Whether or not to draw the horizontal data range. true: Draws the data range using the Y-axis values. false: Draws the data range using the X-axis values.
label	String	Specifies the text displayed in lines.

<Table 30 Properties: AxisRange>

7.11. Zooming and Showing Crosshairs

The zooming feature enables you to enlarge the certain area of the chart, and displaying the mouse pointer as crosshairs is a convenient way to locate a particular point on the chart.

The zooming feature and displaying crosshairs can be used for all charts except the following charts:

- Radar Chart
- Pie Chart, Doughnut Chart
- History Chart, Scroll Chart

- Gauge Chart

<CrossRangeZoomer> needs to be defined to use the zooming feature and to display crosshairs.
The following example shows how to define <CrossRangeZoomer>. <CrossRangeZoomer> should be defined as a child of <annotationElements>.

```
<KoolChart>
<Combination2DChart showDataTips="true">
  <series>
    <Line2DSeries yField="Profit" displayName="Profit"/>
  </series>
  <annotationElements>
    <CrossRangeZoomer zoomType="horizontal" fontSize="11" color="0x00000"
      verticalLabelPlacement="bottom" enableZooming="true" enableCrossHair="true">
      <zoomRangeStroke>
        <Stroke weight="1" color="0xFF0000" alpha="0.3"/>
      </zoomRangeStroke>
      <zoomRangeFill>
        <SolidColor color="0x0000FF" alpha="0.2"/>
      </zoomRangeFill>
    </CrossRangeZoomer>
  </annotationElements>
</Combination2DChart>
</KoolChart>
```

<Example 77 Example: Zooming and Crosshairs>

Properties	Values	Descriptions
horizontalStroke	Stroke	Specifies the horizontal line of crosshairs.
verticalStroke	Stroke	Specifies the vertical line of crosshairs.
zoomRangeStroke	Stroke	Specifies the line style for the zoom area.
zoomRangeFill	SolidColor	Specifies the fill color for the zoom area.
horizontalLabelFormatter	Formatter id	Specifies the formatter for the label of the horizontal line of crosshairs.
horizontalLabelOppFormatter	Formatter id	Specifies the formatter for the right side label of the horizontal line of crosshairs. (It is valid only if useDualCrossXLabel = true)
horizontalLabelPlacement	left| right (default: left)	Specifies the position for the label of the horizontal line of crosshairs. left: Displays the X, Y coordinates on the left side

		of the chart. right: Displays the X, Y coordinates on the right side of the chart.
showValueLabels	true| false (default: true)	Whether or not to display the X, Y coordinates. (The current position of the mouse point.)
verticalLabelFormatter	Formatter id	Specifies the formatter for the label of the vertical line of crosshairs.
verticalLabelOppFormatter	Formatter id	Specifies the formatter for the top side label of the vertical line of crosshairs. (It is valid only if useDualCrossXLabel = true)
verticalLabelPlacement	top|bottom (default: bottom)	Specifies the position for the label of the vertical line of crosshairs. top: Displays the X, Y coordinates on the top side of the chart. bottom: Displays the X, Y coordinates on the bottom side of the chart.
useDualCrossXLabel	false|true (default: false)	Whether or not to display two axes labels if more than two vertical axes are defined. (If useDualCrossXLabel="true" horozontalLabelPlacement will be ignored) For example, if three vertical axes are defined, the labels of the first two axes will be displayed
useDualCrossYLabel	false|true (default: false)	Whether or not to display two axes labels if more than two horizontal axes are defined. (If useDualCrossYLabel="true" verticalLabelPlacement will be ignored) For example, if three horizontal axes are defined, the labels of the first two axes will be displayed.
enableCrossHair		Whether or not to use crosshairs.
enableZooming		Whether or not to use the zooming feature.
zoomType	horizontal, vertical, both (default: both)	Specifies the zooming axis if enableZooming = "true". For example, if zoomType="horizontal", only the horizontal axis will be zoomed in.
resetMode	initial, auto(default: auto)	Specifies the restore method if enableZooming = "true". initial: uses the minimum and maximum value defined by users.

		auto: the minimum and maximum values are determined by system automatically.

<Table 31 Properties: Zoom and Crosshairs >

7.12. Adding Memos

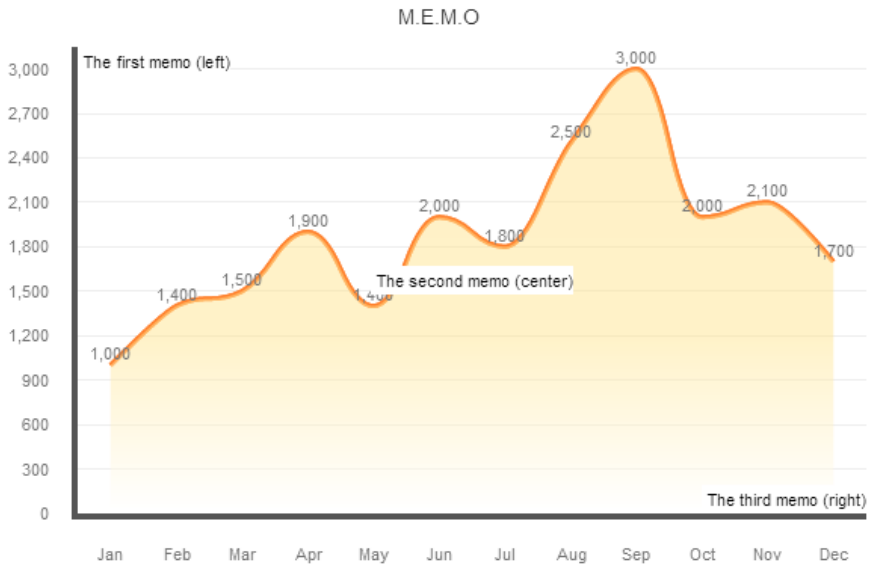
You may want to add the text annotation to the chart. The annotation is placed based on the X, Y coordinates of the chart rather than the item to which you want to add the annotation. You can display the annotation either on the top of the chart using <annotationElements> or at the background of the chart using <backgroudElements>.

The following example shows how to write memos at the top of the chart.

```
<KoolChart>
  <Line2DChart showDataTips="true">
    <series>
    .....
    </series>
    <annotationElements>

      <CanvasElement>
        <!-- label 1 -->
        <Label left="0" text="The first memo (left)" fontSize="11"  borderColor="0x000000" >
        </Label>
        <!-- label 2 -->
        <Label right="0" text="The third memo (right)" fontSize="11" borderColor="0x000000">
        </Label>
        <!-- label 3 -->
        <Label horizontalCenter="0" verticalCenter="0" text=" The second memo (center)" fontSize="11"
        borderColor="0x000000">
        </Label>
      </CanvasElement>
    </annotationElements>
  </Line2DChart>
</KoolChart>
```

<Example 78 Example: Memo>

The following is the output produced by the layout above.



<Figure 48 Output: Adding Memo>

- For further information of the properties of <Label>, please refer to <5.3.1 Creating <Caption> and <SubCaption>.

The following table shows the properties of <Label>.

Properties	Values	Descriptions
backgroundColor	RGB	Specifies the background color.
backgroundAlpha	0~1	Specifies the transparency of the background.
borderColor	RGB	Specifies the border color.
borderAlpha	0~1	Specifies the transparency of the border.
borderThickness	Number (pixel)	Specifies the border thickness.
cornerRadius	Number	Specifies the roundness of the corner.
borderStyle	solid, none	Whether or not to display the border line.

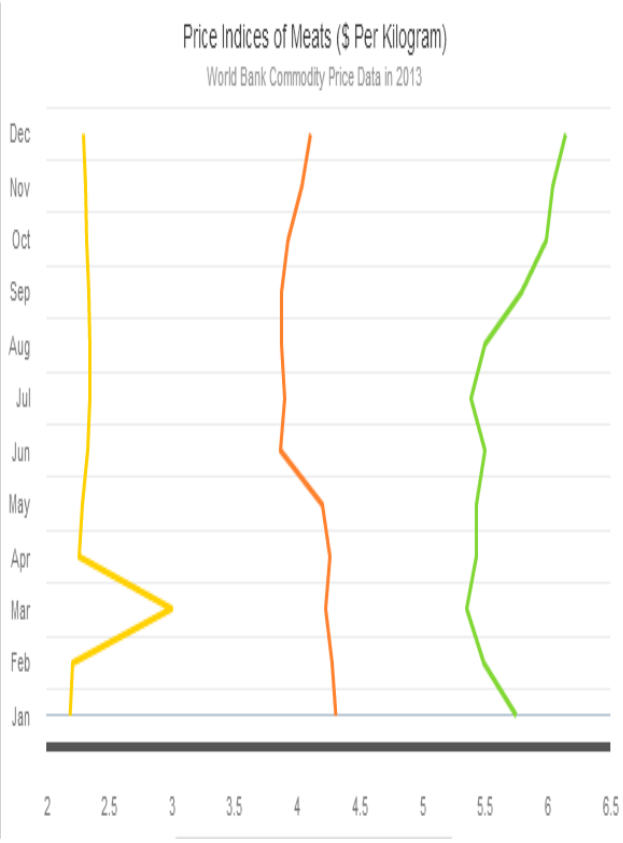
<Table 32 Properties: <Label>>

7.13. Using Vertical Lines in Line Charts

Normally, the Line chart displays the line along the horizontal axis, but if you want to display the line vertically rather than horizontally, you must do the followings:

- Create <CategoryAxis> as a child of <verticalAxis>.
- Set the value of the xField property to the numeric field (value), and set the value of the yField property to the category field.
- Set the value of <sortOnXField> to false.

```
<KoolChart>
  <Line2DChart>
    <verticalAxis>
        <CategoryAxis categoryField="Month"/>
    </verticalAxis>
    <series>
        <Line2DSeries xField="Profits" yField="Month"
sortOnXField="false" displayName="Profits">
        </Line2DSeries>
        <Line2DSeries xField="Costs" yField="Month"
sortOnXField="false" displayName="Costs">
        </Line2DSeries>
         <Line2DSeries xField="Revenue"
yField="Month" sortOnXField="false"
displayName="Revenue">
         </Line2DSeries>
        </series>
  </Line2DChart>
</KoolChart>
```



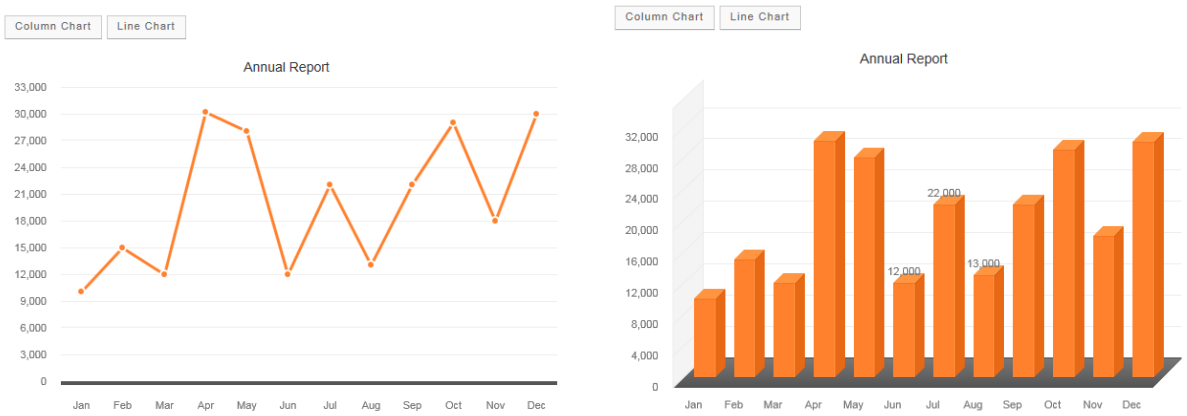
< Example 79 Example: Drawing Line along Y-Axis>

7.14. Changing Layouts and Data Dynamically

You may want to change data in the chart dynamically or convert the type of the chart to the different type. KoolChart supports changing the layout or data dynamically without refreshing the web page.

The following example shows a Line chart which can be converted to a column chart by clicking the button on the top of the Line chart and vice versa. KoolChart recommends that you include

KoolIntegration.js in the HTML page when you change the type of the chart dynamically. As KoolIntegration.js can generate all the types of charts, you don't have to worry about if the chart generated dynamically is supported by the KoolChart JavaScript file that is included in the HTML page.



< Figure 49 Output: Dynamic Changes>

The following HTML is used to generate the above outputs.

```
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script src="KoolIntegration.js" language="javascript"></script>
<script language="JavaScript" type="text/javascript">

var chartVars = "&KoolOnLoadCallFunction=KoolChartOnLoad";

KoolChart.create("chart1", "chartHolder", chartVars, "100%", "100%");

function KoolChartOnLoad()
{
        chart.setLayout(layoutStr);
        chart.setData(chartData);
}

var layoutURL = encodeURIComponent("chartLayout.xml");
var layoutStr = "<KoolChart cornerRadius='12' borderStyle='solid'>"
        +"<Options><Caption text='Annual Report'/></Options>"
        +"<Line2DChart showDataTips='true'>"
        +"<horizontalAxis><CategoryAxis categoryField='Month'/></horizontalAxis>"
        +"<series><Line2DSeries yField='Profit' displayName='Profit'>"
        +"<showDataEffect><SeriesInterpolate/></showDataEffect>"
        +"<lineStroke><Stroke color='0xFF0000'weight='4'/>"
```

```
            +"</lineStroke></Line2DSeries>"
             +"</series></Line2DChart></KoolChart>";

var chartData = [{"Month":"Jan", "Revenue":10000, "Cost":5000, "Profit":5000},
            {"Month":"Feb", "Revenue":15000, "Cost":7000, "Profit":8000},
            {"Month":"Mar", "Revenue":12000 , "Cost":6000, "Profit":6000},
            {"Month":"Apr", "Revenue":30200, "Cost":4000, "Profit":26200},
            {"Month":"May", "Revenue":28000, "Cost":10000, "Profit":18000},
            {"Month":"Jun", "Revenue":12000, "Cost":5000, "Profit":7000},
            {"Month":"Jul", "Revenue":22000, "Cost":10000, "Profit":12000},
            {"Month":"Aug", "Revenue":13000, "Cost":6000, "Profit":7000},
            {"Month":"Sep", "Revenue":22000, "Cost":10000, "Profit":12000},
            {"Month":"Oct", "Revenue":29000, "Cost":8000, "Profit":21000},
            {"Month":"Nov", "Revenue":18000, "Cost":7500, "Profit":10500},
            {"Month":"Dec", "Revenue":30000, "Cost":12000, "Profit":18000} ];

var chartData2 = [{"Month":"Jan", "Revenue":1000, "Cost":500, "Profit":500},
            {"Month":"Feb", "Revenue":1500, "Cost":700, "Profit":800},
            {"Month":"Mar", "Revenue":1200 , "Cost":600, "Profit":600},
            {"Month":"Apr", "Revenue":3020, "Cost":400, "Profit":2620},
            {"Month":"May", "Revenue":2800, "Cost":1000, "Profit":1800},
            {"Month":"Jun", "Revenue":1200, "Cost":500, "Profit":700},
            {"Month":"Jul", "Revenue":2200, "Cost":1000, "Profit":1200},
            {"Month":"Aug", "Revenue":1300, "Cost":600, "Profit":700},
            {"Month":"Sep", "Revenue":2200, "Cost":1000, "Profit":1200},
            {"Month":"Oct", "Revenue":2900, "Cost":800, "Profit":2100},
            {"Month":"Nov", "Revenue":1800, "Cost":750, "Profit":1050},
            {"Month":"Dec", "Revenue":3000, "Cost":1200, "Profit":1800} ];

// Change chart type
function changeLayout()
{
    document.getElementById("chart1").setLayoutURL(layoutURL);
}

function changeLayout3()
{
    document.getElementById("chart1").setLayoutURL(layoutStr);
}

</script>
</head>

<body>
```

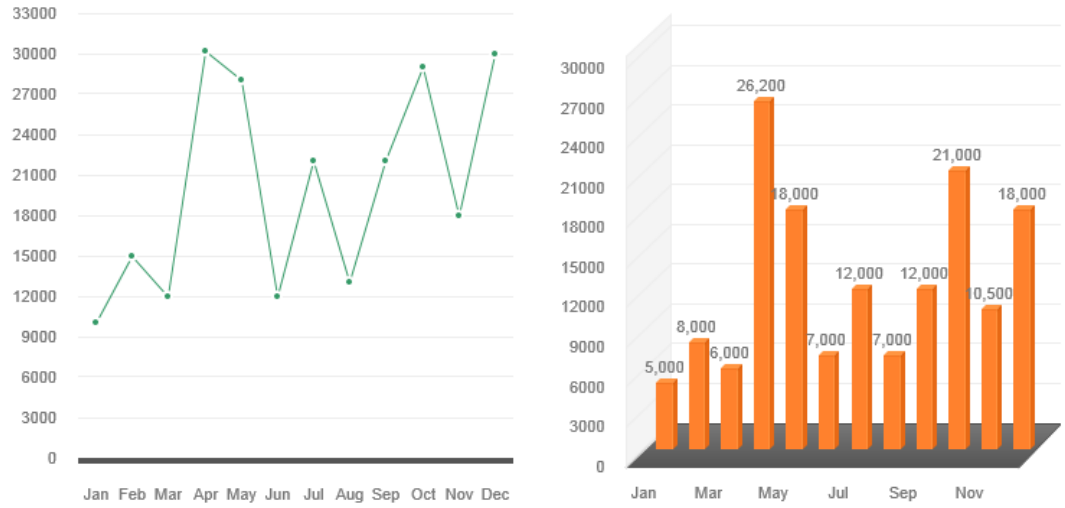
```
        <div id="header">
                <div class="button button_top" onclick="changeLayout()">Column Chart</div>
                <div class="button button_top" onclick="changeLayout3()">Line Chart</div>
        </div>
        <div id="content">
                <!-- The DIV in which the chart is placed. -->
                <div id="chartHolder" style="width:600px; height:400px;">
                </div>
        </div>
</body>
</html>
```

<Example 80 Example: Dynamic Changes>

7.15. Creating Multiple Charts in an HTML Page

You can create multiple charts in an HTML page.

The following example shows how to create two charts in an HTML page.



<Figure 50 Outputs: Two Charts in an HTML Page>

The following example shows how to create two charts in an HTML page.

```
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script src="KoolChart.js" language="javascript"></script>
```

```
<script language="JavaScript" type="text/javascript">

// ----------------------Sets chartVars for the first chart ----------------------
var chartVars = "&KoolOnLoadCallFunction=KoolChartOnLoad";

// ----------------------Sets chartVars for the second chart ----------------------
var chartVars2 = "&KoolOnLoadCallFunction=KoolChartOnLoad2";

// creates KoolChart.
KoolChart.create("chart1", "chartHolder", chartVars, "100%", "100%");
KoolChart.create("chart2", "chartHolder2", chartVars2, "100%", "100%");

function KoolChartOnLoad()
{
	chart1.setLayout(layoutStr);
	chart1.setData(chartData);
}

// This function is called when the second chart is created. (user-defined function)
function KoolChartOnLoad2()
{
	chart2.setLayoutURL(layoutURL2);
	chart2.setData(chartData2);
}

// Layouts used dynamically.
var layoutURL2 = encodeURIComponent("chartLayout.xml");
var layoutStr = "<KoolChart cornerRadius='12' borderStyle='solid'>"
	+"<Options><Caption text='Annual Report'/></Options>"
	+"<Line2DChart showDataTips='true'>"
	+"<horizontalAxis><CategoryAxis categoryField='Month'/></horizontalAxis>"
	+"<series><Line2DSeries yField='Profit' displayName='Profit'>"
	+"<showDataEffect><SeriesInterpolate/></showDataEffect>"
	+"<lineStroke><Stroke color='0xFF0000'weight='4'/>"
	+"</lineStroke></Line2DSeries>"
	+"</series></Line2DChart></KoolChart>";

// Data used dynamically.
var chartData = [{"Month":"Jan", "Revenue":10000, "Cost":5000, "Profit":5000},
	{"Month":"Feb", "Revenue":15000, "Cost":7000, "Profit":8000},
	{"Month":"Mar", "Revenue":12000 , "Cost":6000, "Profit":6000},
	{"Month":"Apr", "Revenue":30200, "Cost":4000, "Profit":26200},
	{"Month":"May", "Revenue":28000, "Cost":10000, "Profit":18000},
	{"Month":"Jun", "Revenue":12000, "Cost":5000, "Profit":7000},
```

```
            {"Month":"Jul", "Revenue":22000, "Cost":10000, "Profit":12000},
            {"Month":"Aug", "Revenue":13000, "Cost":6000, "Profit":7000},
            {"Month":"Sep", "Revenue":22000, "Cost":10000, "Profit":12000},
            {"Month":"Oct", "Revenue":29000, "Cost":8000, "Profit":21000},
            {"Month":"Nov", "Revenue":18000, "Cost":7500, "Profit":10500},
            {"Month":"Dec", "Revenue":30000, "Cost":12000, "Profit":18000} ];

var chartData2 = [{"Month":"Jan", "Revenue":1000, "Cost":500, "Profit":500},
            {"Month":"Feb", "Revenue":1500, "Cost":700, "Profit":800},
            {"Month":"Mar", "Revenue":1200 , "Cost":600, "Profit":600},
            {"Month":"Apr", "Revenue":3020, "Cost":400, "Profit":2620},
            {"Month":"May", "Revenue":2800, "Cost":1000, "Profit":1800},
            {"Month":"Jun", "Revenue":1200, "Cost":500, "Profit":700},
            {"Month":"Jul", "Revenue":2200, "Cost":1000, "Profit":1200},
            {"Month":"Aug", "Revenue":1300, "Cost":600, "Profit":700},
            {"Month":"Sep", "Revenue":2200, "Cost":1000, "Profit":1200},
            {"Month":"Oct", "Revenue":2900, "Cost":800, "Profit":2100},
            {"Month":"Nov", "Revenue":1800, "Cost":750, "Profit":1050},
            {"Month":"Dec", "Revenue":3000, "Cost":1200, "Profit":1800} ];

</script>
</head>

<body>
<table align="center" border="0" cellpadding="0" cellspacing="0">
        <tr>
        <td align="center">
                <div class="content">
                        <div id="chartHolder" style="width:600px; height:400px;">
                        </div>
                </div>
        </td>
        <td align="center">
                <div class="content">
                        <div id="chartHolder2" style="width:600px; height:400px;">
                        </div>
                </div>
        </td>
</tr>
</table>
</body>
</html>
```

<Example 81 Example: Two Charts in a Single HTML File>

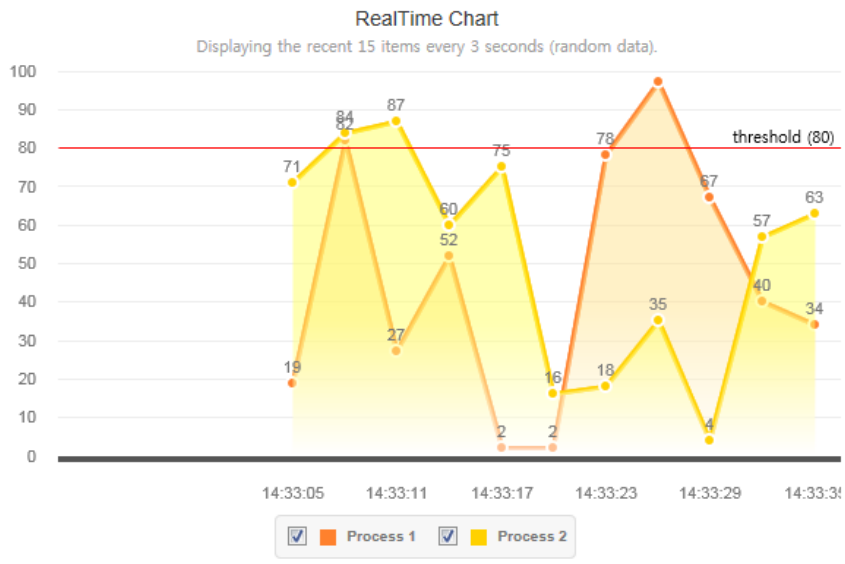
7.16. Real-Time Chart Example – A Stock Monitoring Chart

The Real-time chart is widely used for monitoring the variation of the stock price. The following example shows how to create a Real-time chart to monitor the stock price.

```
<KoolChart backgroundColor="0xFFFFEE" cornerRadius="12" borderStyle="solid">
        <Options>
                <Caption text="Stock Monitoring" />
                <SubCaption text=" Refresh every 3 seconds.(Random data)" fontSize="11" textAlign="right"/>
                <Legend/>
        </Options>
        <DateFormatter id="dateFmt" formatString="HH:NN:SS" /> //Defines formatter

       // Type : time, Refresh : every 3 seconds, Duration : 60 seconds
        <RealTimeChart id="chart" dataDisplayType="time"   timePeriod="60" interval="3" showDataTips="true">

                <backgroundElements>
                        <GridLines direction="both"/>
                </backgroundElements>

            // DateTime is used for the horizontal axis
            // The measure of data is second
            // Data refresh interval is 3 seconds
            // The interval of axis label is 9 deconds
            // The local time is used (not GMT)
                <horizontalAxis>
                          <DateTimeAxis id="hAxis" dataUnits="seconds" labelUnits="seconds" dataInterval="3"
interval="9" formatter="{dateFmt}" displayLocalTime="true"/>
                </horizontalAxis>
                <series>
                        <Column2DSeries yField="Volume" xField="Time"
                                displayName="Trading Volume"
                        itemRenderer="GradientColumnItemRenderer">
                                <fill>
                                        <SolidColor color="0xB0C759"/>
                                </fill>
                                <verticalAxis>
                                   <LinearAxis id="vAxis1"   title="Volume" minimum="0"
maximum="10000"/>
                                </verticalAxis>
                        </Column2DSeries>

                        <Line2DSeries xField="Time" yField="Price" displayName="Stock Price"
```

```
                        itemRenderer="CircleItemRenderer"   radius="5" fill="0xFFFF00">
                            <stroke>
                                <Stroke color="0xE48701" weight="2"/>
                            </stroke>
                            <lineStroke>
                                <Stroke color="0xE48701" weight="4"/>
                            </lineStroke>
                            <verticalAxis>
                                <LinearAxis id="vAxis2" title="Price" minimum="0"
maximum="1000"/>
                            </verticalAxis>
                        </Line2DSeries>
                    </series>
                    <verticalAxisRenderers>
                        <Axis2DRenderer placement="left" axis="{vAxis1}">
                        </Axis2DRenderer>
                        <Axis2DRenderer placement="right" axis="{vAxis2}">
                        </Axis2DRenderer>
                </verticalAxisRenderers>
            </RealTimeChart>
            // Defines RPC to refresh data.
            <HttpServiceRepeater url="http://www.koolchart.com/realtimeSample/data4.php" target="{chart}"
interval="3"   method="get"/>
</KoolChart>
```

<*Example 82 Example: Monitor Stocks*>



<*Figure 51 Output: Real-time Chart for Stock Monitoring*>

7.17. Exporting Charts as Image

You can export the chart as an image file that is encoded in Base64 format. KoolChart supports the chart exporting functionality by using the toDataURL() method which is provided by HTML5 Canvas. **As the toDataURL() method is not supported in excanvas.js, you cannot use IE7 and IE8 for exporting the chart.** The following example shows how to call the saveAsImage() method which is a KoolChart-provided method to export the chart as an image data encoded in Base64 format.

```
function snapShot(){
        var base64src = document.getElementById("chart1").saveAsImage();
        document.getElementById("img").src = base64src;
}
```

The saveAsImage() method is a KoolChart-provided function called by accessing (calling getElementById()) the chart id ("chart1" in the example above). When the first statement is executed, the image data encoded in Base64 format is saved to the variable, base64src. As JavaScript cannot save a file to the local PC, we provide an example that opens a new popup window that displays the chart image. You can save the image file to the local PC by clicking the right mouse button. Please refer to the example (Chart Properties → Export Chart → Save as Image).

7.18. Sending Chart Image to Server

You can send the chart image to the server to save it to your server disk or to create a new image file with the chart image. The following example is a JavaScript function for sending the chart image to the server.

```
// JavaScript function: generating the Base64 formatted data and submitting it to the server.
function getSanpshot(){
        // generating the Base64 formatted data.
        var base64src = document.getElementById("chart1").getSnapshot();
        var data = document.getElementById("data");
        var extension = document.getElementById("extension");
        // extension (image file)
        extension.value = "png";
        // image source
        data.value = base64src
        // when the image is submitted, the server-side script needs to decode the image and save it to the server disk.
        document.getElementById("sumForm").submit();
}
```

In the above JavaScript function, the getSnapshot() method is called to generate the Base64 formatted data which is sent to the server. As the getSnapshot() method also uses the toDataURL() method of HTML5 Canvas, this functionality cannot be used in IE7 and IE8. In order to use the above function, the HTML Form script and the server-side script need to be written. Please refer to the example (Chart Properties → Export Chart → Save as Image).

7.19. Exporting Charts in Mobile

Exporting the chart image is supported in HTML5-supported browsers, so that you can use this functionality in your mobile devices. **The default browser of Android 2.3 (Gingerbread) or below cannot be used to export the chart image, but you can install other browsers that support exporting the chart image. This functionlioty generally works well in Android 4.0 (Ice Cream Sandwitch) and above.** Please refer to the example, http://m.koolchart.com/Samples/GetSnapshot.html

8. For Visually Impaired Users

8.1. Text Substitution

KoolChart provides the text substitution functionality for visually impaired users. If you add data for text substitution when you create a chart, then KoolChart will save the data, and the data can be read by using devices or applications that support reading the substitute text.

KoolChart does not provide any functionality or device to read the substitute text.

Please set the variable, chartVars, as follows:

```
chartVars += "&accessibility=true";
```

8.2. Patterns

As visually impaired people can hardly recognize the color of the chart, KoolChart version 3.0 has a new capability that displays patterns instead of colors in the chart. Twenty different types of patterns are supported, and the patterns can be modified by users.

```
// Adds the string, "usePattern=true" to the variable, chartVars
chartVars += "&usePattern=true";
```

If you add the string, "usePattern=true", to the variable, chartVars, KoolChart will load the image files corresponding to the patterns. You can modify the URL of the image files.

```
// The URL of the directory
KoolChart.patternImageBaseUrl = "../Images/pattern/";

// The url of the image files
KoolChart.patternImageUrl = [
    "diagonal_ltr.png",
    "diagonal_rtl.png",
    "diagonal.png"
    ....
```

```
];
```

The above are the default values.

Once the pattern is applied, you can also unapply the pattern.

```
document.getElementById("chart1").accessibilityPattern(true / false);
```

You can call the accessibilityPattern method to apply (or unapply) the pattern.
(Apply: true, Unapply: false)

9. Using Themes

9.1. Registering Themes

To register the themes, please include theme.js in the HTML file.

```
// KoolChart Theme
// theme.js is placed in the directory, Samples.
// theme.js must be included after the KoolChart library (e.g. KoolChart.js)
<script type="text/javascript"  src="./theme.js"></script>

// The KoolChart.themes variable is defined in theme.js
KoolChart.registerTheme(KoolChart.themes);
```

9.2. Applying Themes

Applies the simple theme which is one of theme provided in theme.js (simple, cyber, modern, lovely, pastel, old).

```
// Runs the setTheme method with the parameter, "simple"
document.getElementById("chart1").setTheme("simple");
```

9.3. Back to Default Theme

After appling the theme, if you want to go back to the default theme, then apply the default theme.

```
// Runs the setTheme method with the parameter, "default"
document.getElementById("chart1").setTheme("default");
```

9.4. Removing Themes

You can remove a certain theme from the registered theme. When you remove a theme, all the charts which are applied by the theme you are removing will be applied by the default theme.

```
// To remove the simple theme, runs the removeTheme method with the parameter, "simple"
KoolChart.removeTheme("simple");
```

〈Thank You〉