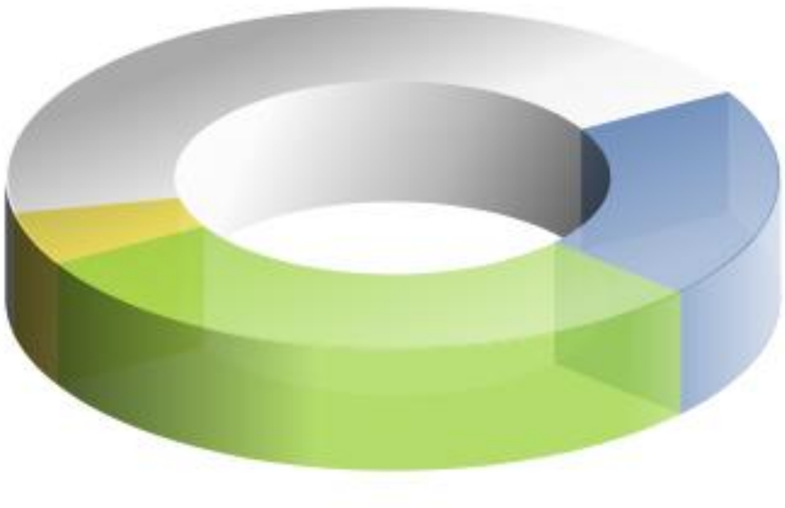
Jensoft
Data Visualization - sw2d charting framework

SW2D Framework v.0.9 Reference Guide

Sébastien JANAUD



To my family, my friends and J.Gil, which will maybe never read this boring reference guide. To all java developer I have never met which maybe would be happy to read this very boring reference guide.

Thanks you for your support.

Table of Contents

1. Introduction

JENSOFT API reference guide helps developers for java charts building with SW2D framework.

1.1. SW2D Framework background

1.1.1. Framework overview

JENSOFT SW2D is a pure java charting framework to make java charts components. JENSOFT API is a complete charting solution for your Swing or Web Application. It provides advanced features you've come to expect with minimal integration efforts. Sw2d gives developers a very flexible toolkit solution for a wide variety of java chart with rich interaction (tools, widgets, transitions) and data visualization needs. The API can be used in many businesses such as reporting tools, health, finance or science and technology.

Core strength of JENSOFT SW2D is its component design. You can easily extend the default components to meet your needs, and extensions will be encapsulated within just those components. As a result, your development teams can create even the largest applications without stepping on each other's code.

Framework provides data oriented plug-ins for most of commons charts components: Curve or Line, Plot or Scatter, Area, Stacked Area, Bar Symbol, Stacked Bar Symbol, Point Symbol, Pie, Pie 3D, Donut2D, Scan, Legend, Capacity, Cloud points, Real time, etc.

Framework provides Tool oriented plug-ins for most of commons transforms and user/chart interaction: Zoom Box, Zoom Lens, Zoom Wheel, Zoom Percent, Translate and Marker. This plug-ins generally offers nice transition during process operations and nice widgets and popup context to lock tool, launch commands or other user intents.

Deploy Desktop and Server Apps use in swing based application and benefits off all swing chart API with very rich interaction and high level user and developer control. Deploy in a Servlet container and process view through internet as a static image processor or use JenScript client invoker to create remote chart from chart server.

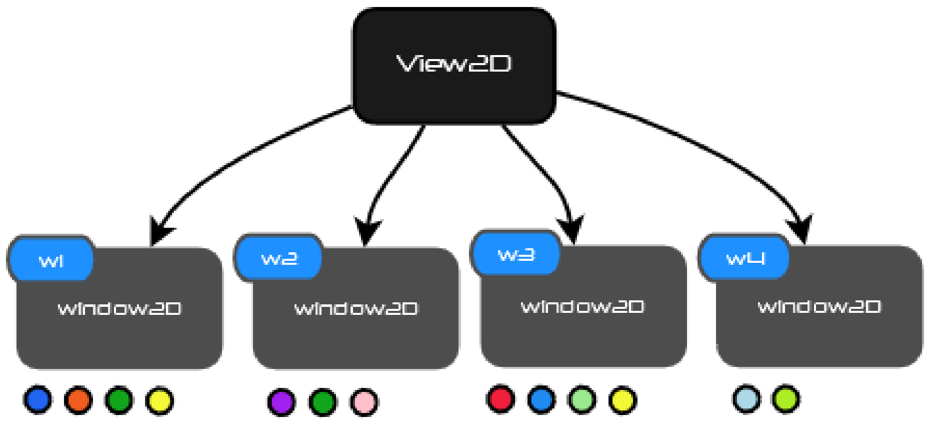
JET (JENSOFT Emitter Template) is a part of JENSOFT API which takes the responsibility of inflate xml document and emits view as various format like view, view as image, view as xml document, view as image file. X2D is the sw2d schema language and it used by JET to validate view document.

1.1.2. When to use SW2D?

In any java application that should have to display charts. That can be in swing based or web application.

1.1.3. Technical logic design

JENSOFT SW2D is built on a simple component model. The view that is the end user chart representation. Windows are object that defines user projection and brings plug-ins which takes the responsibility to process painting operation and user interaction within the window. The figure below shows the view hierarchy. Plug-ins embeds sometimes widgets and context menu.



Note: little's colored circles are not aliens, but plug-ins!

1.1.4. Technical physical design

As a component point of view, the *View2D* is a java AWT component which has 5 children who are called window part.

- North part
- South part
- East part
- West part
- Device part

Each part component invoke plug-in paint operation for the given part. The plug-in takes the responsibility to manage and paint the related part component. The device part is the particular component that has to draw data. East, West, North and south components generally show the information related to data.

1.2. Getting started

Getting started show you how to make a simple view in a minute. First step is to download framework. In second step, you should create a new java project in your favorite IDE.

1.2.1. Download SW2D

To download JENSOFT API, click on the link below:

http://www.jensoft.org/jensoft/Download

Unzip contain and add JENSOFT SW2D jar file into your project class-path.

1.2.2. Veteran Tutorial

Here is a naïve snippet code which shows the framework paradigm. it creates a view, creates a window projection with given bound on x dimension and y dimension, and register two fancy plug-ins *fooPlugin* and *barPlugin* within Window2D instance w2d. This window is register into view. Maybe you need to display overlays for different data source and user projection in only one view. If you need to have multiple projections, you have to create a window instance for each projection.

```
// 1 - view2d is the end user visible chart component.
View2D view2D  = new View2D();

// 2 – window is a user projection that have be to register in view
Window2D w2d = new Window2D.Linear(-10, 12, -5, 26);
view2D.registerWindow2D(w2d);

// 3 – plug-ins paint a scene and handle user interaction in a projection
FooPlugin fooPlugin = new FooPlugin();
BarPlugin barPlugin = new BarPlugin();

// 4 – plug-ins are registered in a window user projection.
w2d.registerPlugin(fooPlugin);
w2d.registerPlugin(barPlugin);

// 5 – plug-ins have sometimes widgets
FooWidget fwidget = new FooWidget();
fooPlugin.registerWidget(fwidget);

// 6 – plug-ins have sometimes context menu
BarContext bcontext = new BarContext();
barPlugin.registerContext(bcontext);
```

Make something with this view! Add in swing panel, return image through network, generate XML templates etc. The framework model allows a very large possibility by plug-in composition. The framework model which is a view, window and plug-ins hierarchy get framework easy for using and learning. You can create rich view in a minute. Framework provides the most common charting component, a rich set of tools with animations and transitions effects.

2. Framework model

2.1. Presentation

As shown in the veteran tutorial, the view building is based on hierarchy of a view that contains windows. Windows which is the user metrics projections hosts plug-ins that represents data and manage user actions and events.

2.2. View2D

View2D is the first API sw2d object that you have to create instance. The view is the end user representation. View inherits from *javax.swing.JComponent* and can be used as any java component is swing UI. Furthermore view can be used outside swing UI as an image or xml document.

The view root component has 5 children. The outer's parts components south, north, west and east and inner device component part.

2.2.1. Create a view

Create a view is quite simple. Only call default constructor. The default outer part components are default set to 40 pixels. For example, sometimes you have to set more because the metrics represents big numbers and needs large width.

```
/**
 * Snippet code to create view2D
 */
View2D view = new View2D();
//set the margin part south, west and east
//default part in view in 40 pixels.
view.setPlaceHolderAxisSouth(60);
view.setPlaceHolderAxisWest(60);
view.setPlaceHolderAxisEast(60);
```

2.2.2. View2D API

For end user the first view responsibility is to register window instance with the method:

```
/**
 * register a window projection in this view
 *
 * @param window2d
 */
 public void registerWindow2D(Window2D window2d)
```

As a developer point of view, there is an interesting method in view class which returns the view as a buffered image. Here is the method signature.

```
/**
 * get the view as an image outside swing UI.
 *
 * @param width
 *            the width to set
 * @param height
 *            the height to set
 *
 * @return the image view as buffered image
 */
public BufferedImage getImageView(int width, int height)
```

This method returns the current image view as a buffered image that should be use outside desktop application. This image can be writing as chart file image or marshall through the network.

2.2.3. View Background

The view has background painters which draw the view background before windows are painting. These painters are only view decorators. Background can be a rounded rectangle filled with a specified shader or an image background.

2.2.3.1.Fill Background

This background fills the view with a gradient.

```
/**
* Snippet code: create a fill view background.
*/
RoundViewFill viewBackground = new RoundViewFill();

Shader s = new Shader(new float[]{0f,1f},new Color[]{new Color(32, 39, 55), Color.
BLACK});
viewBackground.setShader(s);
viewBackground.setOutlineStroke(new BasicStroke(2.5f));

View2D view = new View2D();
view.setBackgroundPainter(viewBackground);
```

2.2.3.2.Image Background

You can use image for the view background. The image is scaled to the entire view by default. If you does not want the scale option, set rescale option to false or use the related constructor. If rescale option is set to false, the image is not scale to the view size and it paint at the view center coordinate.

```
/**
* Snippet code: create image view background.
*/
View2D view = new View2D();
ImageBackground ibg = new ImageBackground(new ImageIcon("bg.png").getImage());
view.setBackgroundPainter(viewBackground);
```

```
/**
* Snippet code: create image view background with no scale image
*/
View2D view = new View2D();
Image i = new ImageIcon("bg.png").getImage();
ImageBackground ibg = new ImageBackground(i,false);
view.setBackgroundPainter(viewBackground);
```

2.3. Window2D

The window 2D defines a user projection with metrics bounds in X and Y dimensions. Windows 2D brings plus-ins that process painting operations in view or handle user interaction. Window instance have to be registered in a view. The window has different nature like Linear.

- *Linear*
- *LogX*
- *LogY*
- *Log*
- *Map*

The Linear implements a linear projection on two dimensions x and y. LogX, LogY and Log inherits from Linear and overrides methods from linear to logarithmic transform. Map is a Mercator projection to use window with map components.

2.3.1. Create window2D

```
/**
 * Snippet code to create a linear window with x and y bounding projection
 */
Window2D w2d = new Window2D.Linear(-60, 100, -20, 140);
w2d.setThemeColor(RosePalette.MANDARIN);
view.registerWindow2D(w2d);
```

```
/**
 * Snippet code to create a log x window with x and y bounding projection
 */
Window2D w2d = new Window2D.LogX(0.01, 1000, -20, 140);
w2d.setThemeColor(RosePalette.MANDARIN);
view.registerWindow2D(w2d);
```

```
/**
 * Snippet code to create a log y window with x and y bounding projection
 */
Window2D w2d = new Window2D.LogY(0, 50, 0.01, 1000);
w2d.setThemeColor(RosePalette.MANDARIN);
view.registerWindow2D(w2d);


/**
 * Snippet code to create a log window with x and y bounding projection
 */
Window2D w2d = new Window2D.Log(0.01, 1000, 0.01, 1000);
w2d.setThemeColor(RosePalette.MANDARIN);
view.registerWindow2D(w2d);
```

2.3.2. Window API

In most of cases, developer has just to create window and register plug-ins. For plug-in developer the interesting window methods are projections methods that process projections from user system coordinate to device system coordinate or device system coordinate to user system coordinate. Projection methods are usually used by plug-in to solve vector objects geometries which have been defined in user projection and obtain vector object in device system coordinate.

```
//create the window projection
Window2D w2d = new Window2D.Linear(-60, 100, -20, 140);
w2d.setThemeColor(RosePalette.MANDARIN);
view.registerWindow2D(w2d);

//transform a user point from user projection to device pixel projection
Point2D userPoint   = new Point2D.Double(-40.34, 88.28);
Point2D devicePoint = w2d.userToPixel(userPoint);
System.out.println(userPoint+" user coordinate get in device pixel system
coordinate : "+devicePoint);

//transform a device pixel point from device projection to user projection
Point2D devicePoint2   = new Point2D.Double(60, 40);
Point2D userPoint2 = w2d.pixelToUser(devicePoint2);
System.out.println(devicePoint2+" device coordinate get in user system coordinate
: "+userPoint2);
```

2.4. Plug-in

A plug-in has two mains roles. Manage painting operations for its host window and manage user interactions. The plug-in is register in a window and listens event type like mouse events to handle actions. In painting view mechanism, the plug-in paint operation is invoked during view rendering. A plug-in can be display widgets in device and have a context menu.

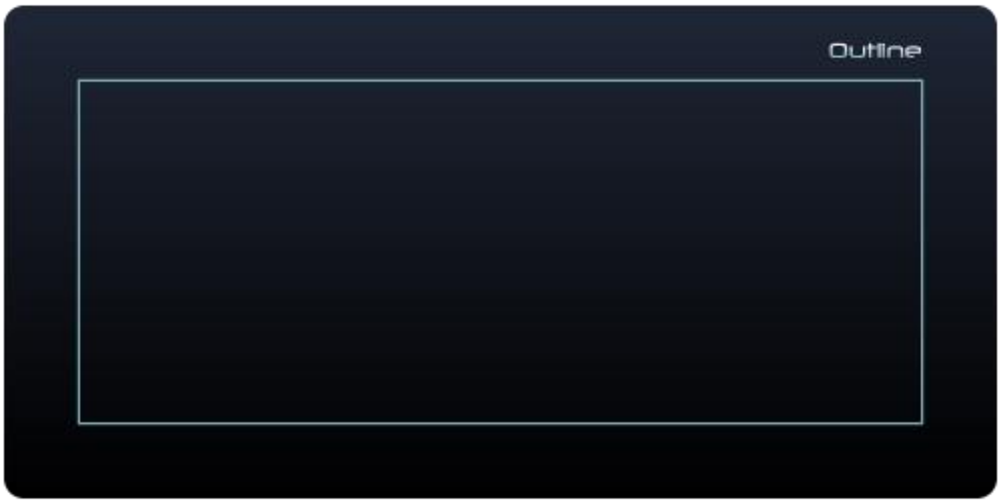
As a user point of view, existing plug-ins include in JenSoft API can be directly use to render such commons charts. Some plug in provides context menu and nice movable widget to execute some command for the host plug in. You can easily extends a context menu or create your new one as an interface between sw2d framework and your business needs. If you are a developer, you can create your own plug in to perform custom operation. An abstract sw2d plug in defines some abstract operation like drawing and has some basic properties and behavior.

3. Common Plug-ins

Commons plug-ins are basics plug-ins like outline, text legend, metrics markers, grids, stripes, etc. It enhances view and charts with decorations.

3.1. Outline

Outline plug-in draws the device outline (not the view outline) with the specified theme color. If the outline theme color is not set, plug-in get the window theme color and use it as the default stroke color. The default place holder for outer parts east, west, south and north is 40 pixels.



```
/**
 * Snippet code to create device outline
 */
View2D view = new View2D();

Window2D w2d = new Window2D.Linear(-60, 100, -20, 140);
view.registerWindow2D(w2d);

OutlinePlugin outline = new OutlinePlugin();
outline.inflateOutline(500);
w2d.registerPlugin(outline);
```

The inflate outline method run an animator that inflate the window outline from device center to final location which represent the device bound.

You can change width for place holder as shown in the following snippet code.

```
/**
 * Snippet code to create view2D with different place holder
 */

View2D view = new View2D();
//set the margin part south, west and east
//default part in view in 40 pixel.
view.setPlaceHolderAxisSouth(60);
view.setPlaceHolderAxisWest(60);
view.setPlaceHolderAxisEast(60);

Window2D w2d = new Window2D.Linear(-60, 100, -20, 140);
view.registerWindow2D(w2d);

OutlinePlugin outline = new OutlinePlugin();
w2d.registerPlugin(outline);
```

3.2. Device Image Background

Image can be drawn in the device part component.

```
/**
 * Snippet code to create device image background
 */

View2D view = new View2D();

Window2D w2d = new Window2D.Linear(-60, 100, -20, 140);
view.registerWindow2D(w2d);

BackgroundPlugin bgPlugin;
try {
     InputStream iis = this.getClass().getResourceAsStream("img.jpg");
     BufferedImage bi = ImageIO.read(iis);
     bgPlugin = new BackgroundPlugin(bi);
     view.registerPlugin(bgPlugin);
    }
catch (IOException e) {}
w2d.registerPlugin(bgPlugin);
```

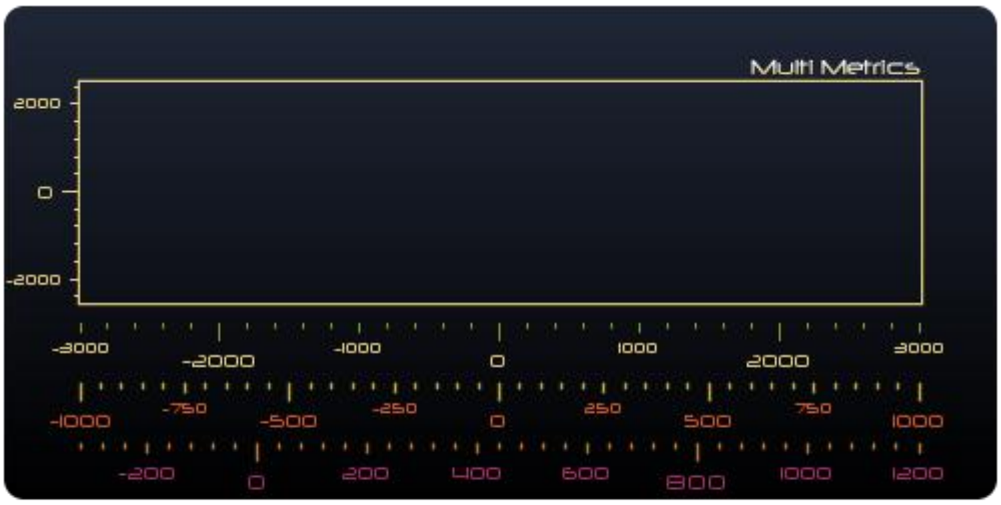
3.3. Text Legend

Legend plug-in is a text legend that lay out text regarding the legend constraints. A legend can be display in axis part south, north, west and east. The legend glyph can be filled with shader.

```
/**
* Snippet code: create a text legend
*/
View2D view = new View2D();

Window2D w2d = new Window2D.Linear(-60, 100, -20, 140);
view.registerWindow2D(w2d);

OutlinePlugin outline = new OutlinePlugin();
outline.inflateOutline(500);
w2d.registerPlugin(outline);

Legend legend = new Legend("My Text Legend");

LegendFill1 fill = new LegendFill1(Color.WHITE,     RosePalette.NEPTUNE);
legend.setLegendFill(fill);
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));

LegendConstraints cons = new LegendConstraints(LegendPosition.North,
                         0.3f,
                         LegendAlignment.Rigth);
legend.setConstraints(cons);

LegendPlugin legendPlugin = new LegendPlugin(legend);
w2d.registerPlugin(legendPlugin);
```

3.4. Metrics

Metrics Plug-in family knows how to draw metrics in window outer parts (north, west, east and south part) and window inner part (device part). Metrics markers and labels are drawn around the device (axis metrics) in an outer part or in the device itself (device metrics). Example above shows a view with multiple axis metrics on left and bottom device sides (for different dimension projection, multiple axis needs multiple windows that define projections).



3.4.1. Metrics Manager

Metrics manager is a metrics solver which creates device metrics for its window2D host with some of given rules. There a kind of metrics manager that lay out metrics for most of common needs. For example, free metrics manager lay out metrics that have been added or register in the manager. Managers process metrics with built-in rules and given input parameters. Other managers generate metrics according some given rules. For figure out a sample, *MiliMetricsManager* generates metrics for the entire window projection (on dimension on it has been associate, west, east, north, south, device) given three factor multiplier major, median and minor and a reference from start. The reference in the implementation is interesting because it gives you a point from start the metrics, then according to your factor multiplier, you can obtain some good accuracy for metrics distribution. If you would obtain low density metrics, dynamic metrics has only one factor multiplier and produces only major metrics, not median and minor metrics.

- Static Metrics manager solves a static distribution of metrics.
- Free Metrics Manager solves metrics for specified given metrics values.
- Flow Metrics Manager solves a metrics flow from start to end value with phase interval.
- Dynamic Metrics Manager solves metrics from reference to window min and max bound.
- Milli Metrics Manager solves different related metrics nature from factors.

3.4.2. Metrics plug-in

Metrics plug-ins renders metrics that are generated by manager which has been registered in plug-in. Plug-ins are be able to display metrics in any view part components. Usually, it can be display on device left and bottom as show above, but it should be in right, left or device. Some built-in metrics plug-in already embeds manager and you don't need to register managers manually except if you want to create your own manager. To create metrics in outer window parts (south, north, west and east parts) you have to use Axis prefixed class plug-in. To create metrics in device part you have to use Device prefixed class plug-in.

3.4.2.1.Axis Metrics Plug-ins

```
package com.jensoft.sw2d.core.plugin.metrics.axis;
```

Each on plug-ins below takes the responsibility to draw metrics label and marker in south, north, west or east part component.

- *AxisStaticMetrics* draws static metrics defined by a given input metrics count.
- *AxisFreeMetrics* draws your specified metrics you registered manually.
- *AxisFlowMetrics* draws a flow of metrics defined by metrics flow start, metrics flow end and a given interval.
- *AxisDynMetrics* draws for the entire window dimension according to a starting reference and a metrics multiplier.
- *AxisMilliMetrics* draws the entire window dimension according to a starting reference and three metrics multiplier, major, median and minor.

All of these plug-in have constructor parameters describe above and the given axis (define by Axis enumeration) on which you expect metrics.

3.4.2.2.Device Metrics Plug-ins

```
package com.jensoft.sw2d.core.plugin.metrics.device;
```

Eachplug-ins below takes the responsibility to draw metrics label and marker in device part component.

- *DeviceStaticMetrics* draws static metrics defined by a given input metrics count.
- *DeviceFreeMetrics* draws your specified metrics you registered manually.
- *DeviceFlowMetrics* draws a flow of metrics defined by metrics flow start, metrics flow end and a given interval.
- *DeviceDynMetrics* draw for the entire window dimension the metrics flow defined by metrics reference and a given interval.
- *DeviceMilliMetrics* draw the entire window dimension with three factor major, median and minor

All of these plug-in have constructor parameters describe above and two additional parameters: the given axis (define by *DeviceAxis* enumeration) on which device x or y axis you expect metrics and the given base line axis value.

3.4.2.3.Metrics Properties

Properties like marker color, label color, label font, axis spacing, metrics format can be set.

- If any color has been set, the window theme color is used.
- Axis spacing is the distance between device and the axis base line.
- Metrics format allow user to format the double representation value.
- Major and Median font.

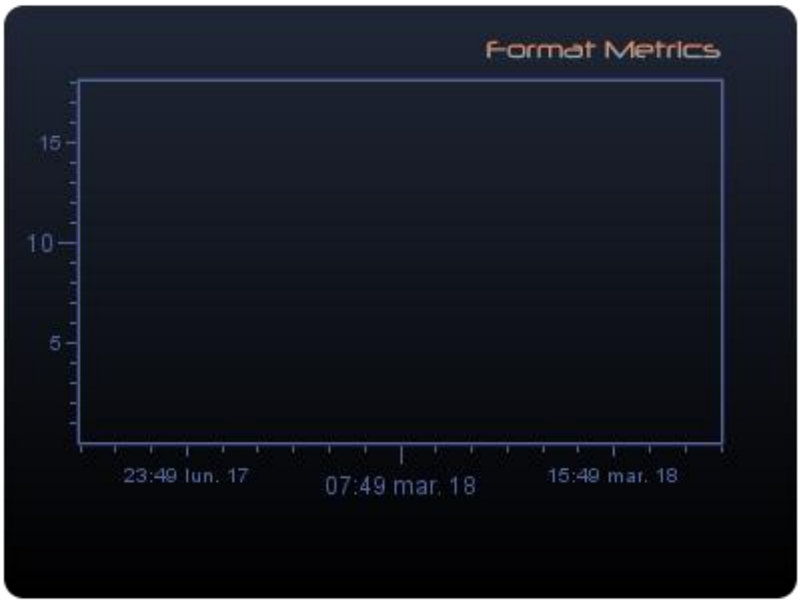
- Marker and label color.
- Range fragment exclusion.

3.4.2.4.Metrics format

The user system precision is double precision, and then framework provides a metrics formatter which is used to format metrics value.

```
/**
* Snippet code: metrics formatter for axis metrics format
*/
View2D view = new View2D();
view.setPlaceHolderAxisSouth(80);
view.setPlaceHolderAxisWest(120);
view.setPlaceHolderAxisEast(120);
//create user projection window and register in view
Window2D w2d = new Window2D.Linear(-60, 100, -20, 140);
w2d.setThemeColor(RosePalette.MANDARIN);
view.registerWindow2D(w2d);

AxisMilliMetrics miliWest = new AxisMilliMetrics(0,Axis.AxisSouth);
miliWest.setMajor(40);
miliWest.setMedian(20);
miliWest.setMinor(5);
w2d.registerPlugin(miliWest);

miliWest.setMetricsFormat(new IMetricsFormat() {

      @Override
      public String format(double d) {
            // Put here you format solver, log, exp, date, latitude, longitude if
            // you are attempting to a specific format.
            return "my value";
      }
});

w2d.registerPlugin(miliWest);
```

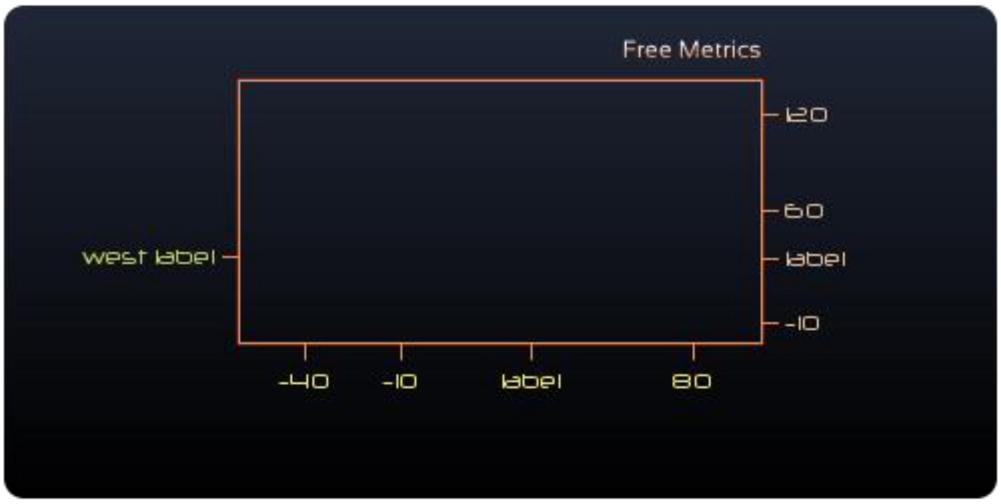
Here is a date format type.



3.4.3. Metrics samples

3.4.3.1.Axis Free metrics

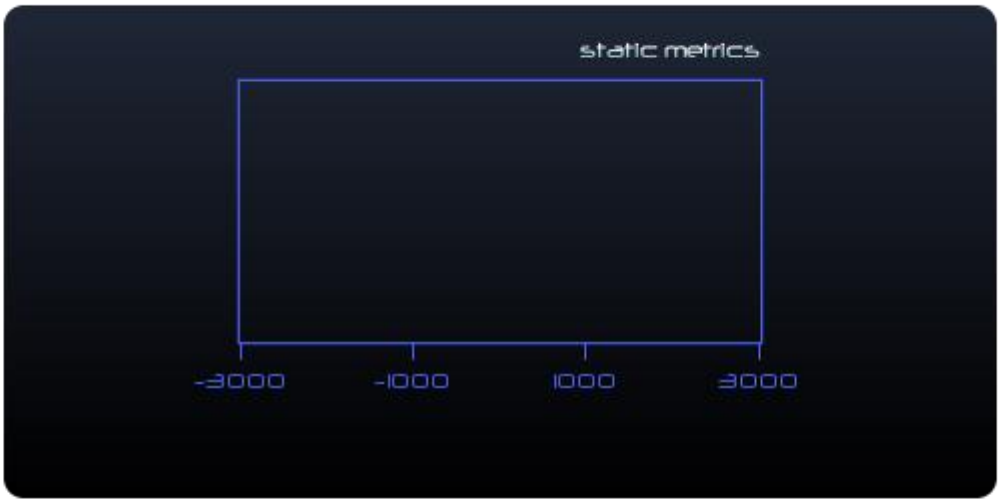
AxisFreeMetrics plug-in embeds free metrics manager. You should have to register your specific metrics. Free metrics rendering is shown in the samples below.



```java
/**
* Snippet code: create free metrics
*/

//first step create view
View2D view = new View2D();
view.setPlaceHolderAxisSouth(80);
view.setPlaceHolderAxisWest(120);
view.setPlaceHolderAxisEast(120);

//create user projection window and register in view
Window2D w2d = new Window2D.Linear(-60, 100, -20, 140);
w2d.setThemeColor(RosePalette.MANDARIN);
view.registerWindow2D(w2d);

//AxisFreeMetrics in part west
AxisFreeMetrics westFreeMetrics = new AxisFreeMetrics(Axis.AxisWest);
westFreeMetrics.addMetrics(32, "west label");
westFreeMetrics.getMetricsLayoutManager().setMetricsLabelColor(RosePalette.
LIME);
w2d.registerPlugin(westFreeMetrics);



// AxisFreeMetrics in part southw
AxisFreeMetrics southFreeMetrics = new AxisFreeMetrics(Axis.AxisSouth);
southFreeMetrics.getMetricsLayoutManager().setMetricsLabelColor(RosePalette
.LEMONPEEL);
southFreeMetrics.addMetrics(-40);
southFreeMetrics.addMetrics(-10);
southFreeMetrics.addMetrics(30, "label");
southFreeMetrics.addMetrics(80);
w2d.registerPlugin(southFreeMetrics);

//AxisFreeMetrics in part east
AxisFreeMetrics eastFreeMetrics = new AxisFreeMetrics(Axis.AxisEast);
eastFreeMetrics.getMetricsLayoutManager().setMetricsLabelColor(RosePalette.
MELON);
eastFreeMetrics.addMetrics(-10);
eastFreeMetrics.addMetrics(30, "label");
eastFreeMetrics.addMetrics(60);
eastFreeMetrics.addMetrics(120);
w2d.registerPlugin(eastFreeMetrics);

//outline plu-in
w2d.registerPlugin(new OutlinePlugin());
```

3.4.3.2.Axis Static Metrics

Static metrics embeds static metrics manager which produces static metrics. The static manager creates static metrics marker with a given constant that split window user bound in equal segments. Even if the window is zoomed in, the static does not change location but change metrics value. Static is related to the physical location in device.

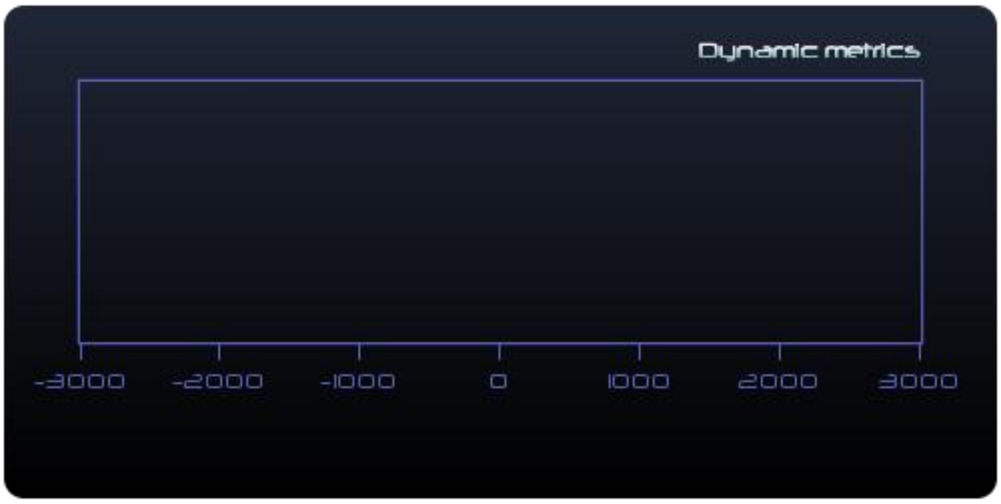


```
/**
* Snippet code: create 4 static metrics
*/

View2D view = new View2D();

view.setPlaceHolderAxisSouth(80);
view.setPlaceHolderAxisWest(120);
view.setPlaceHolderAxisEast(120);

Window2D w2d = new Window2D.Linear(-3000, 3000, -2500, 2500);
w2d.setName("metrics window");
view.registerWindow2D(w2d);

AxisStaticMetrics staticMetrics = new AxisStaticMetrics(4, Axis.AxisSouth);
w2d.registerPlugin(staticMetrics);
```

Static metrics keeps location even if window user bound changed. User bound is always split in equal segment regarding the given constant.

3.4.3.3.Axis Dynamic Metrics

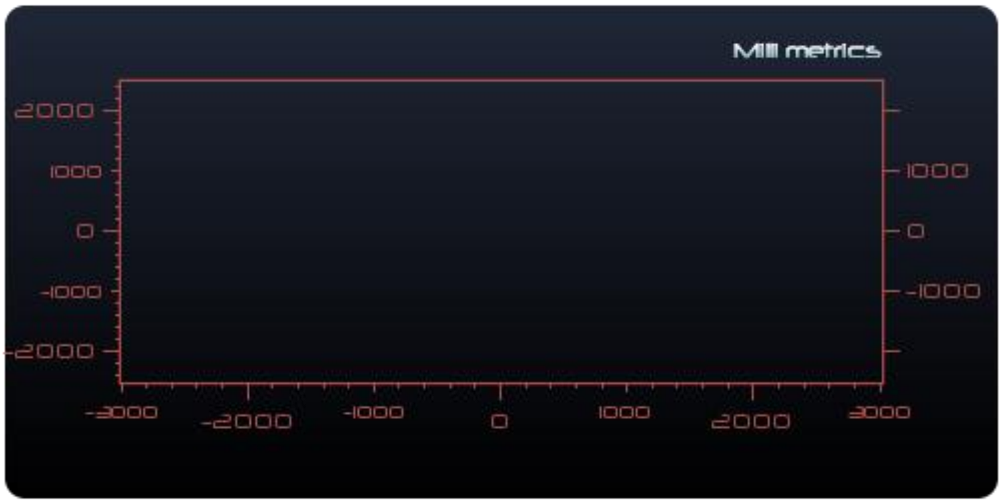
Dynamic metrics creates simple metrics marker for the entire current window user bound regarding the specified reference and metrics multiplier. Dynamic manager



```
/**
* Snippet code: create dynamic metrics on axis west, east and south
* with starting reference 0 and interval 1000
*/

View2D view = new View2D();

view.setPlaceHolderAxisSouth(80);
view.setPlaceHolderAxisWest(80);
view.setPlaceHolderAxisEast(80);

Window2D w2d = new Window2D.Linear(-3000, 3000, -2500, 2500);
w2d.setThemeColor(Spectral.SPECTRAL_PURPLE1.brighter());
view.registerWindow2D(w2d);

AxisDynMetrics dynWest = new AxisDynMetrics(0, 1000, Axis.AxisWest);
w2d.registerPlugin(dynWest);

AxisDynMetrics dynSouth = new AxisDynMetrics(0, 1000, Axis.AxisSouth);
w2d.registerPlugin(dynSouth);

AxisDynMetrics dynEast = new AxisDynMetrics(0, 1000, Axis.AxisEast);
w2d.registerPlugin(dynEast);
```

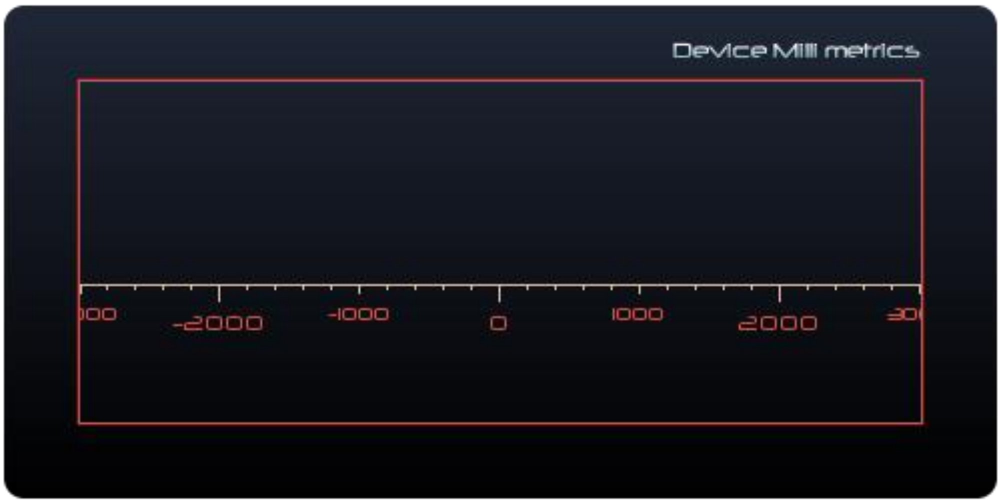
3.4.3.4.Axis Milli Metrics

Milli metrics creates three metrics marker types for the entire current window user bound regarding the specified reference, and minor, median and major factors multiplier. Reference gives you a start point entry to obtain homogenous metrics values. The manager solves the maximum count metrics without overlap according to the reference and multipliers. The manager strategy is to lay out most of metrics along axis, and then major, median and minor factors are divide by 2 to obtain greater metrics density. When metrics overlap conflict occurs, factors are divided by 2 to decrease density and obtain non conflicted metrics.



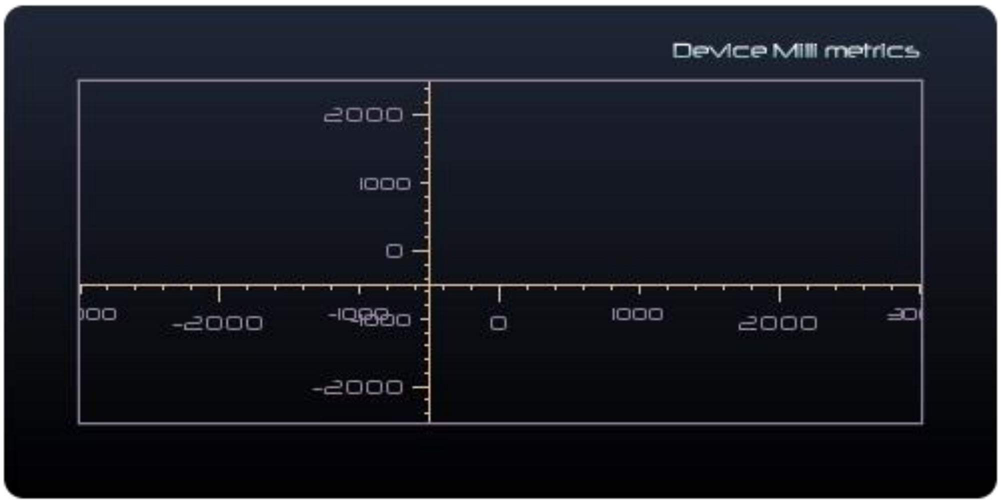
```
/**
* Snippet code: create milli metrics on axis west, east and south
* with starting reference 0 and 1000 as the major factor, 500 as the median
* and 100 as the minor.
*/

View2D view = new View2D();

view.setPlaceHolderAxisSouth(60);
view.setPlaceHolderAxisWest(60);
view.setPlaceHolderAxisEast(60);

Window2D w2d = new Window2D.Linear(-3000, 3000, -2500, 2500);
w2d.setName("metrics window");
view.registerWindow2D(w2d);

AxisMilliMetrics miliWest = new AxisMilliMetrics(0, Axis.AxisWest);
miliWest.setMajor(1000);
miliWest.setMedian(500);
miliWest.setMinor(100);
w2d.registerPlugin(miliWest);

AxisMilliMetrics miliSouth = new AxisMilliMetrics(0, Axis.AxisSouth);
miliSouth.setMajor(1000);
miliSouth.setMedian(500);
miliSouth.setMinor(100);
w2d.registerPlugin(miliSouth);

AxisMilliMetrics miliEast = new AxisMilliMetrics(0, Axis.AxisEast);
miliEast.setMajor(1000);
w2d.registerPlugin(miliEast);

w2d.registerPlugin(new OutlinePlugin());
```

3.4.3.5.Device Milli Metrics



```
/**
* Snippet code: create the x device metrics
*/

View2D view = new View2D();


Window2D w2d = new Window2D.Linear(-3000, 3000, -2500, 2500);
view.registerWindow2D(w2d);

DeviceMilliMetrics dmmX = new DeviceMilliMetrics(0, -500,
                        DeviceAxis.AxisX, MarkerPosition.S);
dmmX.setMajor(1000);
dmmX.setMedian(500);
dmmX.setMinor(100);
w2d.registerPlugin(dmmX);
dmmX.exclude(-500);

dmmX.getMetricsLayoutManager().setMetricsMarkerColor(RosePalette.MELON);
dmmX.getMetricsLayoutManager().setMetricsBaseLineColor(
      RosePalette.MELON);


w2d.registerPlugin(new OutlinePlugin());
```

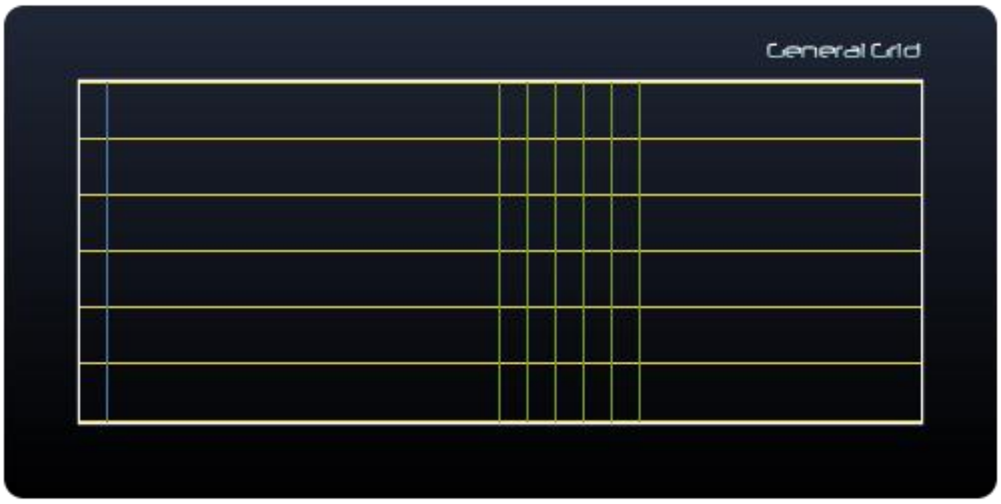
3.4.3.6.Multiple Device Axis



```
/**
* Snippet code: create x and y device metrics
*/
View2D view = new View2D();

Window2D w2d = new Window2D.Linear(-3000, 3000, -2500, 2500);
view.registerWindow2D(w2d);

DeviceMilliMetrics dmmX = new DeviceMilliMetrics(0, -500,
                        DeviceAxis.AxisX, MarkerPosition.S);
dmmX.setMajor(1000);
dmmX.setMedian(500);
dmmX.setMinor(100);
dmmX.exclude(-500);
w2d.registerPlugin(dmmX);

dmmX.getMetricsLayoutManager().setMetricsMarkerColor(RosePalette.MELON);
dmmX.getMetricsLayoutManager().setMetricsBaseLineColor(
      RosePalette.MELON);



DeviceMilliMetrics dmmY = new DeviceMilliMetrics(0, -500,
                        DeviceAxis.AxisY, MarkerPosition.W);
dmmY.setMajor(1000);
dmmY.setMedian(500);
dmmY.setMinor(100);
dmmY.exclude(-500);
w2d.registerPlugin(dmmY);

dmmY.getMetricsLayoutManager().setMetricsMarkerColor(RosePalette.MELON);
dmmY.getMetricsLayoutManager().setMetricsBaseLineColor(
      RosePalette.MELON);

w2d.registerPlugin(new OutlinePlugin());
```

3.5. Grid

Grid Plug-in knows how to draw grid lines in device part component like show below.



3.5.1. Grid Manager

Grid manager is a grid solver which creates device grid for its window2D host with some of given rules. There a kind of grid manager that lay out grid for most of common needs. For example, free grid manager lay out grid that have been added or register in the manager. Managers process grids with built-in rules and given input parameters.

- Static Grid manager solves a static distribution of grids.
- Free Grid Manager solves grids for specified given grids values.
- Flow Grid Manager solves a grid flow from start to end value with phase interval.
- Dynamic Grid Manager solves grids from reference to window min and max bound.
- Compound grid manager which compounds some sub managers.

3.5.2. Grid plug-in

Grid plug-in renders their registered managers in view. You can register the manager of your choice and register it in the Grid plug-in or use built-in plug-ins with embedded manager. The grids managers are nearly from metrics manager except that it generate grid with some input parameters like reference, interval and orientation.

```
package com.jensoft.sw2d.core.plugin.grid;
```

Each on plug-ins below takes the responsibility to draw grids in device part component.

- *StaticGridPlugin* draws static grids defined by a given input grid count.
- *FreeGridPlugin* draws your specified grids you registered manually.
- *FlowGridPlugin* draws a flow of grids defined by grid flow start, grid flow end and a given interval.
- *DynGridPlugin* draw for the entire window dimension the metrics flow defined by metrics reference and a given interval.

All of these plug-in have constructor parameters describe above and the given grid orientation (define by *GridOrientation* enumeration) on which you expect grid.

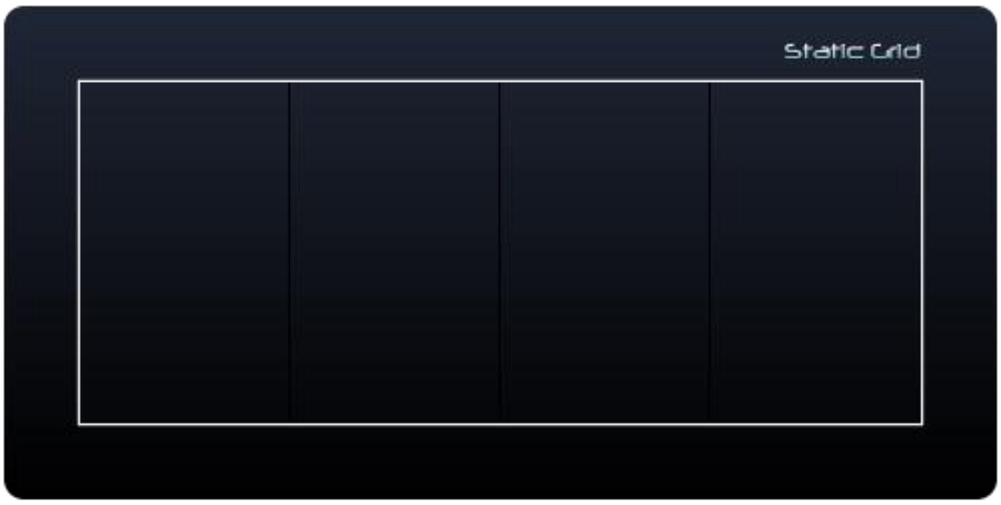
3.5.3. Grid properties

Some grid property can be set.

- Grid color
- Grid Stroke
- Grid label

3.5.4. Grid Samples

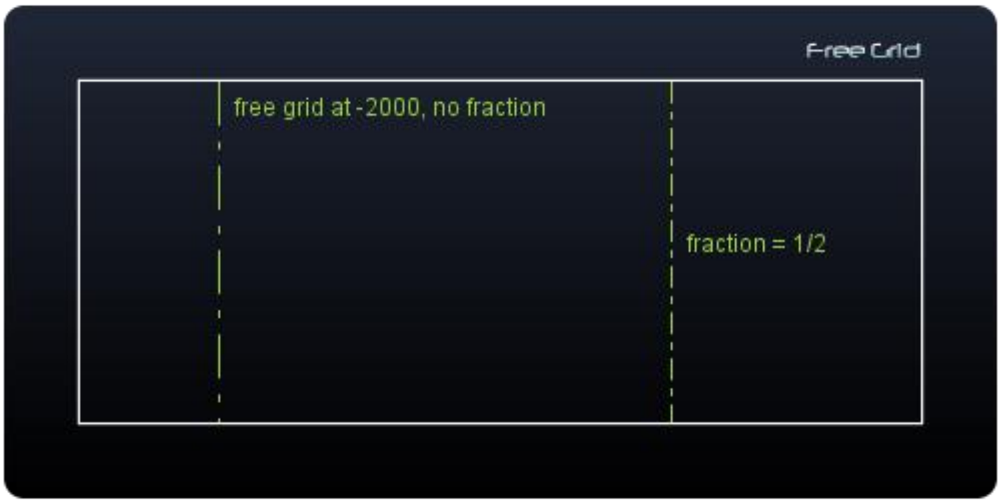
3.5.4.1.Static grid



```
/**
* Snippet code: create static grid
*/


View2D view = new View2D();

Window2D w2d = new Window2D.Linear(-3000, 3000, -3000, 3000);
w2d.setName("free grid window");

view.registerWindow2D(w2d);
w2d.setThemeColor(Color.WHITE);
StaticGridManager sgm = new StaticGridManager(GridOrientation.Vertical,
                       3);

GridPlugin staticGrid = new GridPlugin(sgm);
w2d.registerPlugin(staticGrid);

Legend legend = new Legend("Static Grid");
legend.setLegendFill(new LegendFill1(Color.WHITE,
                       RosePalette.NEPTUNE));
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.North, 0.3f,
                       LegendAlignment.Rigth));
LegendPlugin legendPlugin = new LegendPlugin(legend);
w2d.registerPlugin(legendPlugin);
```

3.5.4.2.Free Grid

With Free grids manager you can add specified value to obtain some expected metrics. On this grid type, you can too specify a label.

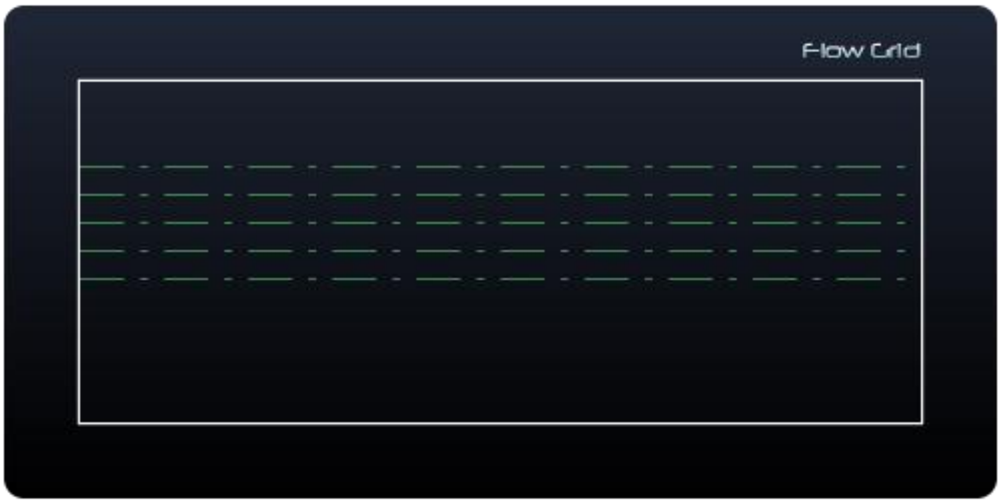


```
/**
* Snippet code: create free X grid
*/
View2D view = new View2D();

Window2D w2d = new Window2D.Linear(-3000, 3000, -3000, 3000);
w2d.setName("free grid window");

view.registerWindow2D(w2d);
w2d.setThemeColor(Color.WHITE);
StaticGridManager sgm = new StaticGridManager(GridOrientation.Vertical,
                        3);

FreeGridManager dglY = new FreeGridManager();
dglY.setGridOrientation(GridOrientation.Vertical);
dglY.setGridColor(JennyPalette.JENNY1);
dglY.addGrid(1234, "  fraction = 1/2", TangoPalette.CHAMELEON2,
                        StrokePalette.drawingStroke6, 1 / 2f);
dglY.addGrid(-2000, "  free grid at -2000, no fraction",
                        TangoPalette.CHAMELEON2,
StrokePalette.drawingStroke5);
dglY.setGridStroke(new BasicStroke(3));

Legend legend = new Legend("Static Grid");
legend.setLegendFill(new LegendFill1(Color.WHITE,
                        RosePalette.NEPTUNE));
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.North, 0.3f,
                        LegendAlignment.Rigth));
LegendPlugin legendPlugin = new LegendPlugin(legend);
w2d.registerPlugin(legendPlugin);


/**
* Snippet code: create free Y grid
*/


View2D view = new View2D();

Window2D w2d = new Window2D.Linear(-3000, 3000, -3000, 3000);
w2d.setName("free grid window");

view.registerWindow2D(w2d);
w2d.setThemeColor(Color.WHITE);
StaticGridManager sgm = new StaticGridManager(GridOrientation.Vertical,
                        3);

FreeGridManager dglY = new FreeGridManager();
dglY.setGridOrientation(GridOrientation.Vertical);
dglY.setGridColor(JennyPalette.JENNY1);
dglY.addGrid(1234, "  fraction = 1/2", TangoPalette.CHAMELEON2,
                        StrokePalette.drawingStroke6, 1 / 2f);
dglY.addGrid(-2000, "  free grid at -2000, no fraction",
                        TangoPalette.CHAMELEON2,
StrokePalette.drawingStroke5);
dglY.setGridStroke(new BasicStroke(3));
```

```
Legend legend = new Legend("Static Grid");
legend.setLegendFill(new LegendFill1(Color.WHITE,
                        RosePalette.NEPTUNE));
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.North, 0.3f,
                        LegendAlignment.Rigth));
LegendPlugin legendPlugin = new LegendPlugin(legend);
w2d.registerPlugin(legendPlugin);
```

3.5.4.3.Flow grid

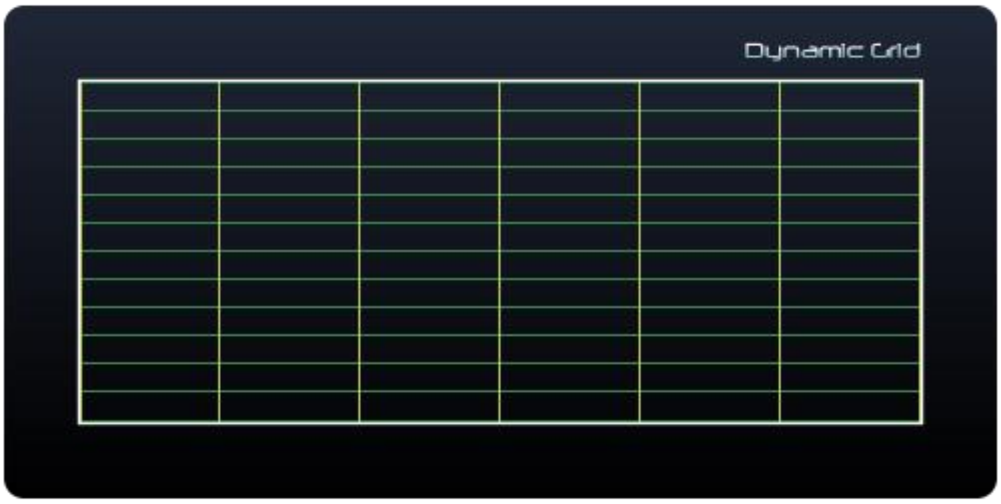
The flow grid manager produces grids between a range segment specified by the flow start and the flow end value in the user system coordinate and the step increment.



```
/**
* Snippet code: create flow grid
*/


View2D view = new View2D();

Window2D w2d = new Window2D.linear(-3000, 3000, -3000, 3000);
w2d.setName("free grid window");

view.registerWindow2D(w2d);
w2d.setThemeColor(Color.WHITE);
StaticGridManager sgm = new StaticGridManager(GridOrientation.Vertical,
                        3);

FlowGridManager dglY = new FlowGridManager(GridOrientation.Horizontal,
                        -500, 1500, 500);
dglY.setGridColor(RosePalette.EMERALD);
dglY.setGridStroke(StrokePalette.drawingStroke5);
w2d.registerPlugin(new GridPlugin(dglY));
```

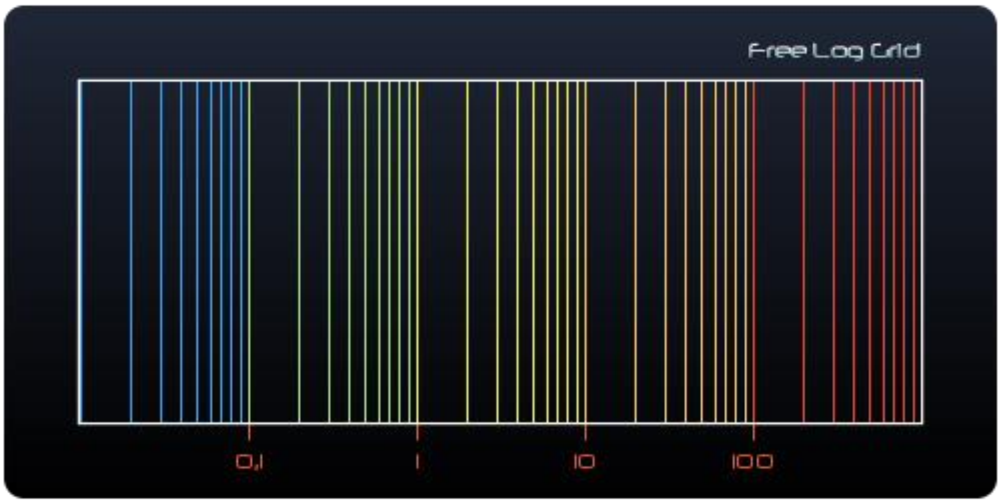
```
Legend legend = new Legend("Static Grid");
legend.setLegendFill(new LegendFill1(Color.WHITE,
                        RosePalette.NEPTUNE));
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.North, 0.3f,
                        LegendAlignment.Rigth));
LegendPlugin legendPlugin = new LegendPlugin(legend);
w2d.registerPlugin(legendPlugin);
```

3.5.4.4.Dynamic grid



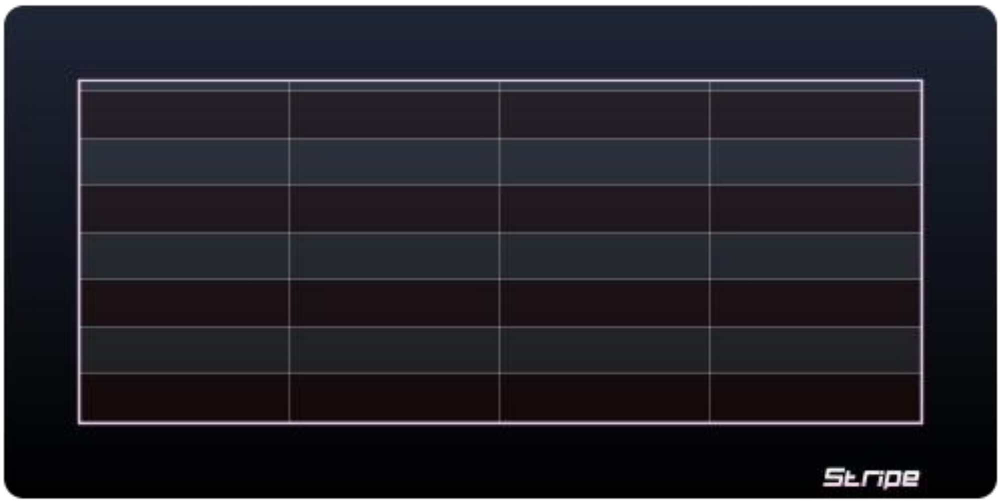
```
/**
* Snippet code: create dynamic grid
*/


View2D view = new View2D();

Window2D w2d = new Window2D(-3000, 3000, -3000, 3000);
w2d.setName("free grid window");

view.registerWindow2D(w2d);
w2d.setThemeColor(Color.WHITE);
StaticGridManager sgm = new StaticGridManager(GridOrientation.Vertical,
                        3);

DynamicGridManager dglX = new DynamicGridManager(
                        GridOrientation.Vertical, 0, 1000);
dglX.setGridColor(RosePalette.LIME);
GridPlugin gridPluginX = new GridPlugin(dglX);
w2d.registerPlugin(gridPluginX);

Legend legend = new Legend("Static Grid");
legend.setLegendFill(new LegendFill1(Color.WHITE,
                        RosePalette.NEPTUNE));
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.North, 0.3f,
                        LegendAlignment.Rigth));
LegendPlugin legendPlugin = new LegendPlugin(legend);
w2d.registerPlugin(legendPlugin);
```

3.5.4.5.Free Grid on Log Window



3.6. Stripe

Stripe Plug-in knows how to draw stripes in device part component like show below.



3.6.1. Stripe Manager

Stripe manager is a stripe solver which creates device stripes for its window2D host with some of given rules. There a kind of stripe manager that lay out stripe for most of common needs. For example, free stripe manager lay out stripe that have been registered in the manager. Managers process stripes with built-in rules and given input parameters.

- Free stripe manager solves grids for specified given grids values.
- Flow stripe manager solves a stripe flow from start to end value with phase interval.
- Dynamic stripe manager solves stripes from reference to window min and max bound.

3.6.2. Stripe plug-in

Stripe plug-in renders their registered managers in view. You can register the manager of your choice and register it in the Strip plug-in or use built-in plug-ins with embedded manager.

```
package com.jensoft.sw2d.core.plugin.stripe;
```

Each on plug-ins below takes the responsibility to draw stripes in device part component.

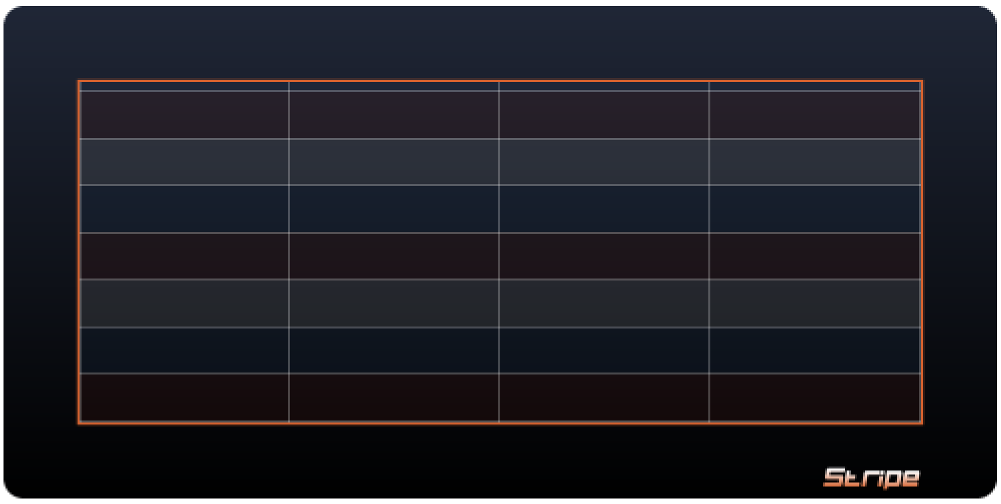
- *FreeStripePlugin* draws specified stripes which have been registered manually.
- *FlowStripePlugin* draws a flow of stripes defined by flow start, flow end and a given interval.
- *DynStripePlugin* draw for the entire window dimension the stripes flow defined by flow reference and a given interval.

All of these plug-in have constructor parameters describe above and the given stripe orientation (define by *StripeOrientation* enumeration) on which you expect stripe.

Here is an example with three colors contain in stripe palette.

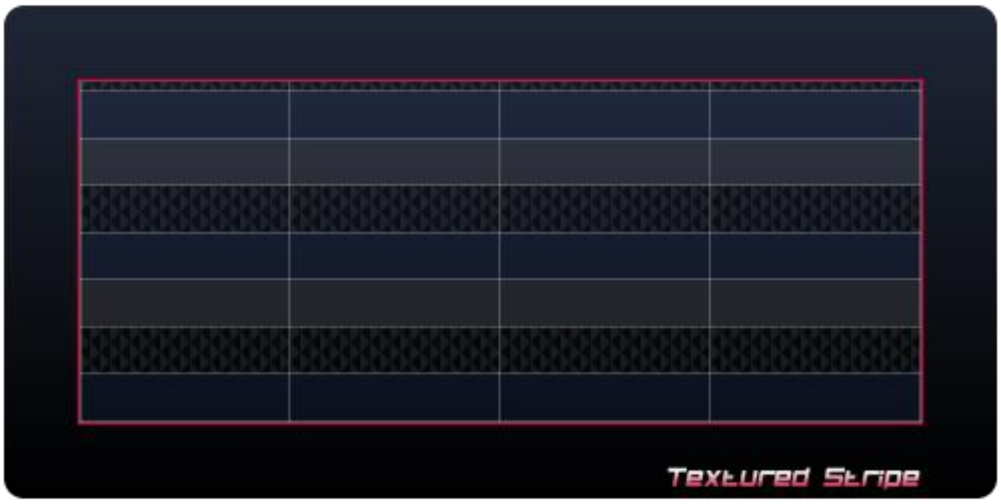
3.6.3. Stripe Sample

3.6.3.1.Dynamic Stripes



```
/**
* Snippet code: create dynamic stripe
*/

View2D view = new View2D();
Window2D window2D = new Window2D(0, 10, 0, 18);
view.registerWindow2D(window2D);

DynamicStripeManager dbm = new DynamicStripeManager(
                        StripeOrientation.Horizontal, 0, 2.5);
StripePlugin stripePlugin = new StripePlugin(dbm);

StripePalette bp = new StripePalette();
bp.addPaint(new Color(255, 255, 255, 80));
bp.addPaint(ColorPalette.alpha(TangoPalette.SCARLETRED3, 80));
dbm.setBandPalette(bp);
stripePlugin.setAlpha(0.3f);
window2D.registerPlugin(stripePlugin);


DynamicGridManager dgm = new DynamicGridManager(
                        GridOrientation.Horizontal, 0, 2.5);
dgm.setGridColor(new Color(255, 255, 255, 60));
GridPlugin gridLayout = new GridPlugin(dgm);
window2D.registerPlugin(gridLayout);

DynamicGridManager dgm2 = new DynamicGridManager(
                        GridOrientation.Vertical, 0, 2.5);
dgm2.setGridColor(new Color(255, 255, 255, 60));
GridPlugin gridLayout2 = new GridPlugin(dgm2);
window2D.registerPlugin(gridLayout2);


window2D.registerPlugin(new OutlinePlugin());

Legend legend = new Legend("Stripe");
legend.setFont(InputFonts.getFont(InputFonts.ELEMENT, 14));
legend.setLegendFill(new LegendFill1(Color.WHITE, window2D
                        .getThemeColor()));
legend.setConstraints(new LegendConstraints(LegendPosition.South, 0.8f,
                        LegendAlignment.Rigth));
LegendPlugin lgendL = new LegendPlugin(legend);
window2D.registerPlugin(lgendL);
```

3.6.3.2. Textured Stripes



```
/**
* Snippet code: create dynamic stripe
*/

View2D view = new View2D();
Window2D window2D = new Window2D(0, 10, 0, 18);
view.registerWindow2D(window2D);

DynamicStripeManager dbm = new DynamicStripeManager(
                        StripeOrientation.Horizontal, 0, 2.5);
StripePlugin stripePlugin = new StripePlugin(dbm);

StripePalette bp = new StripePalette();
bp.addPaint(new Color(255, 255, 255, 80));
bp.addPaint(ColorPalette.alpha(JennyPalette.JENNY8, 80));
bp.addPaint(TexturePalette.getT1());dbm.setBandPalette(bp);
stripePlugin.setAlpha(0.3f);
window2D.registerPlugin(stripePlugin);


DynamicGridManager dgm = new DynamicGridManager(
                        GridOrientation.Horizontal, 0, 2.5);
dgm.setGridColor(new Color(255, 255, 255, 60));
GridPlugin gridLayout = new GridPlugin(dgm);
window2D.registerPlugin(gridLayout);

DynamicGridManager dgm2 = new DynamicGridManager(
                        GridOrientation.Vertical, 0, 2.5);
dgm2.setGridColor(new Color(255, 255, 255, 60));
GridPlugin gridLayout2 = new GridPlugin(dgm2);
window2D.registerPlugin(gridLayout2);


window2D.registerPlugin(new OutlinePlugin());

Legend legend = new Legend("Textured Stripe");
legend.setFont(InputFonts.getFont(InputFonts.ELEMENT, 14));
legend.setLegendFill(new LegendFill1(Color.WHITE, window2D
                        .getThemeColor()));
legend.setConstraints(new LegendConstraints(LegendPosition.South, 0.8f,
                        LegendAlignment.Rigth));
LegendPlugin lgendL = new LegendPlugin(legend);
window2D.registerPlugin(lgendL);
```

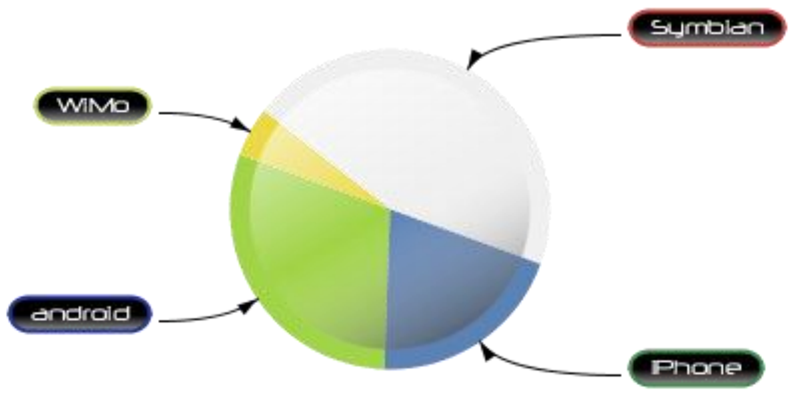
4. Data oriented plug-ins

A data visualization plug-in is data's oriented plug-ins like pie, pie3D, curves, symbols, etc. it main role is to represent data in the view and handles some behavior on figures that represent the data.

4.1. Pie Chart

A pie chart (or a circle graph) is a circular chart divided into sectors, illustrating proportion. In a pie chart, the arc length of each sector (and consequently its central angle and area), is proportional to the quantity it represents. When angles are measured with 1 turn as unit then a number of percent is identified with the same number of centiturns. Together, the sectors create a full disk. It is named for its resemblance to a pie which has been sliced.

The pie chart is perhaps the most widely used statistical chart in the business world. However, it has been criticized, and some recommend avoiding it, pointing out in particular that it is difficult to compare different sections of a given pie chart, or to compare data across different pie charts. Pie charts can be an effective way of displaying information in some cases, in particular if the intent is to compare the size of a slice with the whole pie, rather than comparing the slices among them. Pie charts work particularly well when the slices represent 25 to 50% of the data, but in general, other plots such as the bar chart or the dot plot, may be more adapted for representing certain information.



4.1.1. Pie

```
package com.jensoft.sw2d.core.plugin.pie;
```

The pie geometry is defined by it center and radius in pixel. This pie center location can be projected in device component system coordinate and the window user system coordinate which became a pie vector. (*PieNature* enumeration)

```java
/**
* Snippet code: create pie
*/


Pie pie = new Pie("my pie", 90);
pie.setPieNature(PieNature.PieUser);
pie.setStartAngleDegree(0);
pie.setCenterX(0);
pie.setCenterY(0);
```

This pie is the user system coordinate, it center point P (0,0) with 90 pixel radius then window need to be center at zero like (-1,1,-1,1) for example.

```java
/**
* Snippet code: register pie in a user projection
*/


View2D view = new View2D();


/**
* to expect pie in the view center you need a symmetric window because
* pie center is projected in the user system coordinate (if pie is defined
* with the pie user nature (user system projection) and the center at
* Point P(0,0))
*/

Window2D window2D = new Window2D(-1, 1, -1, 1);
view.registerWindow2D(window2D);

/**
* Register a pie plug in the symmetric window.
*/

PiePlugin piePlugin = new PiePlugin();
window2D.registerPlugin(piePlugin);


/**
* Create pie instance and add it in plug in instance.
*/

piePlugin.addPie(pie);
```

PieToolkit provides static factory methods to create pie and related objects. *PieView* is a view that inherits *view2D* and embeds a symmetric *window2D* and a *piePlugin* (or make manually like the example above)

```
/**
* Snippet code: create pie with factory and register in a pie view.
*
*/


PieView view = PieToolkit.createCompatibleView(0);
Pie pie = PieToolkit.createPie("pie", 70);
pie.setPieEffect(new PieLinearEffect());
view.addPie(pie);
```

4.1.2. Slice

The pie is building by added sectors or slices. A slice is defined by a relative double value which will be normalized to solve slices geometry. Each slice is defined by a name, a value, a theme color and the slice divergence.

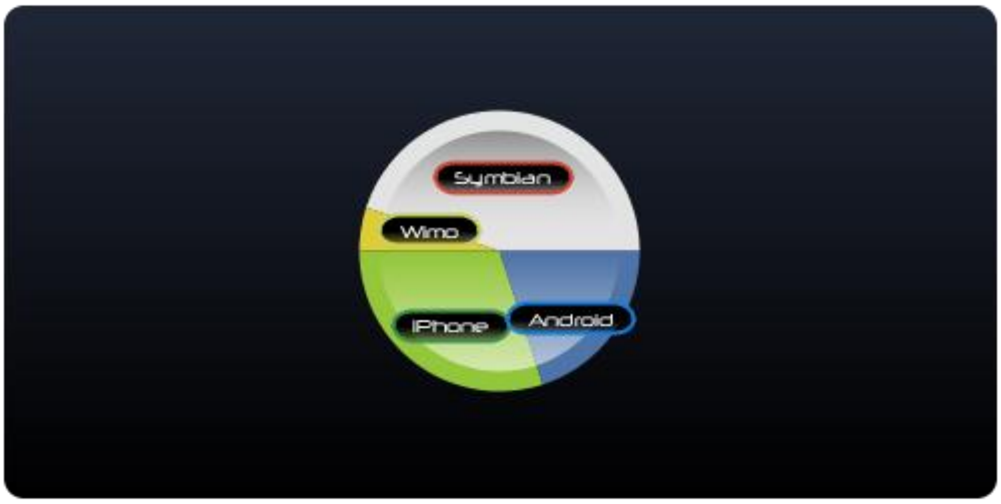
```
/**
* Snippet code: create pie slice.
*/

PieSlice s1 = PieToolkit.createSlice("gray", new Color(240, 240, 240,
                        240), 45, 0);
PieSlice s2 = PieToolkit.createSlice("gray",
                        ColorPalette.alpha(TangoPalette.BUTTER2, 240), 5,
                        0);

PieSlice s3 = PieToolkit.createSlice("chameleon",
                        ColorPalette.alpha(TangoPalette.CHAMELEON2, 240),
                        30, 0);
PieSlice s4 = PieToolkit.createSlice("blue",
                        ColorPalette.alpha(TangoPalette.SKYBLUE2, 240), 20,
                        0);
/**
* register slices in the pie with the push method in toolkit factory
* or register manually with addSlice(Slice slice) method in pie.
*/

PieToolkit.pushSlices(pie, s1, s2, s3, s4);
```

4.1.3. Pie Label

Labels can be added on slice with various kinds of styles.

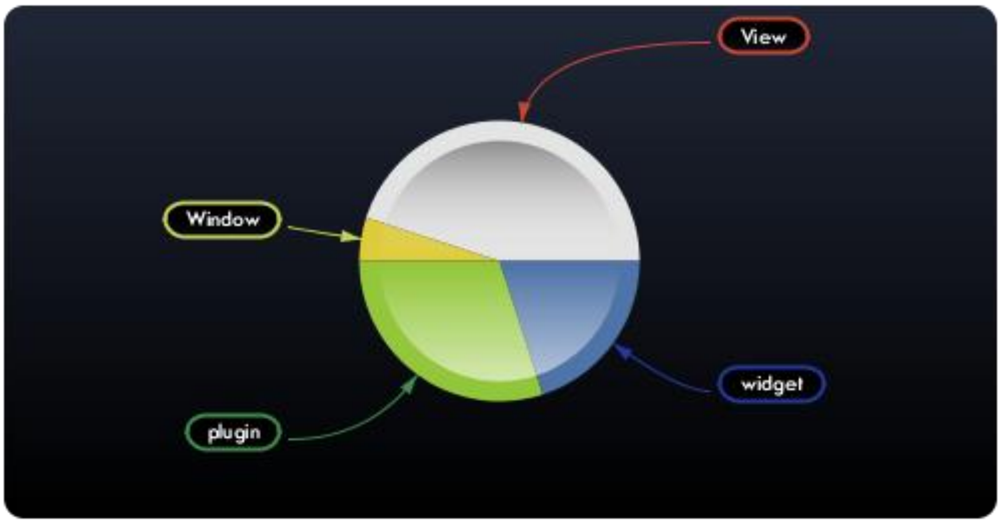
4.1.3.1.Pie Bound Label

The pie bound label draws a centered label in the slice bounding rectangle.



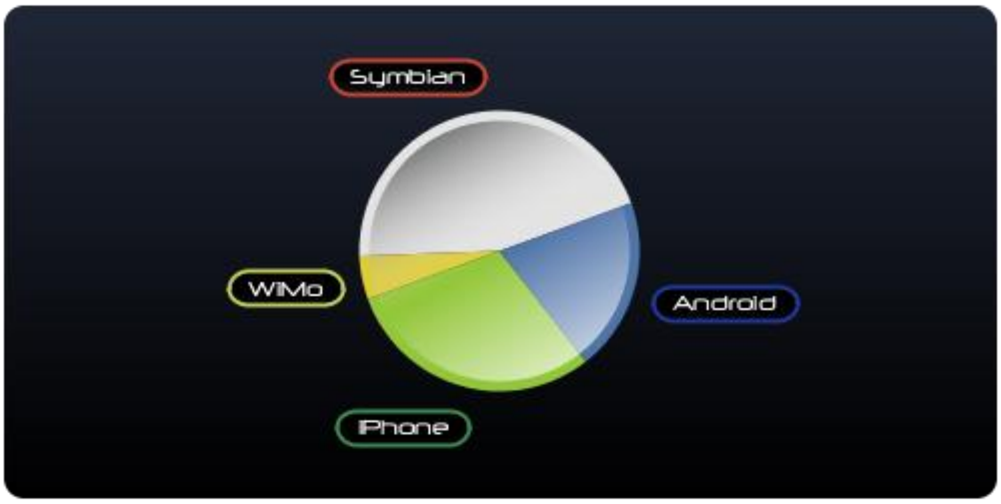
```
/**
* Snippet code: create pie slice bound label
*/

float[] fractions = { 0f, 0.5f, 1f };
Color[] colors = { new Color(0, 0, 0, 100), new Color(0, 0, 0, 255),
                       new Color(0, 0, 0, 255) };
Stroke s = new BasicStroke(2);
pie.setPassiveLabelAtMinPercent(0);

PieBoundLabel label1 = PieToolkit.createBoundLabel("Java",
            ColorPalette.WHITE, InputFonts.getNeuropol(10));
label1.setStyle(Style.Both);
label1.setOutlineStroke(s);
label1.setShader(fractions, colors);
label1.setOutlineColor(RosePalette.REDWOOD);
label1.setOutlineRound(20);


/**
* add label on slice
*/

s1.addSliceLabel(label1);
```

4.1.3.2.Pie Border Label



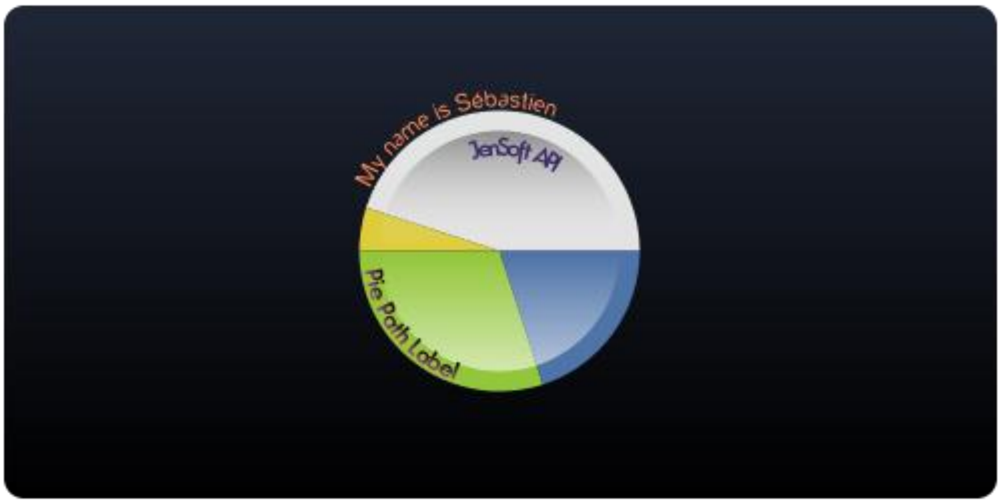
```
/**
* Snippet code: create pie slice border label
*/

float[] fractions = { 0f, 0.5f, 1f };
Color[] colors = { new Color(0, 0, 0, 100), new Color(0, 0, 0, 255),
                       new Color(0, 0, 0, 255) };
Stroke s = new BasicStroke(2);
Font f = InputFonts.getNoMove(10);
PieBorderLabel label1 = PieToolkit.createBorderLabel("View",
                       ColorPalette.WHITE, f, 30);
label1.setStyle(Style.Both);
label1.setOutlineStroke(s);
label1.setShader(fractions, colors);
label1.setOutlineColor(RosePalette.REDWOOD);
label1.setOutlineRound(20);
label1.setLinkColor(RosePalette.REDWOOD);
label1.setLinkStyle(LinkStyle.Quad);
label1.setLinkExtends(40);
label1.setMargin(50);


/**
* add label on slice
*/

s1.addSliceLabel(label1);
```

4.1.3.3.Pie Radial Label



```
/**
* Snippet code: create pie slice radial label
*/

float[] fractions = { 0f, 0.5f, 1f };
Color[] colors = { new Color(0, 0, 0, 100), new Color(0, 0, 0, 255),
                        new Color(0, 0, 0, 255) };
Stroke s = new BasicStroke(2);

/**
* create radial label with pie tool kit facotory
*/


PieRadialLabel label1 = PieToolkit.createRadialLabel("Symbian",
                        ColorPalette.WHITE, InputFonts.getNeuropol(12),
20);
label1.setStyle(Style.Both);
label1.setOutlineStroke(s);
label1.setShader(fractions, colors);
label1.setOutlineColor(RosePalette.REDWOOD);
label1.setOutlineRound(20);

/**
* add label on slice
*/

s1.addSliceLabel(label1);
```

4.1.3.4.Pie Path Label



```
/**
* Snippet code: create pie slice path label
*/

float[] fractions = { 0f, 0.5f, 1f };
Color[] colors = { new Color(0, 0, 0, 100), new Color(0, 0, 0, 255),
                   new Color(0, 0, 0, 255) };
Stroke s = new BasicStroke(2);

/**
* create radial label with pie tool kit facotory
*/


PiePathLabel ppl = new PiePathLabel(TextPosition.Right, "My Label");
ppl.setPathSide(PathSide.Below);
ppl.setLabelFont(InputFonts.getYorkville(12));
ppl.setLabelColor(RosePalette.MANDARIN);
ppl.setDivergence(2);

/**
* add label on slice
*/

s1.addSliceLabel(label1);
```

4.1.4. Pie Effect

4.1.4.1.Pie Linear Effect

4.1.4.2.Pie Radial Effect

4.1.5. Pie Listener

Pie listener handles fire events from slices. Slice enter, exit, press, released, clicked can be observed. If you need to handle specific behaviors when these events occur, you have to register a pie listener.

```
/**
* Snippet code: register a pie listener on pie view (you can register a pie
* listener on the pie plug in, the addPieListener method in pie view is a
* delegate method from pie plug in)
*/


view.addPieListener(new PieListener() {

      @Override
      public void pieSliceReleased(PieEvent e) {
            System.out.println("slice pieSliceReleased "
                        + e.getSlice().getName());
      }

      @Override
      public void pieSlicePressed(PieEvent e) {
            System.out.println("slice pieSlicePressed "
                              + e.getSlice().getName());
      }

      @Override
      public void pieSliceExited(PieEvent e) {
            System.out.println("slice pieSliceExited "
                        + e.getSlice().getName());
      }

      @Override
      public void pieSliceEntered(PieEvent e) {
            System.out.println("slice pieSliceEntered "
                        + e.getSlice().getName());
      }

      @Override
      public void pieSliceClicked(PieEvent e) {
            System.out.println("slice pieSliceClicked "
                        + e.getSlice().getName());
      }
});
```

4.1.6. Pie Animator

4.1.6.1.Pie Alpha animator

Pie alpha animator makes an alpha transition on slice when mouse click or rollover a slice.

```
/**
* Snippet code: register the alpha animator on the pie plug in
*/


piePlugin.addPieAnimator(new PieAlphaAnimator(0.1f, 1f));
```

4.1.6.2.Pie Divergence Animator

Pie divergence runs a divergence animator from current divergence to the target divergence.

```
/**
* Snippet code: register the divergence animator with the 40 divergence
value
*/


piePlugin.addPieAnimator(new PieDivergenceAnimator(40));
```

4.1.6.3.Pie Flash Animator

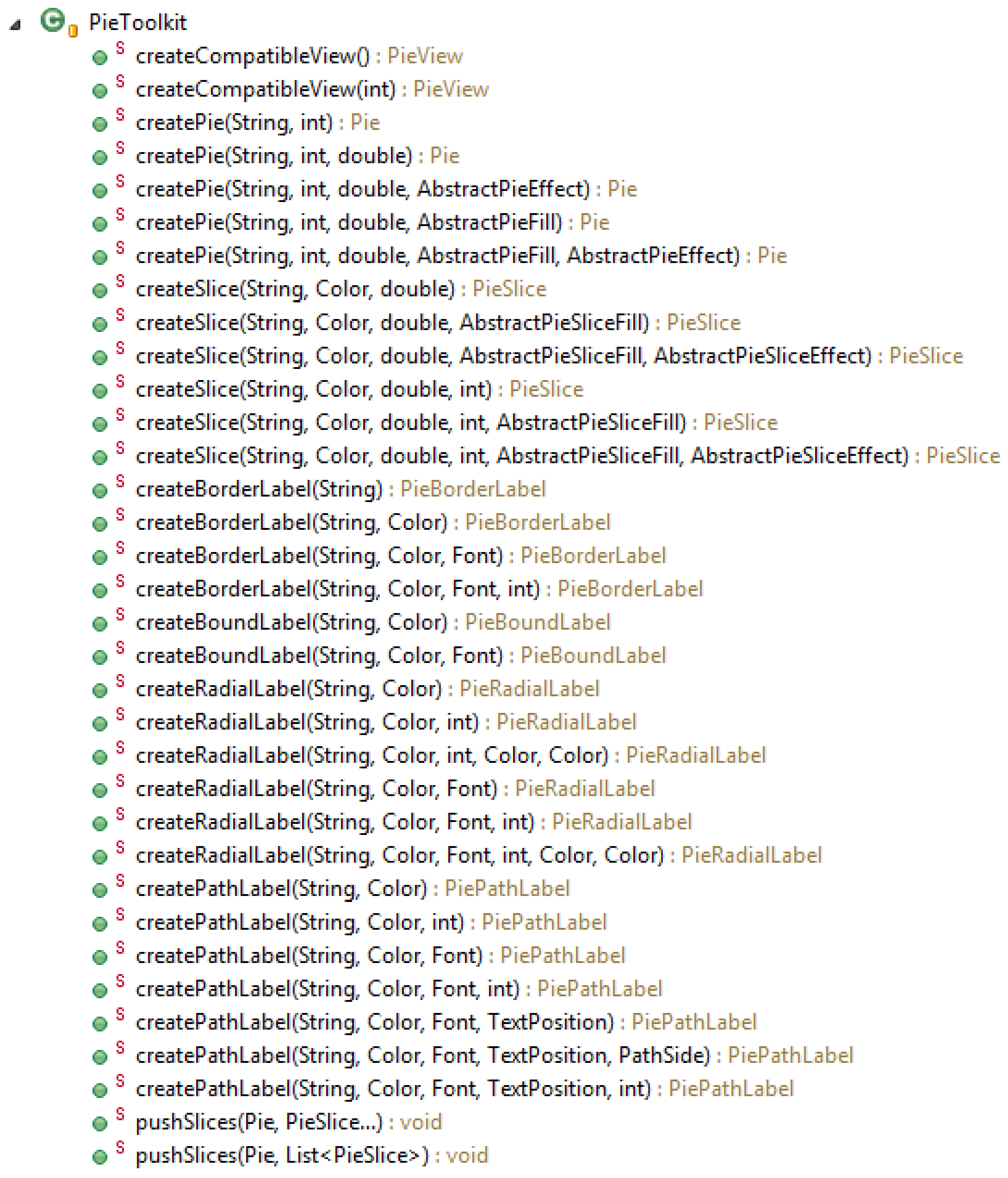
Pie flash run a cycle alpha transition on slice and create a flash effect for the slice.

```
/**
* Snippet code: register the flash animator from the alpha 0.2 to 0.6 with
* a very slow flash.
*/

PieFlashAnimator flash = new PieFlashAnima-
tor(0.2f,0.6f,FlashSpeed.VeryFast);
flash.setShowFlashBehavior(ShowFlashBehavior.ShowOnSliceRollover);

piePlugin.addPieAnimator(flash);
```

You can set the *ShowFlashBehavior* if you want flash start on a slice rollover or on a slice clicked and the flash velocity with *Flash Speed.*

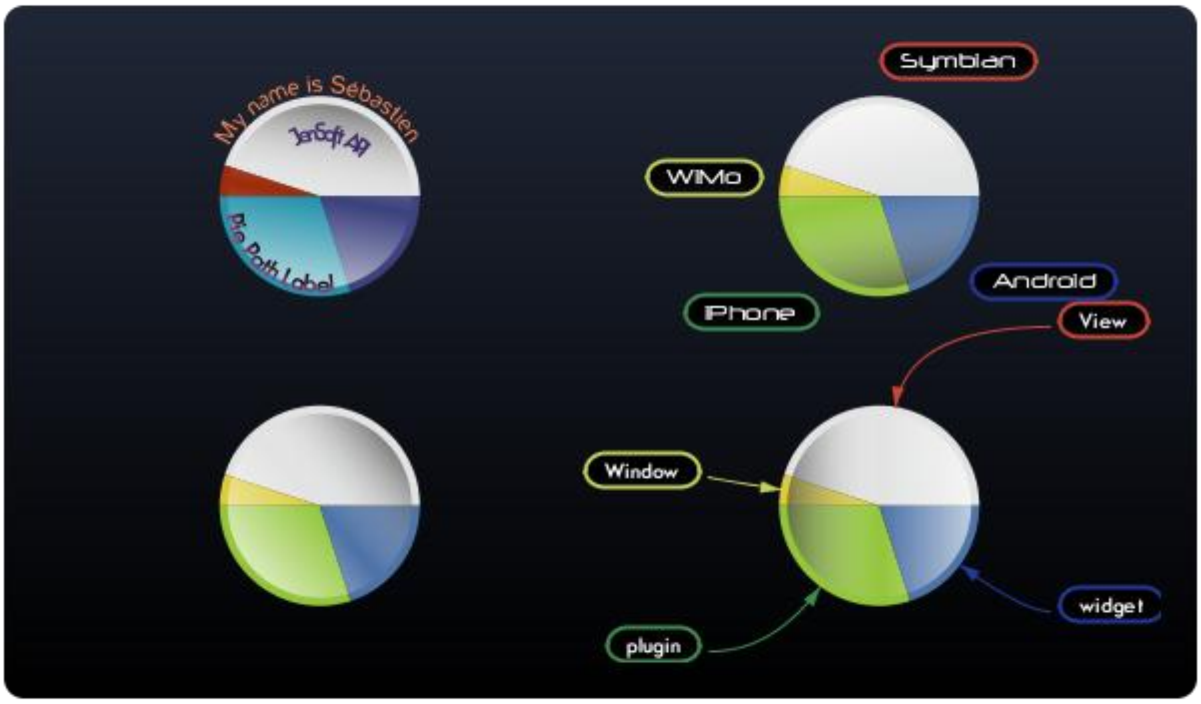
4.1.7. Pie Toolkit

Here is the *PieToolkit* outline. It provides all static factory method to create pie compatible view, pie, pie slice, pie bound label, pie radial label, pie border label, pie slices pushing.



```
PieToolkit
    createCompatibleView() : PieView
    createCompatibleView(int) : PieView
    createPie(String, int) : Pie
    createPie(String, int, double) : Pie
    createPie(String, int, double, AbstractPieEffect) : Pie
    createPie(String, int, double, AbstractPieFill) : Pie
    createPie(String, int, double, AbstractPieFill, AbstractPieEffect) : Pie
    createSlice(String, Color, double) : PieSlice
    createSlice(String, Color, double, AbstractPieSliceFill) : PieSlice
    createSlice(String, Color, double, AbstractPieSliceFill, AbstractPieSliceEffect) : PieSlice
    createSlice(String, Color, double, int) : PieSlice
    createSlice(String, Color, double, int, AbstractPieSliceFill) : PieSlice
    createSlice(String, Color, double, int, AbstractPieSliceFill, AbstractPieSliceEffect) : PieSlice
    createBorderLabel(String) : PieBorderLabel
    createBorderLabel(String, Color) : PieBorderLabel
    createBorderLabel(String, Color, Font) : PieBorderLabel
    createBorderLabel(String, Color, Font, int) : PieBorderLabel
    createBoundLabel(String, Color) : PieBoundLabel
    createBoundLabel(String, Color, Font) : PieBoundLabel
    createRadialLabel(String, Color) : PieRadialLabel
    createRadialLabel(String, Color, int) : PieRadialLabel
    createRadialLabel(String, Color, int, Color, Color) : PieRadialLabel
    createRadialLabel(String, Color, Font) : PieRadialLabel
    createRadialLabel(String, Color, Font, int) : PieRadialLabel
    createRadialLabel(String, Color, Font, int, Color, Color) : PieRadialLabel
    createPathLabel(String, Color) : PiePathLabel
    createPathLabel(String, Color, int) : PiePathLabel
    createPathLabel(String, Color, Font) : PiePathLabel
    createPathLabel(String, Color, Font, int) : PiePathLabel
    createPathLabel(String, Color, Font, TextPosition) : PiePathLabel
    createPathLabel(String, Color, Font, TextPosition, PathSide) : PiePathLabel
    createPathLabel(String, Color, Font, TextPosition, int) : PiePathLabel
    pushSlices(Pie, PieSlice...) : void
    pushSlices(Pie, List<PieSlice>) : void
```

4.1.8. Multiple Pie

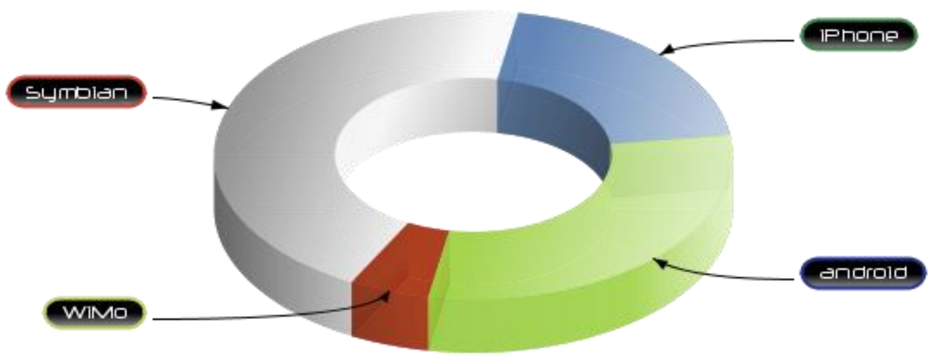

My name is Sébastien
JenSoft API
Pie Path Label
Symbian
WIMo
Android
IPhone
View
Window
widget
plugin

```
/**
* Snippet code: create multiple pie.
*/

PieView view = PieToolkit.createCompatibleView();
//bound the window to center pie on symmetric points (-2,-2), (-2,2), (2,-2),
(2,2)
view.getWindow2D().bound(-4, 4, -4, 4);

//pie is default projected in the user system coordinate
Pie pie = PieToolkit.createPie("pie1", 50);
pie.setCenterX(-2);
pie.setCenterY(-2);
pie.setPieEffect(new PieLinearEffect(40,4));

PieSlice s1 = PieToolkit.createSlice("gray", new Color(240, 240, 240,
                         240), 45, 0);
PieSlice s2 = PieToolkit.createSlice("gray",
                         ColorPalette.alpha(TangoPalette.BUTTER2, 240), 5, 0);
PieSlice s3 = PieToolkit.createSlice("chameleon",
                         ColorPalette.alpha(TangoPalette.CHAMELEON2, 240), 30,
                         0);
PieSlice s4 = PieToolkit.createSlice("blue",
                         ColorPalette.alpha(TangoPalette.SKYBLUE2, 240), 20, 0);
//register slices in pie
PieToolkit.pushSlices(pie, s1, s2, s3, s4);

//register pie in view (delegate method from pie plug in which is embedded in
//compatible pie view)
view.addPie(pie);

//here is labels

pie = PieToolkit.createPie("pie2", 50);
pie.setCenterX(-2);
pie.setCenterY(2);
pie.setPieEffect(new PieLinearEffect(80,4));

//here is slices labels instances..

pie = PieToolkit.createPie("pie3", 50);
pie.setCenterX(2);
pie.setCenterY(-2);
pie.setPieEffect(new PieLinearEffect(180,4));
view.addPie(pie);

//here is slices labels instances..

pie = PieToolkit.createPie("pie4", 50);
pie.setCenterX(2);
pie.setCenterY(2);
pie.setPieEffect(new PieLinearEffect(280,4));
view.addPie(pie);

//here is slices labels instances..
```

4.2. Pie 3D Chart

A Pie 3D chart or Donut3D is a circular 3D chart divided into sectors, illustrating proportion. In a Doughnut chart, the arc length of each sector (and consequently its central angle and area), is proportional to the quantity it represents. When angles are measured with 1 turn as unit then a number of percent is identified with the same number of centiturns.

The pie chart is perhaps the most widely used statistical chart in the business world. However, it has been criticized, and some recommend avoiding it, pointing out in particular that it is difficult to compare different sections of a given pie chart, or to compare data across different pie charts. Pie charts can be an effective way of displaying information in some cases, in particular if the intent is to compare the size of a slice with the whole pie, rather than comparing the slices among them. Pie charts work particularly well when the slices represent 25 to 50% of the data, but in general, other plots such as the bar chart or the dot plot, may be more adapted for representing certain information.



4.2.1. Pie 3D

```
package com.jensoft.sw2d.core.plugin.donut3d;
```

The donut geometry is defined by it center, inner radius, outer radius (radius in pixels), thickness and tilt in degree. This pie 3D center location can be projected in device component system coordinate and the window user system coordinate which became a pie vector. (*Donut3DNature* enumeration). Pie and related objects can be creating manually or with the *Donut3DToolkit.*

```
/**
* Snippet code: create pie 3D manually
*/
Donut3DPlugin donut3dPlugin = new Donut3DPlugin();

Donut3D donut3d = new Donut3D();
donut3d.setDonut3DNature(Donut3DNature.Donut3DUser);
donut3d.setCenterX(0);
donut3d.setCenterY(0);
donut3d.setOuterRadius(160);
donut3d.setInnerRadius(80);
donut3d.setStartAngleDegree(60);
donut3d.setThickness(80);
donut3d.setTilt(40);

donut3dPlugin.addDonut(donut3d);


View2D view = new View2D();
Window2D w2d = new Window2D(-1, 1, -1, 1);
view.registerWindow2D(w2d);

w2d.registerPlugin(donut3dPlugin);

/**
* Snippet code: create donut3D 3D view and donut 3D with tool kit
* A Donut 3D compatible view (Donut3DView) already embeds symmetric window
* and a pie plug in instance.
*/
Donut3DView view = Donut3DToolkit.createCompatibleView();

/**
*create donut and add in the compatible view.
*/
Donut3D donut3d = Donut3DToolkit.createDonut3D("myDonut", 80, 160, 80,
                         60, 40);
view.addDonut3D(donut3d);
```

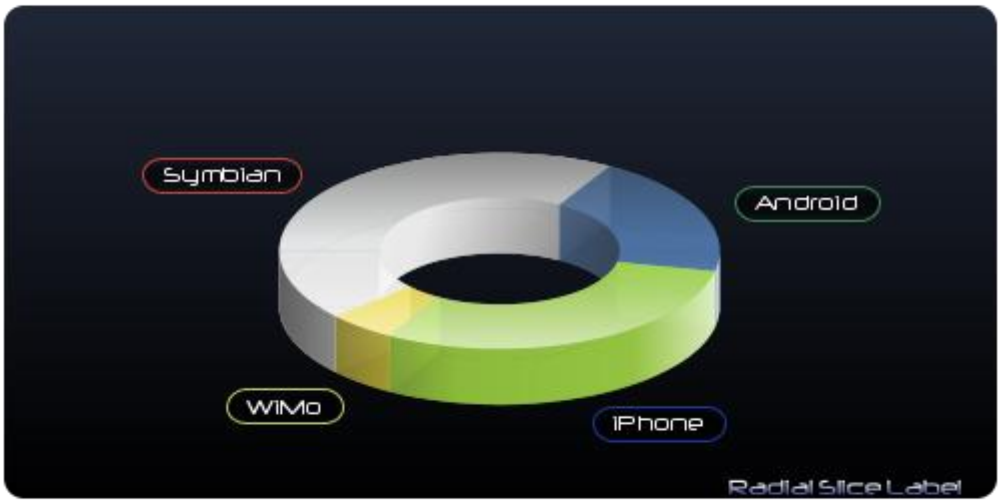
4.2.2. Pie 3D Slice

```
/**
* Snippet code: pie slice manually
*/

Donut3DSlice slice = new Donut3DSlice("white", new Color(250, 250, 250, 255));
slice.setValue(45);

/**
* Snippet code: pie slice with donut 3D tool kit
*/

Donut3DSlice s1 = Donut3DToolkit.createDonut3DSlice("slice",
                            new Color(250, 250, 250), 45);

/**
* Snippet code: pushing pie slices in the host pie
*/
Donut3DToolkit.pushSlices(donut3d, s1, s2, s3, s4);
```

4.2.3. Pie 3D Label

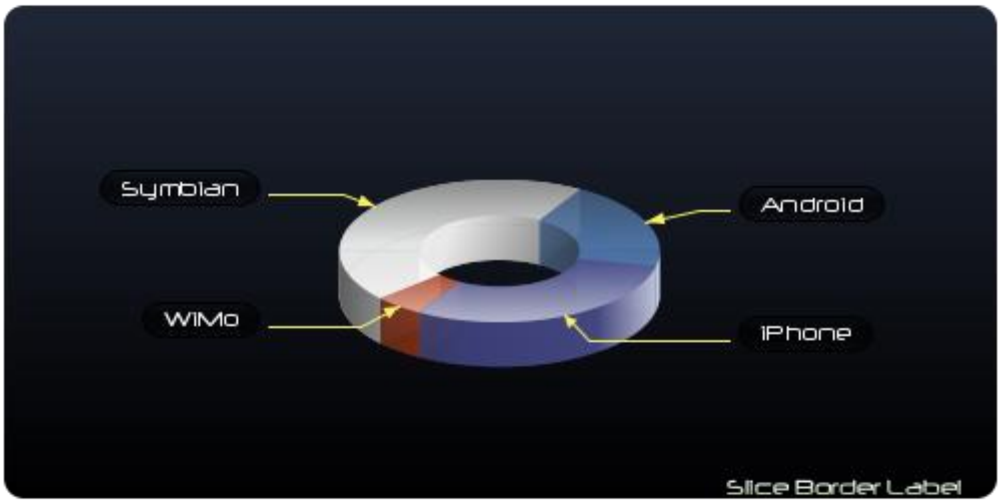
Various kind of pie 3D label helps you to represent data identification and decoration.

4.2.3.1.Pie 3D Radial Label



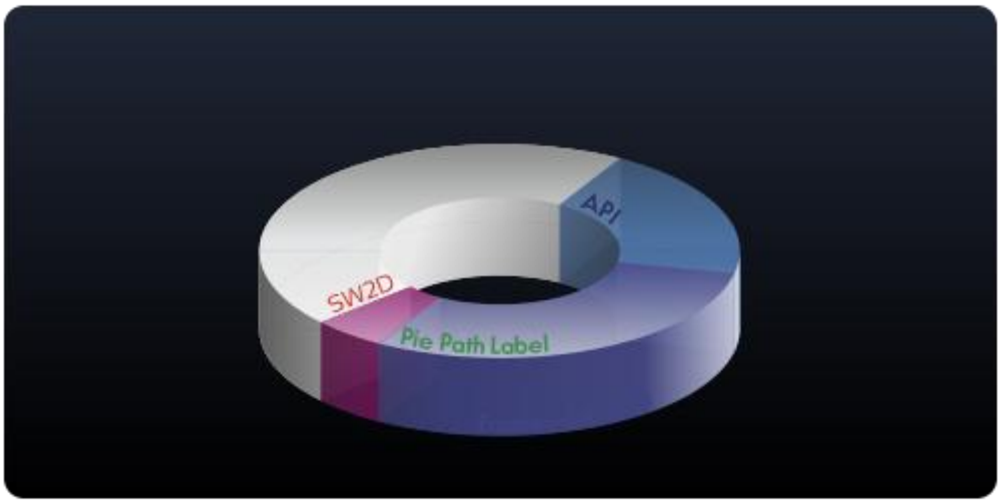
```
/**
* Snippet code: pie radial label
*/

Donut3DRadialLabel label1 = Donut3DToolkit.createRadialLabel("Symbian",
                                RosePalette.COALBLACK,
                                InputFonts.getNeuropol(12),
                                30,
                                20,
                                Style.Both);

label1.setLabelColor(ColorPalette.WHITE);
label1.setOutlineColor(RosePalette.REDWOOD);
label1.setShader(fractions, c);
s1.addSliceLabel(label1);
```

4.2.3.2.Pie 3D Border Label



```
/**
* Snippet code: pie border label
*/

// LABELS
float[] fractions = { 0f, 0.3f, 0.7f, 1f };
Color[] c = { new Color(0, 0, 0, 20), new Color(0, 0, 0, 150),
                                new Color(0, 0, 0, 150), new Color(0, 0, 0, 20) };

Donut3DBorderLabel label1 = Donut3DToolkit.createBorderLabel("Symbian",
                                RosePalette.COALBLACK,
                                InputFonts.getNeuropol(12),
                                20,
                                Style.Both);
label1.setLinkColor(RosePalette.LEMONPEEL);
//label1.setLinkExtends(30);
label1.setLabelColor(ColorPalette.WHITE);
label1.setOutlineColor(Color.BLACK);
label1.setShader(fractions, c);
s1.addSliceLabel(label1);
```

4.2.3.3.Pie 3D Path Label



```
/**
* Snippet code: pie path label
*/

float[] fractions = { 0f, 0.3f, 0.7f, 1f };
Color[] c = { new Color(0, 0, 0, 20), new Color(0, 0, 0, 150),
                         new Color(0, 0, 0, 150), new Color(0, 0, 0, 20) };

Donut3DPathLabel ppl = new Donut3DPathLabel(TextPosition.Right, "JenSoft SW2D
Framework");
ppl.setPathSide(PathSide.Below);
ppl.setLabelFont(InputFonts.getNeuropol(12));
ppl.setLabelColor(RosePalette.MANDARIN);
ppl.setDivergence(2);
ppl.setOffsetRight(0);
ppl.setLockReverse(true);
ppl.setAutoReverse(false);
s1.addSliceLabel(ppl);

ppl = new Donut3DPathLabel(TextPosition.Middle, "SW2D");
ppl.setPathSide(PathSide.Below);
ppl.setFacetPathName(Donut3DFacetPathName.EndLineTop);
ppl.setLabelFont(InputFonts.getSansation(12));
ppl.setLabelColor(RosePalette.CORALRED);
ppl.setDivergence(2);
ppl.setOffsetLeft(0);
ppl.setOffsetRight(0);
//ppl.setLockReverse(true);
//ppl.setAutoReverse(false);
s1.addSliceLabel(ppl);
```

4.2.4. Pie 3D Paint

Pie 3D has only one default paint which takes the responsibility to fill pie 3D slices and create shadow layer on top face, inner and outer face. These options can be enabled or disabled and alpha for these options can be set.

```
/**
* Snippet code: pie 3D paint properties
*/


Donut3DDefaultPaint paint = new Donut3DDefaultPaint();
// set shadow incidence angle degree
paint.setIncidenceAngleDegree(120);
paint.setPaintTopEffect(true);
paint.setAlphaTop(0.9f);
// other face effect on inner and outer face
paint.setPaintInnerEffect(true);
paint.setPaintOuterEffect(true);
// use alpha 0.7 alpha value to fill section
paint.setAlphaFill(0.7f);
// alpha value for each face effect
// paint.setAlphaTop(alphaTop);
// paint.setAlphaOuter(alphaOuter);
// paint.setAlphaInner(alphaInner);

donut3d.setDonut3DPaint(paint);
```

4.2.5. Pie 3D Listener

Pie 3D listener handles fire events from pie 3D slices. Slice enter, exit, press, released, clicked can be observed. If you need to handle specific behaviors when these events occur, you have to register a pie 3D or Donut3D listener.

```java
/**
* Snippet code: register a donut 3D listener on donut 3D view
* or in the donut 3D plug in, the addDonut3DListener method in donut3D view
* is a delegate method from donut 3D plug in)
*/


donut3dPlugin.addDonut3DListener(new Donut3DListener() {

        @Override
        public void donut3DSliceReleased(Donut3DEvent e) {
                System.out.println("donut3DSectionReleased :"
                                        + e.getDonut3DSlice().getName());

        }

        @Override
        public void donut3DSlicePressed(Donut3DEvent e) {
                System.out.println("donut3DSectionPressed :"
                                        + e.getDonut3DSlice().getName());

        }

        @Override
        public void donut3DSliceExited(Donut3DEvent e) {
                System.out.println("donut3DSectionExited :"
                                        + e.getDonut3DSlice().getName());

        }

        @Override
        public void donut3DSliceEntered(Donut3DEvent e) {
                System.out.println("donut3DSectionEntered :"
                                        + e.getDonut3DSlice().getName());

        }

        @Override
        public void donut3DSliceClicked(Donut3DEvent e) {
                System.out.println("donut3DSectionClicked :"
                                        + e.getDonut3DSlice().getName());

        }
});
```

4.2.6. Pie 3D Animator

4.2.6.1.Pie 3D Divergence Animator

Pie 3D divergence runs a divergence animator from current divergence to the target divergence.

```
/**
* Snippet code: register the divergence animator with the 40 divergence
* value
*/


view.addDonutAnimator(new Donut3DDivergenceAnimator(60, 2));
```

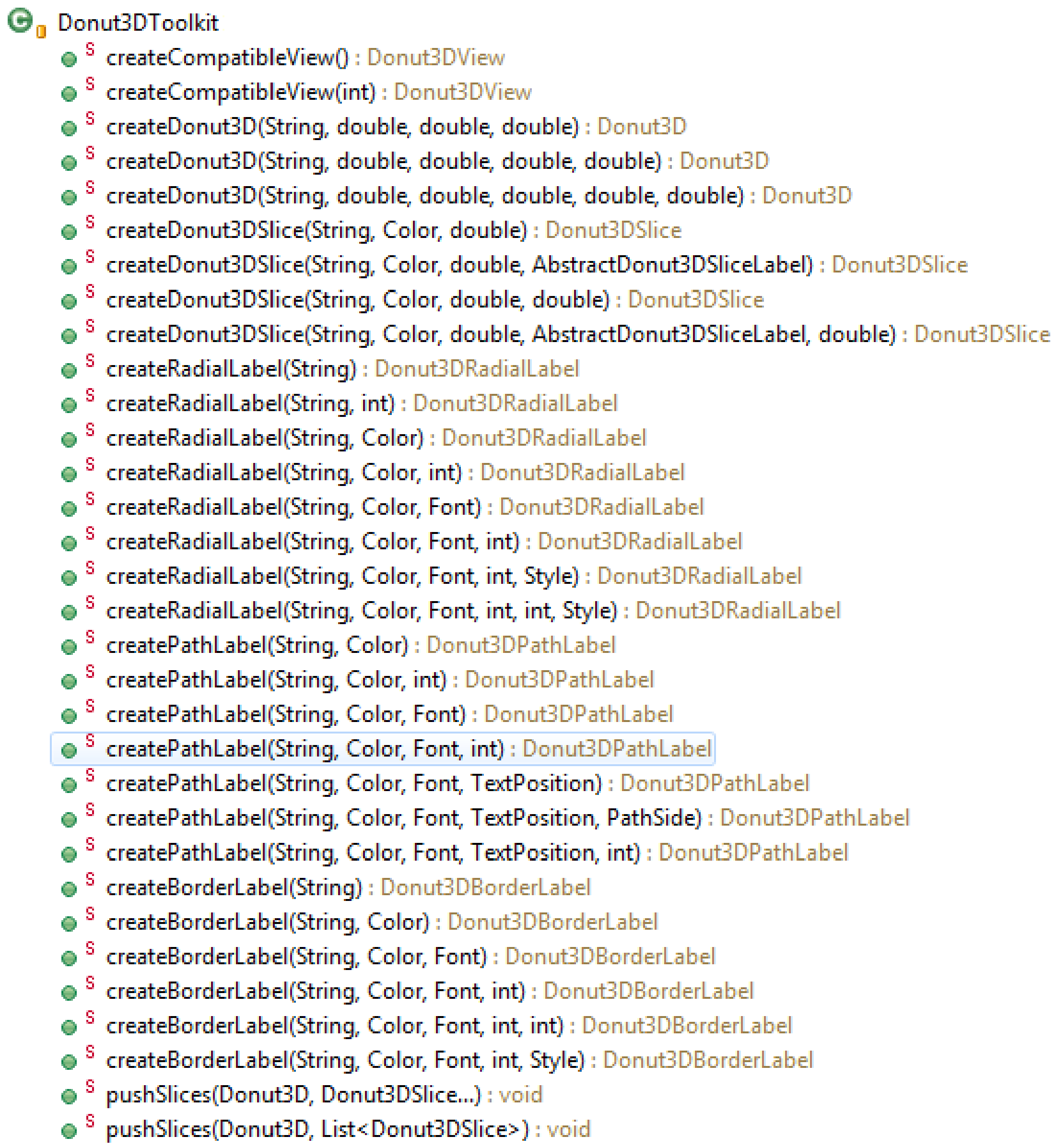
4.2.6.2.Pie 3D Label Animator

Pie 3D label animator show and hide label on a donut 3D slice.

```
/**
* Snippet code: register the labels animator
*
* Assume that slices s1,s2,s3,s4 and slices labels label1,label2,label3,
* label4 already exist.
* Animator can be register on compatible view or on the pie 3D plug in
*/

view.addDonutAnimator(new Donut3DLabelAnimator(s1,label1));
view.addDonutAnimator(new Donut3DLabelAnimator(s2,label2));
view.addDonutAnimator(new Donut3DLabelAnimator(s3,label3));
view.addDonutAnimator(new Donut3DLabelAnimator(s4,label4));
```

4.2.7. Pie 3D Toolkit

Here is the *Donut3DToolkit* outline. It provides all static factory method to create donut3D compatible view, donut3D, donut3D slice, donut3D bound label, donut3D radial label, donut3D border label, donut3D slices pushing.



```
Donut3DToolkit
    createCompatibleView() : Donut3DView
    createCompatibleView(int) : Donut3DView
    createDonut3D(String, double, double, double) : Donut3D
    createDonut3D(String, double, double, double, double) : Donut3D
    createDonut3D(String, double, double, double, double, double) : Donut3D
    createDonut3DSlice(String, Color, double) : Donut3DSlice
    createDonut3DSlice(String, Color, double, AbstractDonut3DSliceLabel) : Donut3DSlice
    createDonut3DSlice(String, Color, double, double) : Donut3DSlice
    createDonut3DSlice(String, Color, double, AbstractDonut3DSliceLabel, double) : Donut3DSlice
    createRadialLabel(String) : Donut3DRadialLabel
    createRadialLabel(String, int) : Donut3DRadialLabel
    createRadialLabel(String, Color) : Donut3DRadialLabel
    createRadialLabel(String, Color, int) : Donut3DRadialLabel
    createRadialLabel(String, Color, Font) : Donut3DRadialLabel
    createRadialLabel(String, Color, Font, int) : Donut3DRadialLabel
    createRadialLabel(String, Color, Font, int, Style) : Donut3DRadialLabel
    createRadialLabel(String, Color, Font, int, int, Style) : Donut3DRadialLabel
    createPathLabel(String, Color) : Donut3DPathLabel
    createPathLabel(String, Color, int) : Donut3DPathLabel
    createPathLabel(String, Color, Font) : Donut3DPathLabel
    createPathLabel(String, Color, Font, int) : Donut3DPathLabel
    createPathLabel(String, Color, Font, TextPosition) : Donut3DPathLabel
    createPathLabel(String, Color, Font, TextPosition, PathSide) : Donut3DPathLabel
    createPathLabel(String, Color, Font, TextPosition, int) : Donut3DPathLabel
    createBorderLabel(String) : Donut3DBorderLabel
    createBorderLabel(String, Color) : Donut3DBorderLabel
    createBorderLabel(String, Color, Font) : Donut3DBorderLabel
    createBorderLabel(String, Color, Font, int) : Donut3DBorderLabel
    createBorderLabel(String, Color, Font, int, int) : Donut3DBorderLabel
    createBorderLabel(String, Color, Font, int, Style) : Donut3DBorderLabel
    pushSlices(Donut3D, Donut3DSlice...) : void
    pushSlices(Donut3D, List<Donut3DSlice>) : void
```

4.3. Donut 2D Chart

```
package com.jensoft.sw2d.core.plugin.donut2d;
```

The donut geometry is defined by it center, inner radius, outer radius (radius in pixels). The donut 2D center location can be projected in device component system coordinate and the window user system coordinate which became a donut 2D vector. (*Donut2DNature* enumeration). Donut2D and related objects can be creating manually or with the *Donut2DToolkit.*

4.3.1. Donut2D

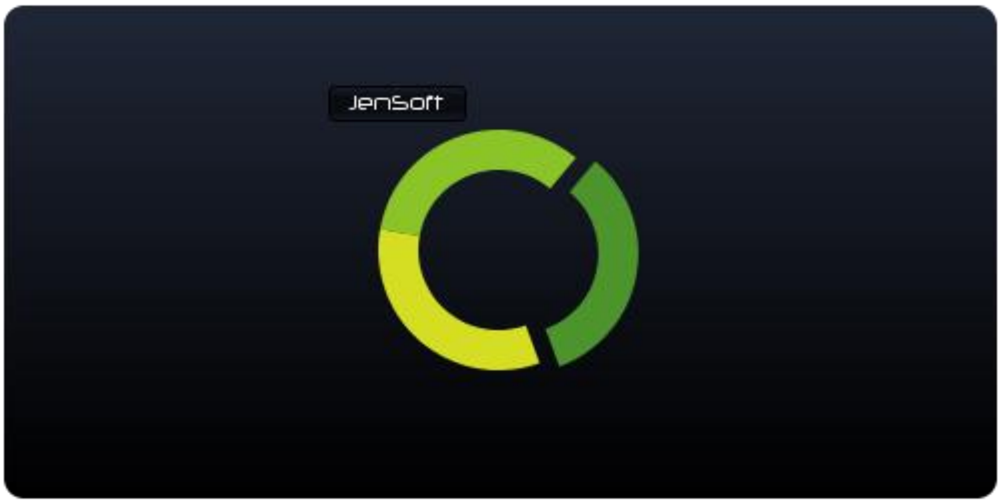
```java
/**
* Snippet code: create donut2D manually
*/
final View2D view = new View2D(0);

Window2D w2d = new Window2D(-2, 2, -2, 2);
view.registerWindow2D(w2d);

w2d.setThemeColor(Color.WHITE);
w2d.registerPlugin(new OutlinePlugin(RosePalette.MELON));

w2d.registerPlugin(new ZoomWheelPlugin());

TranslatePlugin translatePlugin = new TranslatePlugin();
translatePlugin.registerContext(new TranslateDefaultDeviceContext());
w2d.registerPlugin(translatePlugin);

Donut2DPlugin donut2DPlugin = new Donut2DPlugin();
w2d.registerPlugin(donut2DPlugin);

Donut2D donut1 = new Donut2D();
donut1.setNature(Donut2DNature.Donut2DUser);
donut1.setCenterX(0);
donut1.setCenterY(0);
donut1.setInnerRadius(70);
donut1.setOuterRadius(100);
donut1.setStartAngleDegree(0);

donut2DPlugin.addDonut(donut1);


/**
* Snippet code: create donut2D view and donut 2D with tool kit
* A Donut 2D compatible view (Donut2DView) already embeds symmetric window
* and a donut 2D plug in instance.
*/
Donut2DView view = Donut2DToolkit.createCompatibleView();

/**
* create donut and add in the compatible view.
*/
Donut2D donut2d = Donut2DToolkit.createDonut2D("myDonut2D", 80, 160);
view.addDonut2D(donut2d);
```

4.3.2. Donut 2D Slicing

```
/**
* Snippet code: create donut2D slice manually
*/

Donut2DSlice s1 = new Donut2DSlice("s1", Spectral.SPECTRAL_RED.brighter());
s1.setValue(20.0);


/**
* create donut slice with tool kit
*/
Donut2DSlice s1 = Donut2DToolkit.createDonut2DSlice("s1",FilPalette.BLUE1, 20, 0);
```

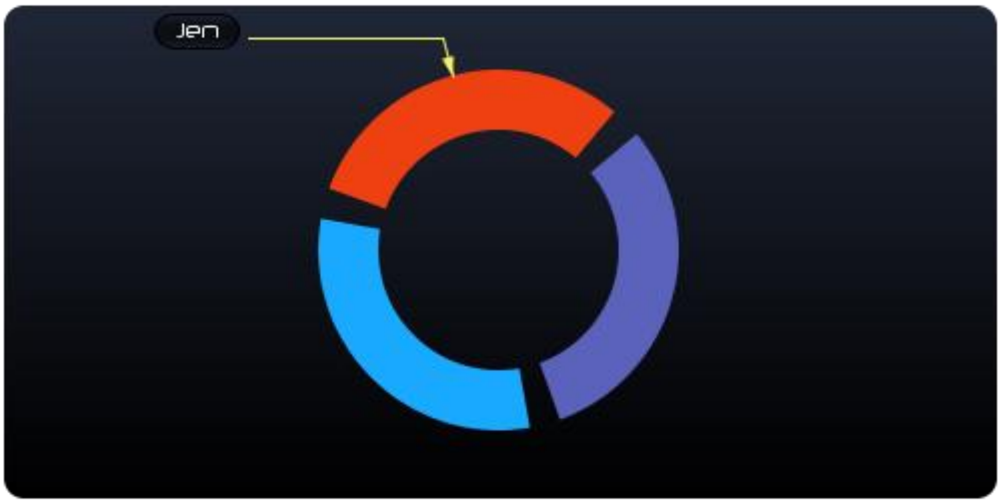
4.3.3. Donut 2D Label

4.3.3.1.Donut 2D Radial Label



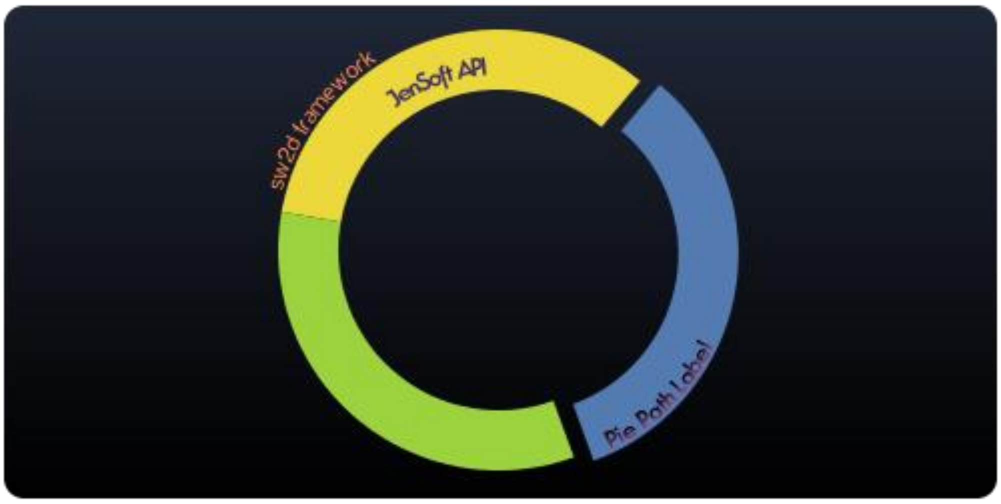
```
/**
* Snippet code: create donut2D radial slice label
*/

float[] fractions = { 0f, 0.3f, 0.7f, 1f };
Color[] c = { new Color(0, 0, 0, 20), new Color(0, 0, 0, 150),
                          new Color(0, 0, 0, 150), new Color(0, 0, 0, 20) };

Donut2DRadialLabel label1 = Donut2DToolkit.createRadialLabel("JenSoft",
                          RosePalette.COALBLACK, InputFonts.getNeuropol(12),
                          20,Style.Both);

label1.setLabelColor(ColorPalette.WHITE);
label1.setOutlineColor(Color.BLACK);
label1.setShader(fractions, c);
s1.addSliceLabel(label1);
```

4.3.3.2.Donut 2D Border Label



```
/**
* Snippet code: create donut2D border slice label
*/

float[] fractions = { 0f, 0.3f, 0.7f, 1f };
Color[] c = { new Color(0, 0, 0, 20), new Color(0, 0, 0, 150),
                            new Color(0, 0, 0, 150), new Color(0, 0, 0, 20) };

Donut2DBorderLabel label1 = Donut2DToolkit.createBorderLabel("Symbian",
                            RosePalette.COALBLACK, InputFonts.getNeuropol(12), 20,
                            Style.Both);
label1.setLinkColor(RosePalette.LEMONPEEL);
label1.setLabelColor(ColorPalette.WHITE);
label1.setOutlineColor(Color.BLACK);
label1.setShader(fractions, c);
s1.addSliceLabel(label1);
```

4.3.3.3.Donut 2D Path Label



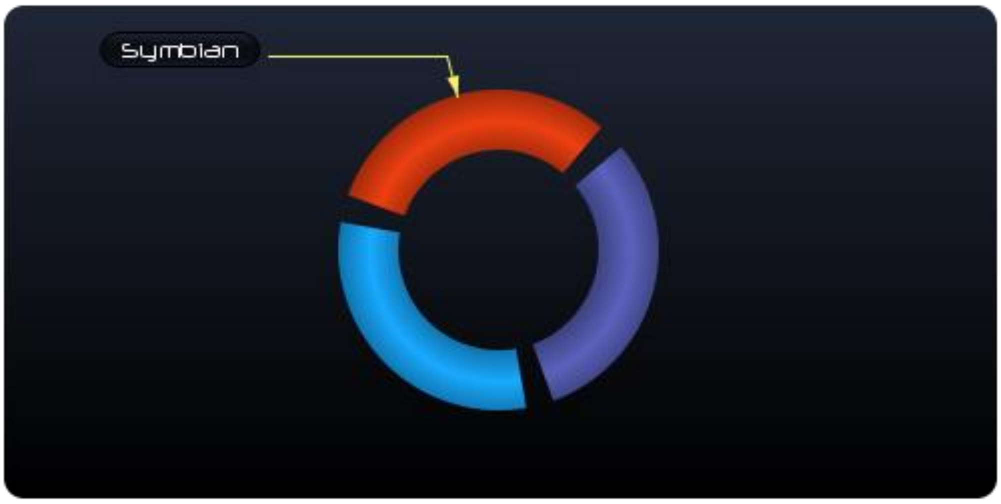
```
/**
* Snippet code: create donut2D path slice label
*/

float[] fractions = { 0f, 0.3f, 0.7f, 1f };
Color[] c = { new Color(0, 0, 0, 20), new Color(0, 0, 0, 150),
                        new Color(0, 0, 0, 150), new Color(0, 0, 0, 20) };

Donut2DPathLabel ppl = new Donut2DPathLabel(TextPosition.Right, "sw2d framework");
ppl.setPathSide(PathSide.Below);
ppl.setLabelFont(InputFonts.getYorkville(12));
ppl.setLabelColor(RosePalette.MANDARIN);
ppl.setDivergence(2);
s1.addSliceLabel(ppl);

Donut2DPathLabel ppl12 = Donut2DToolkit.createPathLabel("JenSoft API",
                        RosePalette.INDIGO,
                        InputFonts.getFont(InputFonts.NO_MOVE, 12),
                        TextPosition.Middle);
s1.addSliceLabel(ppl12);



Donut2DPathLabel ppl3 = new Donut2DPathLabel(TextPosition.Left, "Pie Path Label",
                        TangoPalette.CHAMELEON2.darker());
float[] fractions = { 0f,1f };
Color[] colors = { Color.BLACK,RosePalette.AMETHYST };
ppl3.setLabelFont(InputFonts.getNoMove(12));
ppl3.setTextShader(fractions, colors);
ppl3.setPathSide(PathSide.Above);
ppl3.setDivergence(2);

s3.addSliceLabel(ppl3);
```

4.3.4. Donut2D Fill

4.3.4.1.Default Fill

The default donut 2D fill use the slice theme color to fill slice.

4.3.4.2.Radial Fill



```
/**
* Snippet code: create donut2D radial fill
*/

Donut2DPlugin donut2DPlugin = new Donut2DPlugin();
w2d.registerPlugin(donut2DPlugin);

Donut2D donut1 = new Donut2D();
donut1.setNature(Donut2DNature.Donut2DUser);
donut1.setCenterX(0);
donut1.setCenterY(0);
donut1.setInnerRadius(50);
donut1.setOuterRadius(80);
donut1.setStartAngleDegree(50);
donut1.setExplose(10);

Donut2DRadialFill donut2DRadialFill = new Donut2DRadialFill();
donut1.setDonut2DFill(donut2DRadialFill);
```

The explose property is not divergence, only a fragment slice degree which is subtracted from the slice degree proportion value.

4.3.5. Donut 2D Listener

With the donut 2D listener you can drive effect or action when an event occurs from a donut2D slice.

```
/**
* Snippet code: create donut2D listener to make action code on slice event
*/

donut2DPlugin.addDonut(donut1);

donut2DPlugin.addDonut2DListener(new Donut2DListener() {

        @Override
        public void donut2DSliceReleased(Donut2DEvent e) {
                // TODO Put Action code here
        }

        @Override
        public void donut2DSlicePressed(Donut2DEvent e) {
                // TODO Put Action code here
        }

        @Override
        public void donut2DSliceExited(Donut2DEvent e) {
                // TODO Put Action code here
        }

        @Override
        public void donut2DSliceEntered(Donut2DEvent e) {
                // TODO Put Action code here
        }

        @Override
        public void donut2DSliceClicked(Donut2DEvent e) {
                // TODO Put Action code here
        }
});
```

4.3.6. Donut 2DAnimator

4.3.6.1.Donut2D Divergence Animator

Donut2D divergence runs a divergence animator from current divergence to the target divergence.

```
/**
* Snippet code: register the divergence animator with the 40 divergence
* value
*/


view.addDonutAnimator(new Donut2DDivergenceAnimator(60, 2));
```

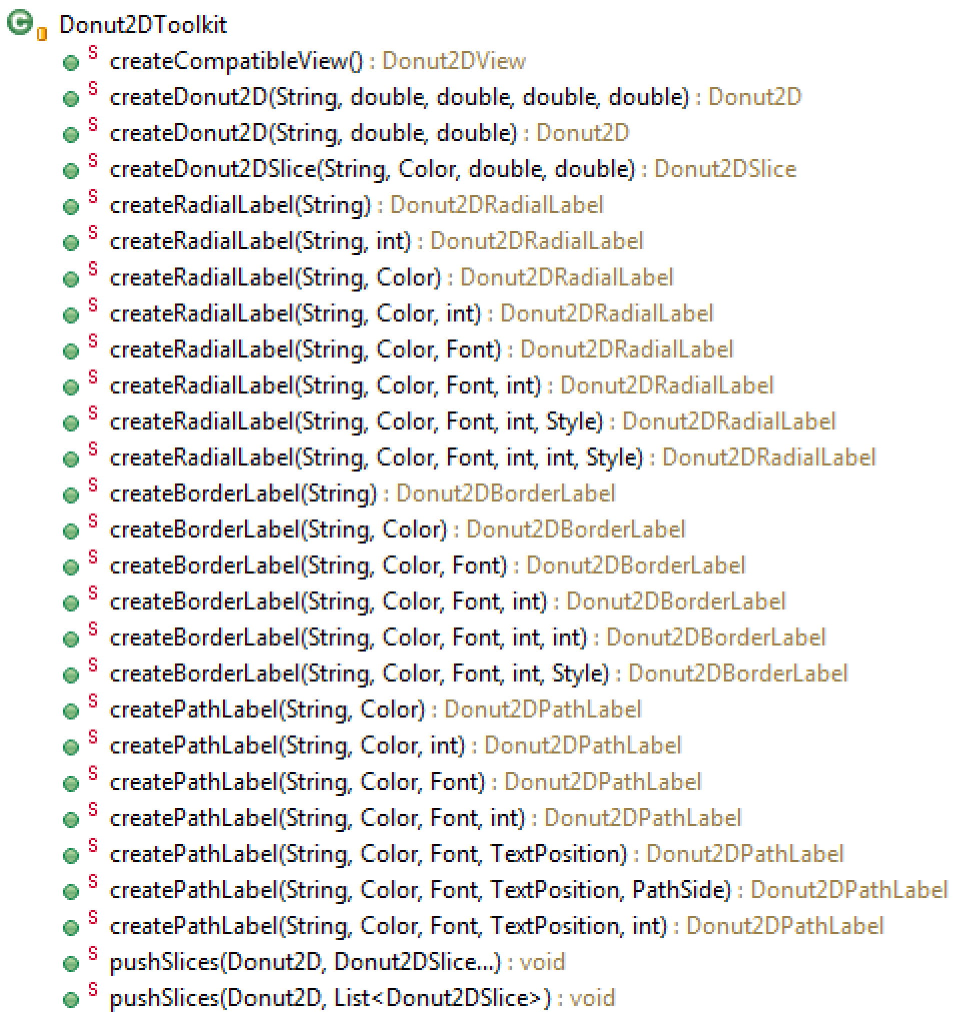
4.3.6.2.Donut2D Label Animator

Donut 2D label animator show and hide label on a donut 2D slice.

```
/**
* Snippet code: register the labels animator
*
* Assume that slices s1,s2,s3,s4 and slices labels label1,label2,label3,
* label4 already exist.
* Animator can be register on compatible view or on the donut2D plug in
*/

view.addDonutAnimator(new Donut2DLabelAnimator(s1,label1));
view.addDonutAnimator(new Donut2DLabelAnimator(s2,label2));
view.addDonutAnimator(new Donut2DLabelAnimator(s3,label3));
view.addDonutAnimator(new Donut2DLabelAnimator(s4,label4));
```

4.3.7. Donut2D Toolkit

Here is the *Donut2DToolkit* outline. It provides all static factory method to create donut2D compatible view, donut2D, donut2D slice, donut2D radial label, donut2D border label, donut2D slices pushing.



```
Donut2DToolkit
    createCompatibleView() : Donut2DView
    createDonut2D(String, double, double, double, double) : Donut2D
    createDonut2D(String, double, double) : Donut2D
    createDonut2DSlice(String, Color, double, double) : Donut2DSlice
    createRadialLabel(String) : Donut2DRadialLabel
    createRadialLabel(String, int) : Donut2DRadialLabel
    createRadialLabel(String, Color) : Donut2DRadialLabel
    createRadialLabel(String, Color, int) : Donut2DRadialLabel
    createRadialLabel(String, Color, Font) : Donut2DRadialLabel
    createRadialLabel(String, Color, Font, int) : Donut2DRadialLabel
    createRadialLabel(String, Color, Font, int, Style) : Donut2DRadialLabel
    createRadialLabel(String, Color, Font, int, int, Style) : Donut2DRadialLabel
    createBorderLabel(String) : Donut2DBorderLabel
    createBorderLabel(String, Color) : Donut2DBorderLabel
    createBorderLabel(String, Color, Font) : Donut2DBorderLabel
    createBorderLabel(String, Color, Font, int) : Donut2DBorderLabel
    createBorderLabel(String, Color, Font, int, int) : Donut2DBorderLabel
    createBorderLabel(String, Color, Font, int, Style) : Donut2DBorderLabel
    createPathLabel(String, Color) : Donut2DPathLabel
    createPathLabel(String, Color, int) : Donut2DPathLabel
    createPathLabel(String, Color, Font) : Donut2DPathLabel
    createPathLabel(String, Color, Font, int) : Donut2DPathLabel
    createPathLabel(String, Color, Font, TextPosition) : Donut2DPathLabel
    createPathLabel(String, Color, Font, TextPosition, PathSide) : Donut2DPathLabel
    createPathLabel(String, Color, Font, TextPosition, int) : Donut2DPathLabel
    pushSlices(Donut2D, Donut2DSlice...) : void
    pushSlices(Donut2D, List<Donut2DSlice>) : void
```

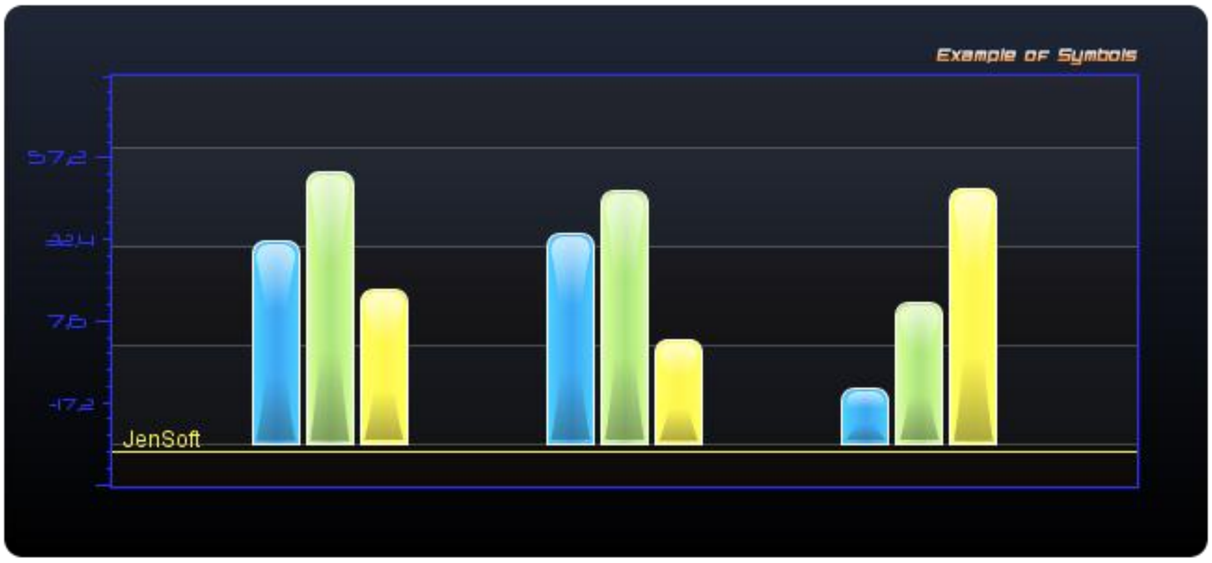
4.3.8. Multiple Donut 2D

Multiple donut 2D can be achieve. Outer radius for the inner donut should be lesser than the inner radius for the outer donut2D. The building donut2D strategy is the same than for one donut.

4.4. Symbols

```
package com.jensoft.sw2d.core.plugin.symbol;
```

Here is the typical produced component with the symbol plug-in. it can be combo bar, combo stacked bar, point symbol or line curve symbol.



Symbol is a component which has only one metric dimension, the other is symbolic. Through the symbolic dimension, the bar is seen as a component and is positioned relative to another bar and constraints that can be added.

The most common symbol is the bar chart or bar graph which is a chart with rectangular bars with lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally. Bar charts are used for marking clear data which has learned values. Some examples of discontinuous data include 'shoe size' or 'eye color', for which you would use a bar chart. In contrast, some examples of continuous data would be 'height' or 'weight'. A bar chart is very useful if you are trying to record certain information whether it is continuous or not continuous data. Bar charts also look a lot like a histogram. They are often mistaken for each other.

4.4.1. Symbol Plug-in

The symbol plug-in knows how to design bar graph, histogram, symbol point, or symbol curve. All of these symbols has only one metrics dimension.

```
/**
* Snippet code: create a symbol plug in and register it into the window
*
* Horizontal symbol needs only x metrics (defines) dimension which is the
* projected dimension for the symbol. Y user dimension is ignored, symbol
* is lay out as a component regarding the layer constraints.
*
*/

Window2D w2d = new Window2D(-100, 120, 0, 0);
view.registerWindow2D(w2d);

SymbolPlugin barPlugin = new SymbolPlugin();
barPlugin.setNature(SymbolNature.Horizontal);
w2d.registerPlugin(barPlugin);
```

The symbol plug in constructor needs to know the symbol nature which is vertical or horizontal. If the symbol is vertical, it means that the x dimension is the symbolic dimension and the y dimension is the metrics dimension. If the symbol is horizontal, the dimension y is the symbolic dimension and x the metrics dimension.

The symbol view is a compatible view which embeds a window and a symbol plug-in. the view constructor needs the symbol nature (vertical or horizontal nature which is defines by *SymbolNature* enumeration, the symbol nature defines the metrics dimension x or y) and the bound of the unique metrics dimension (y for vertical symbols and x for horizontal).

```
/**
* Snippet code: create a symbol a compatible symbol view
*
* vertical symbols with the window y bound -42 to 82
*/
SymbolView v = SymbolToolkit.createView(SymbolNature.Vertical,-42,82);
```

On this view, like symbol plug-in, you have to register layer that host symbol which will be render in view.

```
/**
* Snippet code: register layer in view
*/
BarSymbolLayer barLayer = new BarSymbolLayer();
view.addLayer(barLayer);
```

4.4.2. Bar Symbol Components

Bar symbols is a component with a single scalar dimension, the other one is only symbolic (this dimension does not refer to this related window user system coordinate. The location of symbol is relative to the others symbols) If the symbol inflates on y dimension, the symbol is vertical else if the symbol inflates along x dimension, the symbol is horizontal. Through the symbolic dimension, the bar is seen as a component and is positioned relative to another bar and constraints that can be added. It has geometry and rendering properties. In the other dimension, the bar symbol is defined by its base and top. It is either up or down.

There are different types of bar symbol. The abstract definition of a symbol is the *SymbolComponent* which has inherits bar symbols class.

- *BarSymbol* which is the simplest bar type symbol.
- *StackedBarSymbol* which is a composed symbol that hosts stack symbol.
- *BarSymbolGroup* which embeds sub symbols and properties for all children component.
- *Glue* is a stretchable filler component which is used to lay out symbol within view.
- *Strut* is a fixed size box filler component which is used to lay out symbol within view.

4.4.2.1.Bar Layer

The bar layer is the bar symbol container that hosts bar symbols. Each bar symbol has to be registered into the layer with relative constraints.

Through the symbolic dimension, symbols should be lay out into container such as bar layer (layer is for the entire view) or group (embed sub symbols, cf Bar Symbol group). *SymbolComponent* gives invisible components as filler. Each symbol controlled by the plug in butts up against its neighboring symbols and device bound. If you want space between symbols, you can insert invisible symbols to provide the space. You can create invisible components with the help of the *SymbolComponent* class.

```
/**
* Snippet code: lay out symbol in bar symbol layer with classic
* distribution (glue on side left and side right, strut between symbol)
*/
BarSymbolLayer barLayer = new BarSymbolLayer();
view.addLayer(barLayer);

barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
barLayer.addSymbol(symbol1);
barLayer.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 10));
barLayer.addSymbol(symbol2);
barLayer.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 10));
barLayer.addSymbol(symbol3);
barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
```

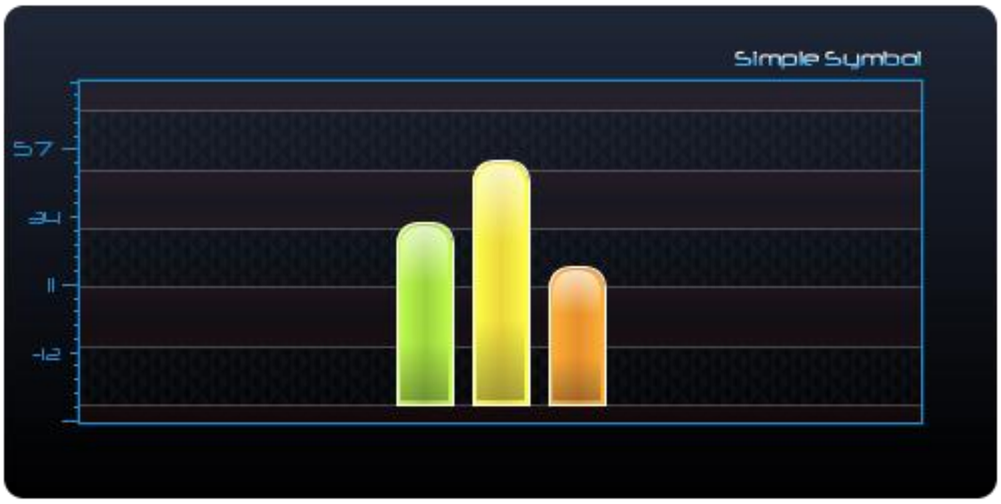
Glue and Strut are non visible symbol which contributes to the lay out. It's a very simplest way to lay out symbol like you needs. Layer lays out on the entire device all of symbols children as a flow of symbol which has fixed size in pixel. It's the symbolic dimension.

4.4.2.2.Simple Bar Symbol

A simple bar symbol can be created from scratch or with the symbol tool kit which provides some static methods to create symbols related components. A bar symbol is defined by a base and the ascent or descent value. Ascent symbol inflates value along positive dimension and descent symbol inflates along negative dimension. You can choose options like symbol geometry (rounded rectangle or rectangle), draw, fill and effect and labels (axis label or symbol label)

```
/**
* Snippet code: create a simple bar symbol from scratch
*
*/

BarSymbol b1 = new BarSymbol();
b1.setThemeColor(PetalPalette.PETAL1_LC);
b1.setThickness(32);
b1.setBase(-30);
b1.setAscentValue(62);
b1.setName("b1");
b1.setSymbol("bar 1");
b1.setMorpheStyle(MorpheStyle.Round);
b1.setBarDraw(new BarDefaultDraw());
b1.setBarFill(new BarFill1());
b1.setBarEffect(new BarEffect1());
b1.setBarDraw(new BarDefaultDraw(Color.WHITE));
b1.setRound(10);


/**
* Snippet code: create a simple bar symbol with symbol toolkit
*
* The static method create symbol with default configuration for draw,
* fill, effect and style
*/
BarSymbol b1g2 = SymbolToolkit.createSymbol("b1g1", blue,
                                SymbolInflate.Ascent, 64);
```

Here is the example of simple bar symbols which are lay out into a bar layer with the following sequence: glue, symbol1, strut (10), symbol2, strut (10), symbol3, glue.

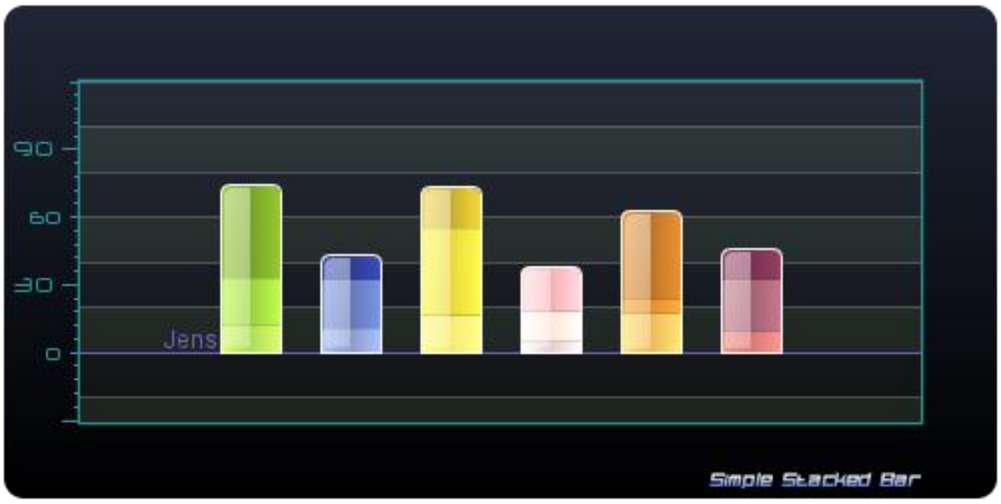


4.4.2.3.Stacked Bar Symbol

A stacked bar chart or bar graph is a chart with rectangular bars with lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally. Stacked bar graph is a graph that is used to compare the parts to the whole. The bars in a stacked bar graph are divided into categories. Each bar represents a total.

Bar charts are used for marking clear data which has learned values. Some examples of discontinuous data include 'shoe size' or 'eye color', for which you would use a bar chart. In contrast, some examples of continuous data would be 'height' or 'weight'. A bar chart is very useful if you are trying to record certain information whether it is continuous or not continuous data. Bar charts also look a lot like a histogram. They are often mistaken for each other.

He is an example of stacked bar.



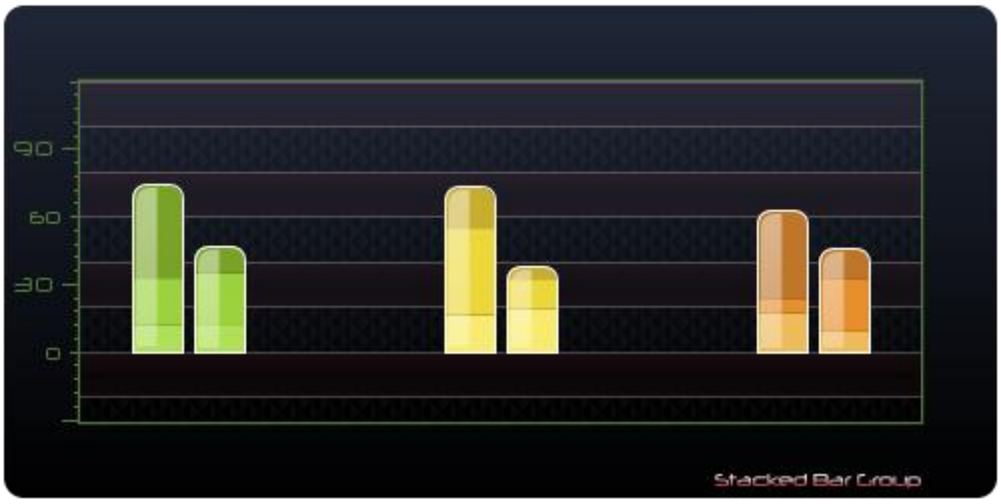
StackedBarSymbol is also a symbol which inherits from *BarSymbol. Stack* which is also a symbol is added as child to the stacked symbol. You have to create stacked symbol which commons properties like thickness, base value, fill, draw, effect, etc. and a ascent or descent value. Stacks are part of this symbol and are added to stacked host symbol with a value (stack value is normalized, put only relative value is valid)

```
/**
* Snippet code: create stacked symbol
*
*/
// create symbol from scratch (or use tool kit)
StackedBarSymbol stackeSymbol = new StackedBarSymbol();
stackeSymbol.setThemeColor(new Color(255, 255, 255));
stackeSymbol.setBase(30);
stackeSymbol.setThickness(16);
stackeSymbol.setAscentValue(73);
stackeSymbol.setMorpheStyle(MorpheStyle.Round);
stackeSymbol.setBarDraw(barDraw);
stackeSymbol.setBarFill(new BarFill2());
// and needs properties for a bar symbol…


// create stack with tool kit
Stack stack1 = SymbolToolkit.createStack("orange", orange, 12);
Stack stack2 = SymbolToolkit.createStack("green", green, 28);
Stack stack3 = SymbolToolkit.createStack("blue", blue, 13);
Stack stack3 = SymbolToolkit.createStack("pink", pink, 9);

// register stacks into stacked host symbol
SymbolToolkit.pushStacks(stackeSymbol, stack1, stack2, stack3, stack4);
```

4.4.2.4.Bar Symbol Group

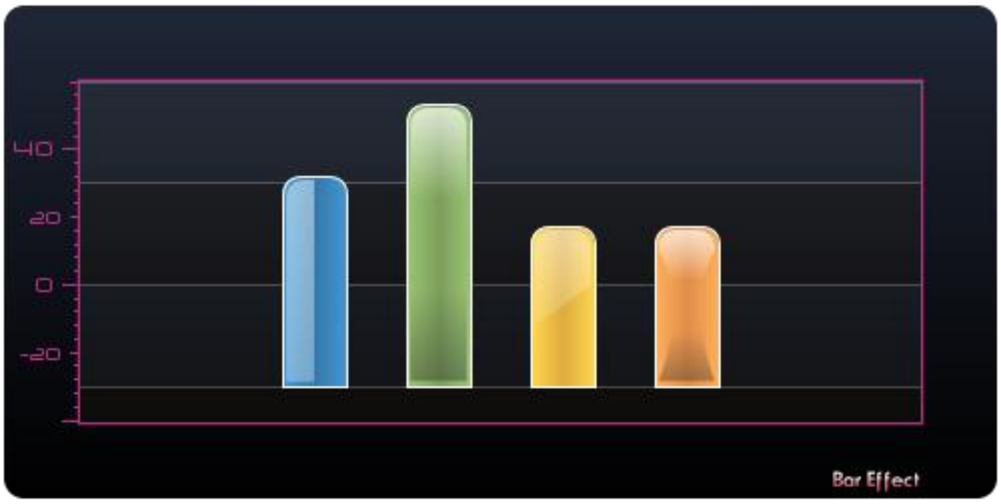
A bar symbol group *BarSymbolGroup* allows to lay out some children symbols with common property set without to have register property for each registered child symbol. Groups are quite different from layer because glue cannot be used in group. Only strut filler and symbol are allowed. Moreover group shared property like fill, draw, base value, etc.

A *BarSymbolGroup* which inherits *BarSymbol* have an axis label that can be set. It's interesting to create symbol group when you want create a symbol set with same properties and make a group legend.



4.4.2.5.Bar Symbol Effect

JenSoft API provides some common effects shown as the view below.



```
/**
* Snippet code: create show the bar effect
*
*/

// create compatible bar view
SymbolView view = SymbolToolkit.createView(SymbolNature.Vertical, -40,
                          60);

// create symbols from scratch or use BarFactory.
// BarFactory.createSymbol(nameSymbol, themeColor, symbolInflate,
// symbolInflateValue)
// BarFactory.createSymbol(nameSymbol, themeColor, thickness, base,
// symbolInflate, symbolInflateValue)
// etc.
BarSymbol b1 = new BarSymbol();
b1.setThemeColor(PetalPalette.PETAL1_LC);
b1.setThickness(32);
b1.setBase(-30);
b1.setAscentValue(62);
b1.setName("b1");
b1.setSymbol("bar 1");
b1.setMorpheStyle(MorpheStyle.Round);
b1.setBarDraw(new BarDefaultDraw());
b1.setBarFill(new BarFill1());
b1.setBarEffect(new BarEffect1());
b1.setBarDraw(new BarDefaultDraw(Color.WHITE));
b1.setRound(10);
// bar 2
BarSymbol b2 = new BarSymbol();
b2.setThemeColor(PetalPalette.PETAL2_LC);
b2.setThickness(32);
b2.setBase(-30);
b2.setAscentValue(83);
b2.setName("b2");
b2.setSymbol("bar 2");
b2.setMorpheStyle(MorpheStyle.Round);
b2.setBarDraw(new BarDefaultDraw());
b2.setBarFill(new BarFill1());
b2.setBarEffect(new BarEffect2());
b2.setBarDraw(new BarDefaultDraw(Color.WHITE));
b2.setRound(10);

// bar 3
BarSymbol b3 = new BarSymbol();
b3.setThemeColor(PetalPalette.PETAL3_LC);
b3.setThickness(32);
b3.setBase(-30);
b3.setAscentValue(47);
b3.setName("b3");
b3.setSymbol("bar 3");
b3.setMorpheStyle(MorpheStyle.Round);
b3.setBarDraw(new BarDefaultDraw());
b3.setBarFill(new BarFill1());
b3.setBarEffect(new BarEffect3());
```

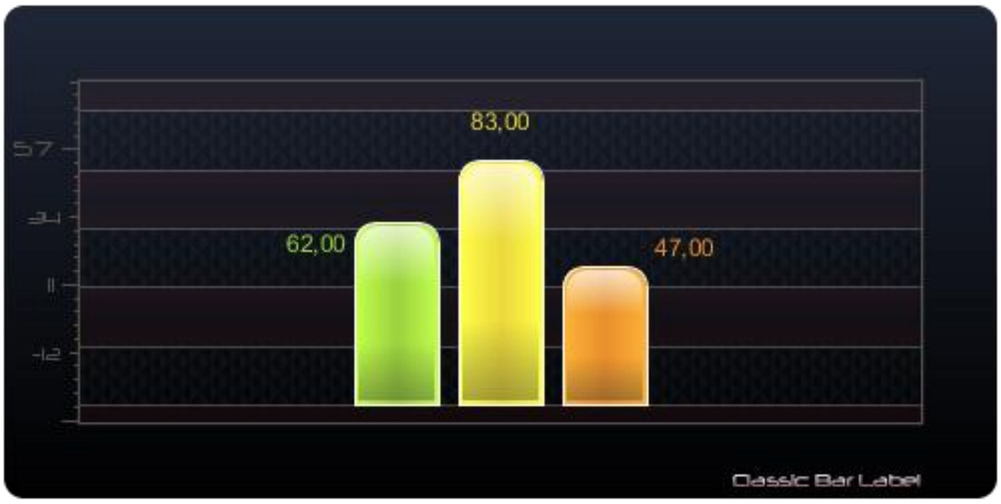
```
b3.setBarDraw(new BarDefaultDraw(Color.WHITE));
b3.setRound(10);
// bar 4
BarSymbol b4 = new BarSymbol();
b4.setThemeColor(PetalPalette.PETAL4_LC);
b4.setThickness(32);
b4.setBase(-30);
b4.setAscentValue(47);
b4.setName("b4");
b4.setSymbol("bar 3");
b4.setMorpheStyle(MorpheStyle.Round);
b4.setBarDraw(new BarDefaultDraw());
b4.setBarFill(new BarFill1());
b4.setBarEffect(new BarEffect4());
b4.setBarDraw(new BarDefaultDraw(Color.WHITE));
b4.setRound(10);

// HORIZONTAL BAND DECORATOR
DynamicStripeManager dbm = new DynamicStripeManager(
             StripeOrientation.Horizontal, 0, 30);
StripePlugin bandLayout = new StripePlugin(dbm);
StripePalette bp = new StripePalette();
bp.addPaint(new Color(255, 255, 255, 40));
bp.addPaint(ColorPalette.alpha(TangoPalette.ORANGE3, 40));
dbm.setBandPalette(bp);
bandLayout.setAlpha(0.3f);
view.registerPlugin(bandLayout);

// HORIZONTAL AND VERTICAL GRID DECORATOR
DynamicGridManager dgm = new DynamicGridManager(
                   GridOrientation.Horizontal, 0, 30);
dgm.setGridColor(new Color(255, 255, 255, 60));
GridPlugin gridLayout = new GridPlugin(dgm);
view.registerPlugin(gridLayout);

BarSymbolLayer barLayer = new BarSymbolLayer();
view.addLayer(barLayer);

barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
barLayer.addSymbol(b1);
barLayer.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 30));
barLayer.addSymbol(b2);
barLayer.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 30));
barLayer.addSymbol(b3);
barLayer.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 30));
barLayer.addSymbol(b4);
barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
```

4.4.2.6.Bar Symbol Label

Bar default label is the most basic label which draws a simple label at the top of the bar symbol, you can set offset on x and y dimension.The relative label is more complex but easy to use. It place label regarding to some relatives symbol constraints like vertical or horizontal alignment, label background can be stroked and filled with shader. Another label type is the axis label which draws the label in window outer parts like south or west part components.

Default Bar Label

The default label draws the label at the top of the bar. Label can be shifted by using offset x and y properties. Here an example with 3 default label with shifting.

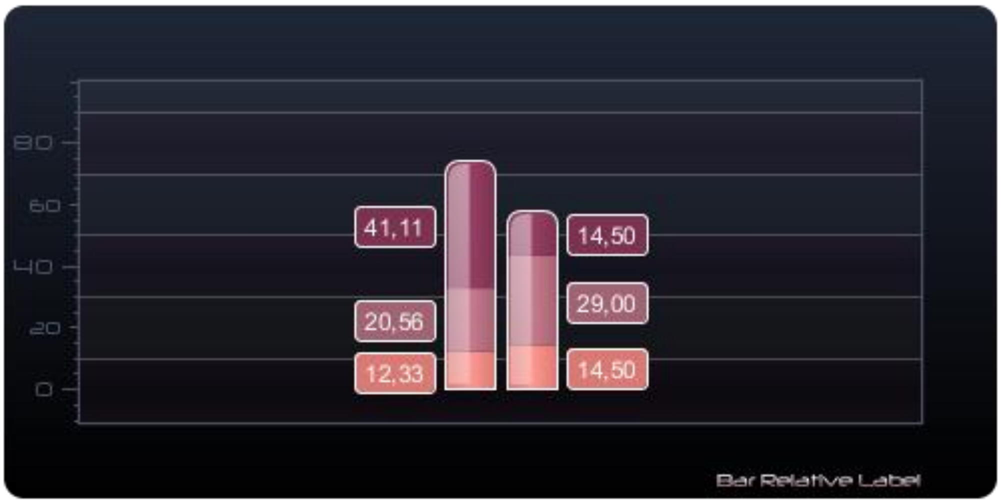


```
/**
* Snippet code: create simple bar in a group, add default label
* and lay out group.
*/
SymbolView view = SymbolToolkit.createView(SymbolNature.Vertical, -35, 80);

BarSymbol b1g1 = new BarSymbol("bar 1");
b1g1.setAscentValue(62);
b1g1.setThemeColor(TangoPalette.CHAMELEON2);
b1g1.setBarLabel(new BarSymbolDefaultLabel(-20, -40));

BarSymbol b2g1 = new BarSymbol("bar 2");
b2g1.setAscentValue(83);
b2g1.setThemeColor(TangoPalette.BUTTER2);
b2g1.setBarLabel(new BarSymbolDefaultLabel(10, 0));

BarSymbol b3g1 = new BarSymbol("bar 3");
b3g1.setAscentValue(47);
b3g1.setThemeColor(TangoPalette.ORANGE2);
b3g1.setBarLabel(new BarSymbolDefaultLabel(0, +40));

BarSymbolGroup group1 = new BarSymbolGroup("Group 1");
group1.setBase(-30);
group1.setThickness(42);
group1.setRound(12);
group1.setMorpheStyle(MorpheStyle.Round);
group1.setBarDraw(new BarDefaultDraw(Color.WHITE));
group1.setBarFill(new BarFill1());
group1.setBarEffect(new BarEffect2());

group1.addSymbol(b1g1);
group1.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 10));
group1.addSymbol(b2g1);
group1.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 10));
group1.addSymbol(b3g1);

BarSymbolLayer barLayer = new BarSymbolLayer();
view.addLayer(barLayer);

barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
barLayer.addSymbol(group1);
barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));

//+ stripe, grid, legend etc.
```

Relative bar label

The relative label draws label according to vertical and horizontal alignment relative to the symbol. You can set alignments with following enumerations. Moreover you can set offset x and y that take in account for solve label location. The rendering properties like fill, stroke, and theme colors can be changed.

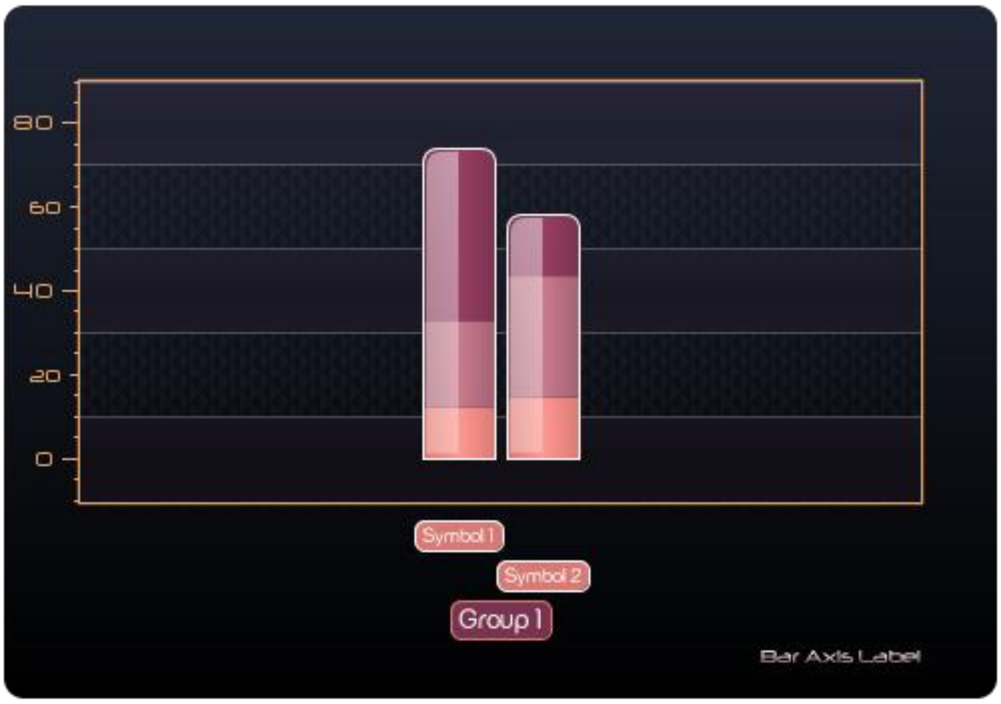
- *VerticalAlignment*
- *HorizontalAlignment*


80
60
40
20
0
41,11
20,56
12,33
14,50
29,00
14,50
Bar Relative Label

```
/**
* Snippet code: create simple bar relative label
*/
View2D view = new View2D();
Window2D w2d = new Window2D(0, 0, -10, 100);
view.registerWindow2D(w2d);
BarSymbolGroup group1 = new BarSymbolGroup("G1");
group1.setBase(0);
group1.setThickness(25);
group1.setRound(8);
group1.setMorpheStyle(MorpheStyle.Round);
group1.setBarDraw(new BarDefaultDraw(Color.WHITE));
group1.setBarFill(new BarFill2());
group1.setBarEffect(new BarEffect1());

StackedBarSymbol b1g1 = new StackedBarSymbol("b1g1");
b1g1.setAscentValue(74);
b1g1.addStack("s1", FilPalette.COPPER1, 12, new BarSymbolRelativeLabel(
             VerticalAlignment.Middle, HorizontalAlignment.WestLeft,
             Color.WHITE, FilPalette.COPPER1, Color.WHITE));
b1g1.addStack("s2", FilPalette.COPPER2, 20, new BarSymbolRelativeLabel(
             VerticalAlignment.Middle, HorizontalAlignment.WestLeft,
             Color.WHITE, FilPalette.COPPER2, Color.WHITE));
b1g1.addStack("s3", FilPalette.COPPER3, 40, new BarSymbolRelativeLabel(
             VerticalAlignment.Middle, HorizontalAlignment.WestLeft,
             Color.WHITE, FilPalette.COPPER3, Color.WHITE));

StackedBarSymbol b2g1 = new StackedBarSymbol("b2g1");
b2g1.setAscentValue(58);
b2g1.addStack("s1", FilPalette.COPPER1, 20, new BarSymbolRelativeLabel(
             VerticalAlignment.Middle, HorizontalAlignment.EastRight,
             Color.WHITE, FilPalette.COPPER1, Color.WHITE));
b2g1.addStack("s2", FilPalette.COPPER2, 40, new BarSymbolRelativeLabel(
             VerticalAlignment.Middle, HorizontalAlignment.EastRight,
             Color.WHITE, FilPalette.COPPER2, Color.WHITE));
b2g1.addStack("s3", FilPalette.COPPER3, 20, new BarSymbolRelativeLabel(
             VerticalAlignment.Middle, HorizontalAlignment.EastRight,
             Color.WHITE, FilPalette.COPPER3, Color.WHITE));

group1.addSymbol(b1g1);
group1.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 6));
group1.addSymbol(b2g1);

SymbolPlugin barPlugin = new SymbolPlugin();
BarSymbolLayer barLayer = new BarSymbolLayer();
barPlugin.addLayer(barLayer);
barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
barLayer.addSymbol(group1);
barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));

barPlugin.setNature(SymbolNature.Vertical);
w2d.registerPlugin(barPlugin);
```

Axis bar Label

If you desire label annotation in axis region, you should use bar axis label.



```
/**
* Snippet code: create simple bar axis label
*/

// axis label for symbol 1
BarDefaultAxisLabel axisLabelBar1Group1 = new BarDefaultAxisLabel(
             "b1 G1", Color.BLACK);
axisLabelBar1Group1.setFont(InputFonts.getYorkville(10));
axisLabelBar1Group1.setFillColor(FilPalette.COPPER1);
axisLabelBar1Group1.setDrawColor(Color.WHITE);
axisLabelBar1Group1.setTextColor(Color.WHITE);
axisLabelBar1Group1.setText("Symbol 1");
axisLabelBar1Group1.setTextPaddingY(1);
axisLabelBar1Group1.setOffsetY(10);
b1g1.setAxisLabel(axisLabelBar1Group1);

// axis label for symbol 2
BarDefaultAxisLabel axisLabelBar2Group1 = new BarDefaultAxisLabel(
             "b2 G1", Color.BLACK);
axisLabelBar2Group1.setFont(InputFonts.getYorkville(10));
axisLabelBar2Group1.setFillColor(FilPalette.COPPER1);
axisLabelBar2Group1.setDrawColor(Color.WHITE);
axisLabelBar2Group1.setTextColor(Color.WHITE);
axisLabelBar2Group1.setText("Symbol 2");
axisLabelBar2Group1.setTextPaddingY(1);
axisLabelBar2Group1.setOffsetY(30);
b2g1.setAxisLabel(axisLabelBar2Group1);

// axis label for group
BarDefaultAxisLabel axisLabelGroup1 = new BarDefaultAxisLabel("Group1",
             Color.BLACK);
axisLabelGroup1.setDrawColor(Color.BLACK);
axisLabelGroup1.setFillColor(FilPalette.COPPER3);
axisLabelGroup1.setDrawColor(FilPalette.COPPER1);
axisLabelGroup1.setTextColor(Color.WHITE);
axisLabelGroup1.setText("Group 1");
axisLabelGroup1.setFont(InputFonts.getYorkville(14));
axisLabelGroup1.setTextPaddingY(1);
axisLabelGroup1.setOffsetY(50);
group1.setAxisLabel(axisLabelGroup1);
```

4.4.2.7.Bar Listener

Bar listener provides an interface that handles bar symbols events like enter, exit, pressed, released and click. On this event, you can custom your application to make animator or interaction with other components. For example, on a bar enter you can make appear a label or change alpha for all symbol to show in first scene the rollover symbol.

```
/**
* Snippet code: add bar listener
*/

barLayer.addBarListener(new BarListener() {
	@Override
	public void barSymbolReleased(BarEvent e) {
		System.out.println("bar released : "+ e.getBarSymbol().getName());
		// put action code
	}

	@Override
	public void barSymbolPressed(BarEvent e) {
		System.out.println("bar pressed : "+ e.getBarSymbol().getName();
		// put action code
	}

	@Override
	public void barSymbolExited(BarEvent e) {
		System.out.println("bar exited : " + e.getBarSymbol().getName())
		// put action code
	}

	@Override
	public void barSymbolEntered(BarEvent e) {
		System.out.println("bar entered : "+e.getBarSymbol().getName());
		// put action code
	}

	@Override
	public void barSymbolClicked(BarEvent e) {
		System.out.println("bar clicked : "+e.getBarSymbol().getClass());
		// put action code
	}

});
```

4.4.3. Point & Curve Symbol

Point symbol are only point reference which one of the dimension is symbolic and the other is scalar metrics like bar symbol. The bar symbol inflate from base to a value (ascent or in descend mode), the point has only a value reference in the scalar metrics dimension and it is lay out as the symbol (in pixel only regarding constraints), like bar symbol, through the symbolic dimension.

A symbol polyline is the curve that links a symbol point collection. Polyline does not contribute itself to layer constraints solving for location, it is only solved by its point's elements coordinates which have already solved in the same layer. Assume that all registered point in polyline are registered in point layer, you can add the polyline in the point layer.

4.4.3.1.Point Layer

The point layer is a layer that hosts point symbol.

```
/**
* Snippet code: point symbol layer
*/

SymbolPlugin sunshineSymbolPlugin = new SymbolPlugin();
sunshineSymbolPlugin.setNature(SymbolNature.Vertical);
sunshineSymbolPlugin.setPriority(100);
sunshineWindow.registerPlugin(sunshineSymbolPlugin);

PointSymbolLayer temperatureLayer = new PointSymbolLayer();
sunshineSymbolPlugin.addLayer(temperatureLayer);
```

4.4.3.2.Point Symbol

The point is defined by it value on the scalar metrics. The location one the symbolic dimension is managed by the point layer.

```
/**
* Snippet code: point symbol
*/
PointSymbol ps1 = new PointSymbol(7);

/**
* Snippet code: point symbol with toolkit
*/
PointSymbol ps2 = SymbolToolkit.createPointSymbol(6.9)
```

4.4.3.3.Point Painter

Point painter draws the point symbol. For example, Image painter draw image at the point symbol coordinate with a specified offset x dimension and offset on y dimension that allow to draw the image at the desired location.

```
/**
* Snippet code: point image painter
*/
pointSymbol.addPointSymbolPainter(new PointSymbolImage(ic.getImage()));
```

To debug the point you are attempting you can use the following debug painter. Ensure that the point is in the location you expect in the layer.

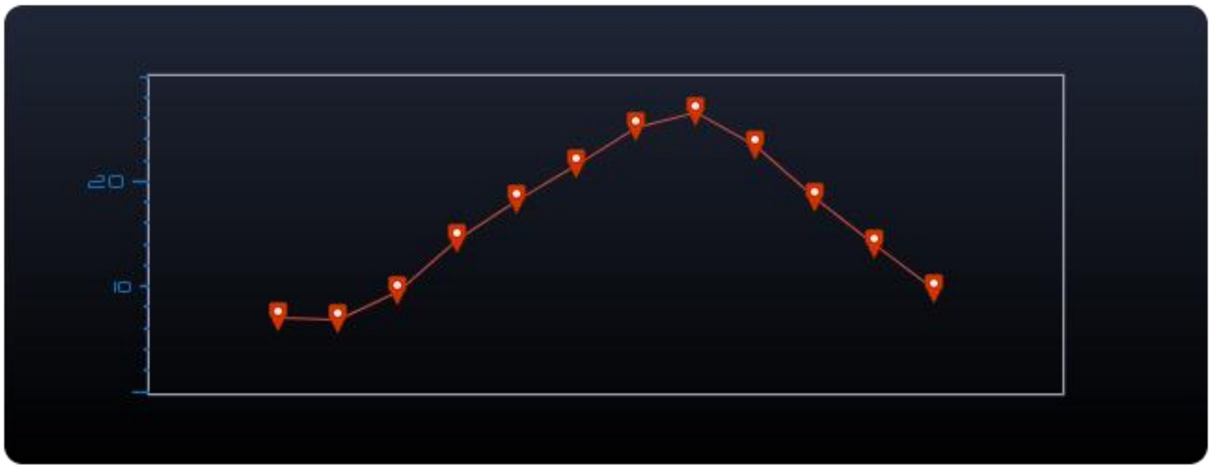
```
/**
* Snippet code: point debug painter which draws the center of point and a
* line that represent the symbol thickness (default at 0 but sometimes you
* wish assign a thickness to the symbol)
*/
pointSymbol.addPointSymbolPainter(new PointSymbolDebugGeometryPainter());
```

4.4.3.4.Polyline Symbol

A polyline symbol defines a point symbol set. It can be used to draw a symbolic curve for a set of points. A polyline requires that the point set symbol have been registered into the point layer and does not contribute to the lay out solving process (in term of component constraints) because polyline is derivatived of the points geometry.

```
/**
* Snippet code: polyline point symbol
*/

PointSymbolLayer pointLayer = new PointSymbolLayer();
symbolPlugin.addLayer(pointLayer);

PointSymbol p1 = new PointSymbol(7);
PointSymbol p2 = new PointSymbol(9);

pointLayer.addSymbol(SymbolComponent.createGlue(PointSymbol.class));
pointLayer.addSymbol(p1);
pointLayer.addSymbol(SymbolComponent.createStrut(PointSymbol.class, 80));
pointLayer.addSymbol(p2);
pointLayer.addSymbol(SymbolComponent.createGlue(PointSymbol.class));


PolylinePointSymbol polylineSymbol = new PolylinePointSymbol("a polyline");
polylineSymbol.addSymbol(p1);
polylineSymbol.addSymbol(p2);

//don’t participate to the layout, add a polyline at any time, assume only
//registerd point are lay out into the point layer.
pointLayer.addSymbol(polylineSymbol);
```

Here a sample of polyline symbols. The symbolic dimension is along X axis and the scalar metrics dimension is along Y axis. The symbol is vertical. Don’t forget that the nature indicates the scalar metrics dimension and not the symbolic dimension. The point symbol value refers to the y window dimension. The symbol is lay out x as a symbol component regarding layer constraints like other symbols and fillers as glues or struts which have been added in layer.



```
/**
* Snippet code: polyline sample
*/
View2D polylineView = new View2D();
polylineView.setPlaceHolderAxisEast(80);
polylineView.setPlaceHolderAxisWest(80);

Window2D polylineWindow = new Window2D(0, 0, 0, 30);
polylineView.registerWindow2D(polylineWindow);


polylineWindow.registerPlugin(new OutlinePlugin());
AxisMilliMetrics metrics = new AxisMilliMetrics(0,
             Axis.AxisWest);
metrics.setMajor(10);
metrics.setMedian(5);
metrics.setMinor(1);
metrics.setMetricsLabelColor(Spectral.SPECTRAL_BLUE2);
metrics.setMetricsMarkerColor(Spectral.SPECTRAL_BLUE2);
polylineWindow.registerPlugin(metrics);

SymbolPlugin symbolPlugin = new SymbolPlugin();
symbolPlugin.setNature(SymbolNature.Vertical);
symbolPlugin.setPriority(100);
polylineWindow.registerPlugin(symbolPlugin);

PointSymbolLayer pointLayer = new PointSymbolLayer();
symbolPlugin.addLayer(pointLayer);
```

```
/**
* Snippet code: polyline sample
*/
PointSymbol ps1 = new PointSymbol(7);
PointSymbol ps2 = new PointSymbol(6.9);
PointSymbol ps3 = new PointSymbol(9.5);
PointSymbol ps4 = new PointSymbol(14.5);
PointSymbol ps5 = new PointSymbol(18.2);
PointSymbol ps6 = new PointSymbol(21.5);
PointSymbol ps7 = new PointSymbol(25.2);
PointSymbol ps8 = new PointSymbol(26.5);
PointSymbol ps9 = new PointSymbol(23.3);
PointSymbol ps10 = new PointSymbol(18.3);
PointSymbol ps11 = new PointSymbol(13.9);
PointSymbol ps12 = new PointSymbol(9.6);

PointSymbol[] pointSymbols = { ps1, ps2, ps3, ps4, ps5, ps6, ps7, ps8,
             ps9, ps10, ps11, ps12 };

/**
 * polyline does not contribute itself to layer constraints solving for
 * location,  it only solve by its points elements coordinates which
 * have to solved in the same layer. you can add without contrainst
 * at any time. Assume that all registered point in polyline are
 * registered in point layer.
 */

PolylinePointSymbol polylineSymbol = new PolylinePointSymbol("polyline");


for (int i = 0; i < pointSymbols.length; i++) {
      polylineSymbol.addSymbol(pointSymbols[i]);
}
pointLayer.addSymbol(polylineSymbol);
polylineSymbol.setPolylinePainter(new PolylinePointSymbolDefaultPainter());

pointLayer.addSymbol(SymbolComponent
             .createGlue(PointSymbol.class));
for (int i = 0; i < pointSymbols.length; i++) {
      ImageIcon ic = SharedIcon.getMarker(Marker.MARKER_SQUARED_RED);
      pointSymbols[i].addPointSymbolPainter(new PointSymbolImage(ic
                   .getImage()));
      pointLayer.addSymbol(pointSymbols[i]);
       //add strut filler except at the last iteration
      if (i < 11)
             pointLayer.addSymbol(SymbolComponent.createStrut(
                          PointSymbol.class, 30));
}
pointLayer.addSymbol(SymbolComponent.createGlue(PointSymbol.class));
```

4.4.3.5.Polyline Painter

A polyline painter draws the curve which is defined by the point symbol coordinate. The curve properties like color and stroke can be set and allow to custom rendering of the line symbol.

```
/**
* Snippet code: polyline painter
*/

PolylinePointSymbol polylineSymbol = new PolylinePointSymbol("my polyline");
polylineSymbol.setPolylinePainter(new PolylinePointSymbolDefaultPainter());
```

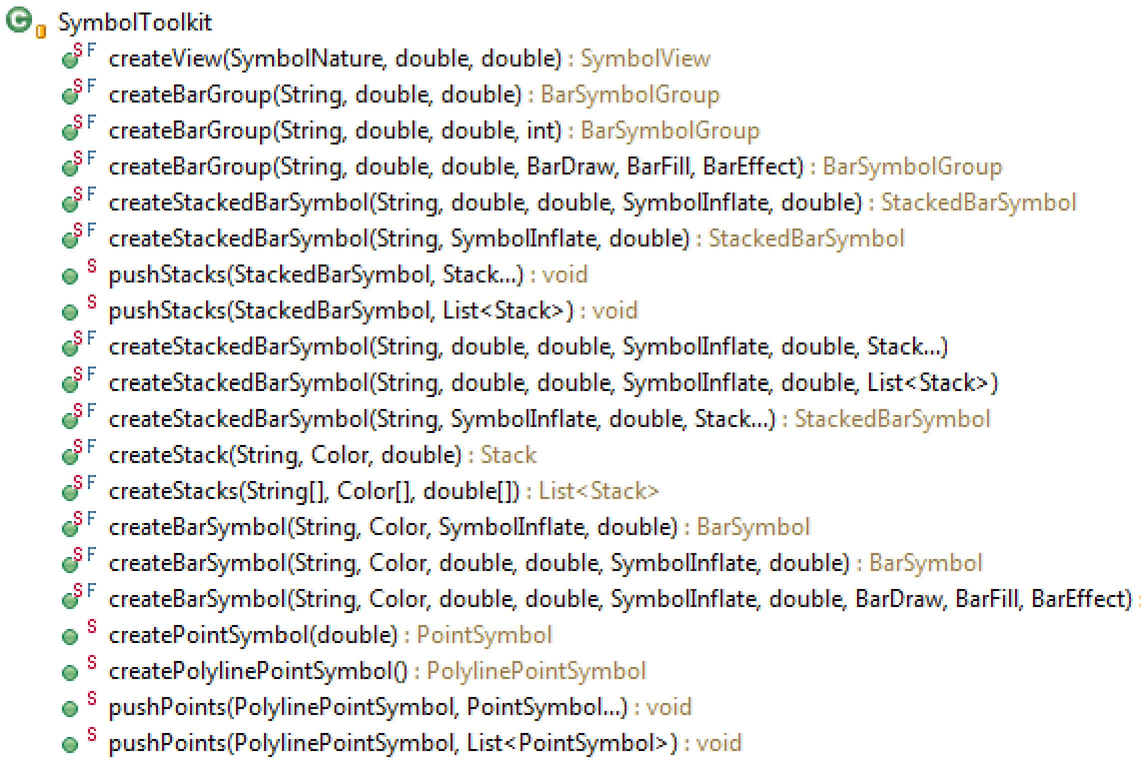
4.4.3.6.Point Listener

Point Layer provides a point listener interface to handle point events which can be occurred on point. The point event brings point symbol that you can retrieve properties. For example you can add a label on point on a point rollover, make interaction with others symbol components or make interaction with your desktop application.

```
/**
* Snippet code: add point listener
*/

pointLayer.addPointListener(new PointListener() {

	@Override
	public void pointSymbolReleased(PointEvent e) {
		System.out.println("point pointSymbolReleased");
		// put action code
	}

	@Override
	public void pointSymbolPressed(PointEvent e) {
		System.out.println("point pointSymbolPressed");
		// put action code
	}

	@Override
	public void pointSymbolExited(PointEvent e) {
		System.out.println("point pointSymbolExited");
		// put action code
	}

	@Override
	public void pointSymbolEntered(PointEvent e) {
		System.out.println("point pointSymbolEntered");
		// put action code
	}

	@Override
	public void pointSymbolClicked(PointEvent e) {
		System.out.println("point pointSymbolClicked");
		// put action code
	}
});
```

4.4.4. Symbol toolkit

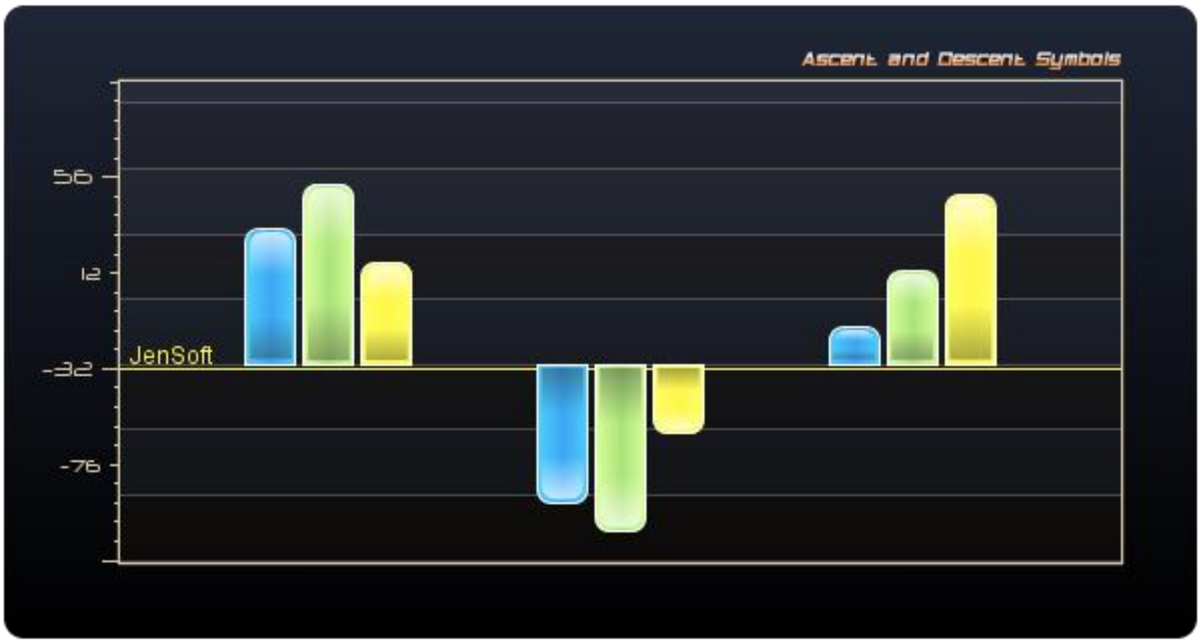
Here is the symbol toolkit methods outline.



```
SymbolToolkit
    createView(SymbolNature, double, double) : SymbolView
    createBarGroup(String, double, double) : BarSymbolGroup
    createBarGroup(String, double, double, int) : BarSymbolGroup
    createBarGroup(String, double, double, BarDraw, BarFill, BarEffect) : BarSymbolGroup
    createStackedBarSymbol(String, double, double, SymbolInflate, double) : StackedBarSymbol
    createStackedBarSymbol(String, SymbolInflate, double) : StackedBarSymbol
    pushStacks(StackedBarSymbol, Stack...) : void
    pushStacks(StackedBarSymbol, List<Stack>) : void
    createStackedBarSymbol(String, double, double, SymbolInflate, double, Stack...)
    createStackedBarSymbol(String, double, double, SymbolInflate, double, List<Stack>)
    createStackedBarSymbol(String, SymbolInflate, double, Stack...) : StackedBarSymbol
    createStack(String, Color, double) : Stack
    createStacks(String[], Color[], double[]) : List<Stack>
    createBarSymbol(String, Color, SymbolInflate, double) : BarSymbol
    createBarSymbol(String, Color, double, double, SymbolInflate, double) : BarSymbol
    createBarSymbol(String, Color, double, double, SymbolInflate, double, BarDraw, BarFill, BarEffect) :
    createPointSymbol(double) : PointSymbol
    createPolylinePointSymbol() : PolylinePointSymbol
    pushPoints(PolylinePointSymbol, PointSymbol...) : void
    pushPoints(PolylinePointSymbol, List<PointSymbol>) : void
```

4.4.5. Sample Symbols

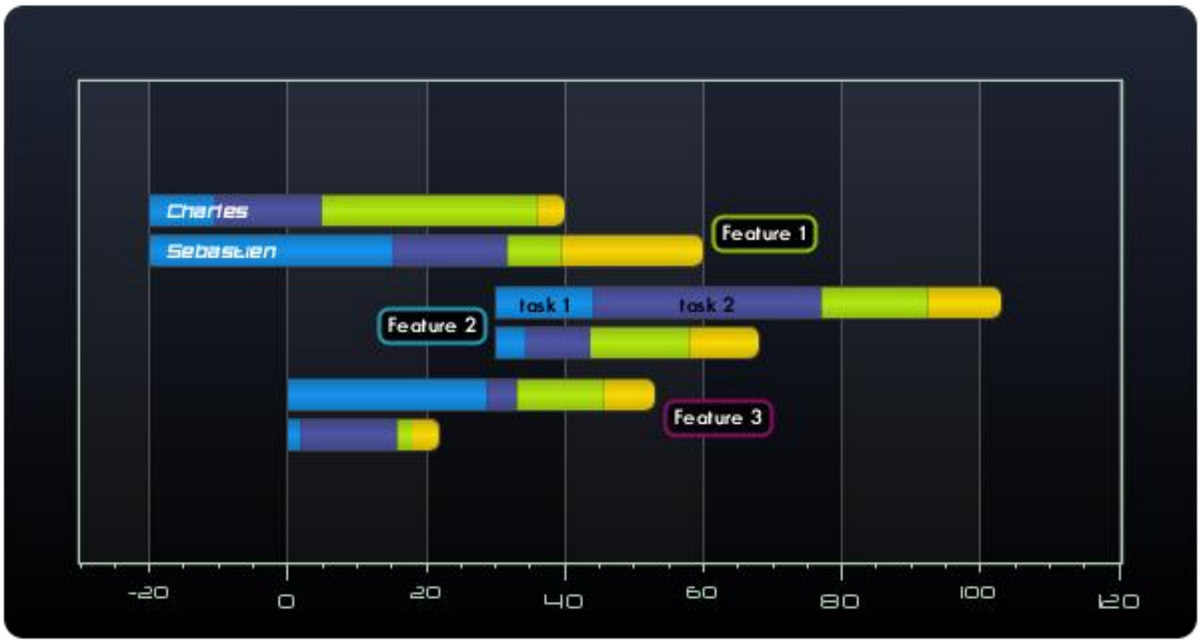
4.4.5.1. Vertical Ascent and Descent Bar Symbol



```
/**
* Snippet code: Ascent and Descent mode
*/



// Descent Symbols for Group 2; the symbol inflates to negative value in the
//scalar metrics dimension in the descent mode
BarSymbol b1g2 = SymbolToolkit.createBarSymbol("b1g1", blue,
             SymbolInflate.Descent, 64);
BarSymbol b2g2 = SymbolToolkit.createBarSymbol("b2g1", green,
             SymbolInflate.Descent, 77);
BarSymbol b3g2 = SymbolToolkit.createBarSymbol("b3g1", orange,
             SymbolInflate.Descent, 32);

// or use factory
// BarFactory.createGroup(nameSymbol, base, thickness);
// BarFactory.createGroup(nameSymbol, base, thickness, round)
// BarFactory.createGroup(nameSymbol, base, thickness, barDraw, barFill,
// barEffect);
BarSymbolGroup group2 = new BarSymbolGroup("G2");
group2.setBase(-30);
group2.setThickness(25);
group2.setRound(8);
group2.setMorpheStyle(MorpheStyle.Round);
group2.setBarDraw(new BarDefaultDraw(Color.WHITE));
group2.setBarFill(new BarFill1());
group2.setBarEffect(new BarEffect2());

group2.addSymbol(b1g2);
group2.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 4));
group2.addSymbol(b2g2);
group2.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 4));
group2.addSymbol(b3g2);
```

4.4.5.2.Horizontal Gant Type

The bar symbol which is used in common charts. Symbol chart is great to design Gantt type chart, the symbol bar can be used as shown in the figure chart below.

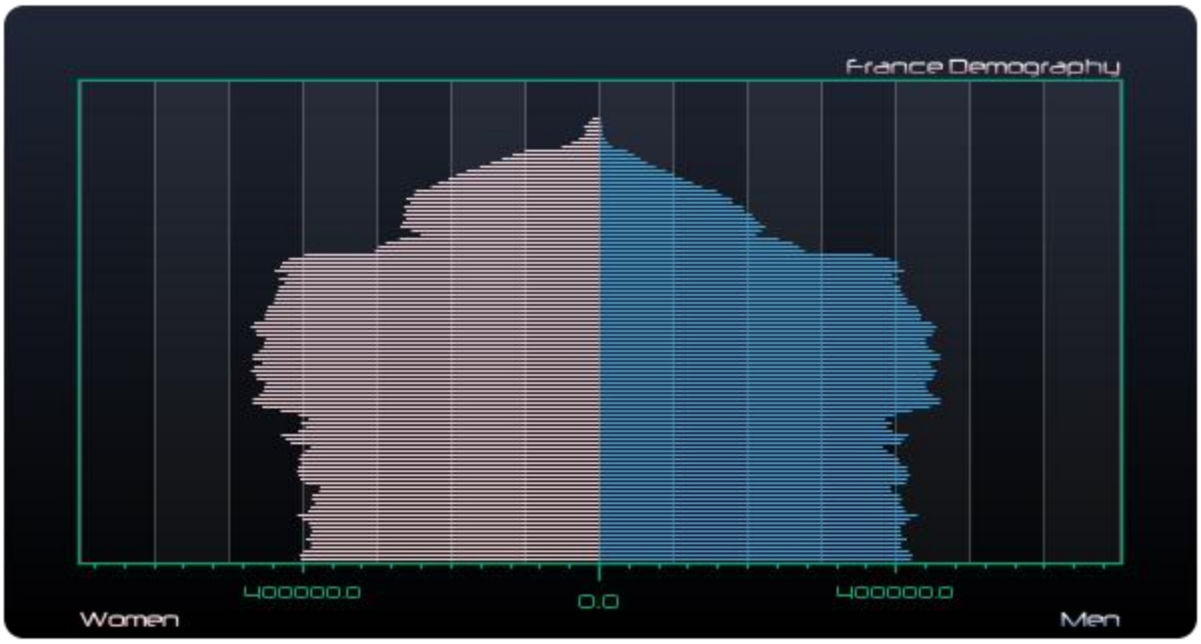


Here is naïve implementation of a Gantt chart which is shown. Symbol represent “feature” for different people. Each feature is divided into task. The axis should be symbolized by time or points.

With static label or dynamic label (show on a rollover or a click on the symbol or sub symbol) you can achieve a very pretty Gantt chart type.

4.4.5.3.Country Age pyramid

Here is a simple example that shows age structure for a population. It is another example of a trendy symbol chart.



On left side is showed the age distribution for women and on the right the age distribution for men.

```
/**
* Age structure part 1: create view, window, plug ins and
* layers
*/
View2D view = new View2D();

Window2D w2d = new Window2D(-700000, 700000, 0, 0);
view.registerWindow2D(w2d);

w2d.registerPlugin(new OutlinePlugin());

SymbolPlugin barPlugin = new SymbolPlugin();
barPlugin.setNature(SymbolNature.Horizontal);
w2d.registerPlugin(barPlugin);

BarSymbolLayer menLayer = new BarSymbolLayer();
BarSymbolLayer womenLayer = new BarSymbolLayer();
```

```
/**
* Age structure part 2 : lay out symbols in layers
*/
barPlugin.addLayer(menLayer);
barPlugin.addLayer(womenLayer);

menLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
womenLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));

Collections.reverse(years);

for(YearDemography yd : years){

        System.out.println(yd);
        BarSymbol men = new BarSymbol("men "+yd.year);
        BarSymbol women = new BarSymbol("women "+yd.year);

        men.setUserObject(yd);
        women.setUserObject(yd);

        men.setBase(0);
        women.setBase(0);

        men.setThickness(2);
        women.setThickness(2);

        men.setAscentValue(yd.men);
        women.setDescentValue(yd.women);

        men.setThemeColor(FilPalette.BLUE12);
        women.setThemeColor(FilPalette.PINK1);

        men.setBarFill(new BarFill1());
        women.setBarFill(new BarFill1());

        menLayer.addSymbol(men);
        //menLayer.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 1));

        womenLayer.addSymbol(women);
        //womenLayer.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 1));

}
```

```
/**
* Age structure part 3: add dynamic label
*/
menLayer.addBarListener(new BarListener() {

	@Override
	public void barSymbolReleased(BarEvent e) {
		// TODO Auto-generated method stub

	}

	@Override
	public void barSymbolPressed(BarEvent e) {
		// TODO Auto-generated method stub

	}

	@Override
	public void barSymbolExited(BarEvent e) {
		// TODO Auto-generated method stub
		e.getBarSymbol().setBarLabel(null);
		e.getBarSymbol().getHost().getWindow2D().getView2D().repaintDevice();
	}

	@Override
	public void barSymbolEntered(BarEvent e) {
	float[] fractions = { 0f, 0.5f, 1f };
	Color[] c = { new Color(0, 0, 0, 200), new Color(0, 0, 0, 255),
			new Color(0, 0, 0, 200) };
	Shader labelShader = new Shader(fractions, c);
	Font fontGroup = InputFonts.getNoMove(10);
	BarSymbolRelativeLabel groupRelativeLabel = new BarSymbolRelativeLabel(
			VerticalAlignment.Middle,
			HorizontalAlignment.EastRight);

	groupRelativeLabel.setDrawColor(FilPalette.BLUE12);
	groupRelativeLabel.setTextColor(Color.WHITE);
	groupRelativeLabel.setOutlineRound(10);
	groupRelativeLabel.setOutlineStroke(new BasicStroke(2f));
	groupRelativeLabel.setShader(labelShader);
	groupRelativeLabel.setFont(fontGroup);

	BarSymbol bs = e.getBarSymbol();
	YearDemography yd = (YearDemography)bs.getUserObject();
	groupRelativeLabel.setText(yd.year+" : "+yd.men +" men");
	bs.setBarLabel(groupRelativeLabel);
	e.getBarSymbol().getHost().getWindow2D().getView2D().repaintDevice();
	}

	@Override
	public void barSymbolClicked(BarEvent e) {


	}
});
```

```
/**
* Age structure part 4: stripes, metrics, grids and legends decorators
*/
DynamicStripeManager dbm = new DynamicStripeManager(
            StripeOrientation.Vertical, 0, 100000);
StripePlugin bandLayout = new StripePlugin(dbm);
StripePalette bp = new StripePalette();
bp.addPaint(new Color(255, 255, 255, 40));
bp.addPaint(new Color(40, 40, 40, 40));
dbm.setBandPalette(bp);
bandLayout.setAlpha(0.3f);
w2d.registerPlugin(bandLayout);

DynamicGridManager dgm = new DynamicGridManager(
      GridOrientation.Vertical, 0, 100000);
dgm.setGridColor(new Color(255, 255, 255, 60));
GridPlugin gridLayout = new GridPlugin(dgm);
w2d.registerPlugin(gridLayout);

 AxisMilliMetrics miliWest = new AxisMilliMetrics(0,Axis.AxisSouth);
 miliWest.setMajor(200000);
 miliWest.setMedian(100000);
 miliWest.setMinor(10000);
 w2d.registerPlugin(miliWest);
 miliWest.getMetricsLayoutManager().setMetricsFormat(new IMetricsFormat() {
      @Override
      public String format(double d) {
            return ""+Math.abs(d);
      }
});

LegendPlugin legendPlugin = new LegendPlugin();

Legend legend = new Legend("France Demography");
legend.setLegendFill(new LegendFill1(Color.WHITE, FilPalette.COPPER2));
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.North, 0.1f,
            LegendAlignment.Rigth));

Legend menLegend = new Legend("Men");
menLegend.setLegendFill(new LegendFill1(Color.WHITE, FilPalette.BLUE2));
menLegend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
menLegend.setConstraints(new LegendConstraints(LegendPosition.South, 0.8f,
            LegendAlignment.Rigth));

Legend womenLegend = new Legend("Women");
womenLegend.setLegendFill(new LegendFill1(Color.WHITE, FilPalette.PINK1));
womenLegend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
womenLegend.setConstraints(new LegendConstraints(LegendPosition.South, 0.8f,
            LegendAlignment.Left));

legendPlugin.addLegend(legend);
legendPlugin.addLegend(menLegend);
legendPlugin.addLegend(womenLegend);
w2d.registerPlugin(legendPlugin);
```

4.4.5.4.Bar Animator

Animate bar is pretty simple with the method inflate in the *BarSymbol* class. This method inflates symbol for the current value to the specified delta value as argument method. You have to choose the delay for the total transition and the step count to attempt the new specified value during the specified delay. A good step count to make the transition is to 10 to 50 step count for duration around 300 milliseconds.

As show in the animated bar demo, assume you have some symbols:

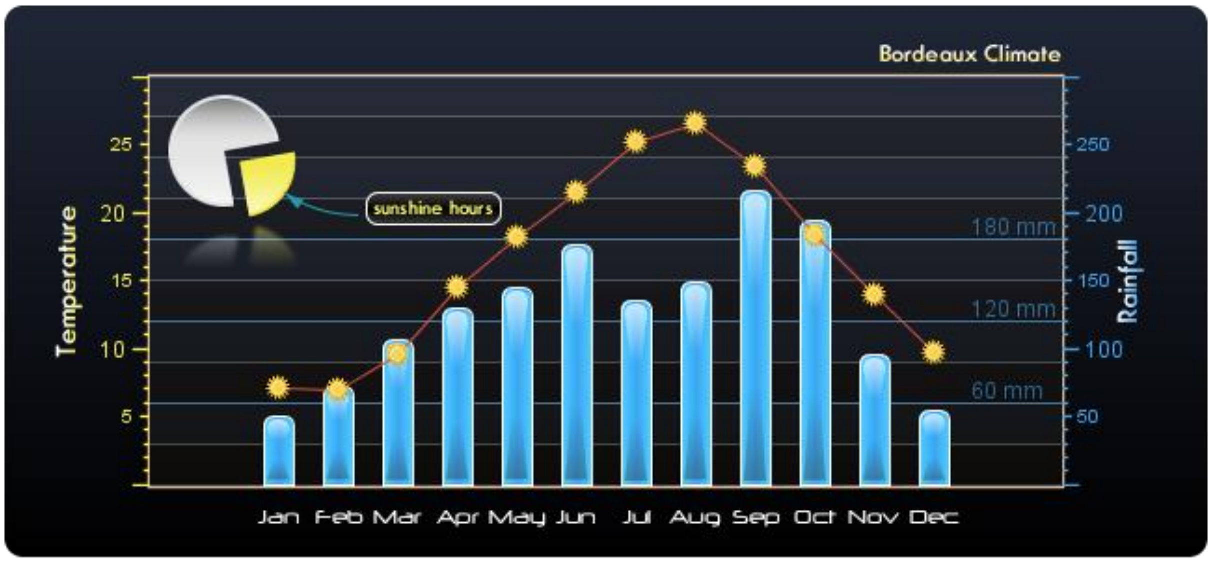
```java
/**
* Bar symbol animator : 9 symbols, 3 groups and three symbol by group.
*/
BarSymbol b1g1;
BarSymbol b2g1;
BarSymbol b3g1;

BarSymbol b1g2;
BarSymbol b2g2;
BarSymbol b3g2;

BarSymbol b1g3;
BarSymbol b2g3;
BarSymbol b3g3;

// create compatible bar view
SymbolView view = SymbolToolkit.createView(SymbolNature.Vertical, -42,
             260);

Color blue = PetalPalette.PETAL1_HC;
Color green = PetalPalette.PETAL2_HC;
Color orange = PetalPalette.PETAL3_HC;

// Group1
b1g1 = new BarSymbol();
b1g1.setAscentValue(62);
b1g1.setThemeColor(blue);
b2g1 = new BarSymbol();
b2g1.setAscentValue(83);
b2g1.setThemeColor(green);
b3g1 = new BarSymbol();
b3g1.setAscentValue(47);
b3g1.setThemeColor(orange);

BarSymbolGroup group1 = new BarSymbolGroup();
group1.addSymbol(b1g1);
group1.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 4));
group1.addSymbol(b2g1);
group1.addSymbol(SymbolComponent.createStrut(BarSymbol.class, 4));
group1.addSymbol(b3g1);

group1.setSymbol("Group 1");
group1.setName("group1");
group1.setBase(-30);
group1.setThickness(25);
group1.setRound(8);
group1.setMorpheStyle(MorpheStyle.Round);
group1.setBarDraw(new BarDefaultDraw(Color.WHITE));
group1.setBarFill(new BarFill1());
group1.setBarEffect(new BarEffect4());

// Group 2 and Group 3…
```

```
/**
* Bar symbol animator part 2: lay out symbols in layer
*/
BarSymbolLayer barLayer = new BarSymbolLayer();
view.addLayer(barLayer);

barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
barLayer.addSymbol(group1);
barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
barLayer.addSymbol(group2);
barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
barLayer.addSymbol(group3);
barLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));

/**
* Bar symbol animator part 3: thread sample
*/
class InflateDefateDemo extends Thread {

      public InflateDefateDemo() {
      }

      Random r1 = new Random();
      Random r2 = new Random();

      int delay = 50;
      int delta = 20;

      private void animate(BarSymbol s) {
            int i = r1.nextInt(2);
            int j = r2.nextInt(delta);

            if (i == 0) {
                  if (s.getValue() < 240)
                        s.inflate(j, 0, delay, 20);
            } else {
                  if (s.getValue() > 20)
                        s.deflate(j, 0, delay, 20);
            }
      }

      @Override
      public void run() {
            //impl
      }
}
```

```
/**
* Bar symbol animator part 3: run method
*/
@Override
public void run() {
	try {

		// inflate all bar to 10
		b1g1.setAscentValue(10);
		//other
		Thread.sleep(1000);
		// inflate all with delay
		b1g1.inflate(160, 0, 300, 20);
		b2g1.inflate(160, 50, 300, 20);
		b3g1.inflate(160, 100, 300, 20);
		b1g2.inflate(160, 150, 300, 20);
		b2g2.inflate(160, 200, 300, 20);
		b3g2.inflate(160, 250, 300, 20);
		b1g3.inflate(160, 300, 300, 20);
		b2g3.inflate(160, 350, 300, 20);
		b3g3.inflate(160, 450, 300, 20);
		Thread.sleep(1000);
		// deflate all together
		b1g1.deflate(100, 0, 300, 20);
		b2g1.deflate(100, 0, 300, 20);
		b3g1.deflate(100, 0, 300, 20);
		b1g2.deflate(100, 0, 300, 20);
		b2g2.deflate(100, 0, 300, 20);
		b3g2.deflate(100, 0, 300, 20);
		b1g3.deflate(100, 0, 300, 20);
		b2g3.deflate(100, 0, 300, 20);
		b3g3.deflate(100, 0, 300, 20);
		Thread.sleep(1000);
		while (true) {
			animate(b1g1);
			animate(b2g1);
			animate(b3g1);
			animate(b1g2);
			animate(b2g2);
			animate(b3g2);
			animate(b1g3);
			animate(b2g3);
			animate(b3g3);
			Thread.sleep(100);
		}

	} catch (InterruptedException e) {
		e.printStackTrace();
	}

}
```

4.4.5.5. Climate Bar Symbol and point curve symbols

A climate chart can be show rainfall, temperature or sunshine hours. Symbols is suitable plug-in to build a climate view.



As shown in the example above, rainfall are symbolized by bar symbol component, temperatures are symbolized by points symbols which are linked by a symbol curve, and sunshine hours by a pie in component system coordinates.

```
/**
* Climate view, first create the view 2D
*/
View2D climateView = new View2D();
climateView.setPlaceHolderAxisEast(80);
climateView.setPlaceHolderAxisWest(80);
```

```
/**
* Climate view, create a window for rainfall with an east axis plug-in
* and the symbol plug-in
*/

Window2D window2DRainfall = new Window2D(0, 0, 0, 300);
window2DRainfall.setName("compatible vertical bar window");
climateView.registerWindow2D(window2DRainfall);

window2DRainfall.setThemeColor(RosePalette.MELON);
window2DRainfall.registerPlugin(new OutlinePlugin());

AxisMilliMetrics axisMiliMetrics = new AxisMilliMetrics(0,
             Axis.AxisEast);
axisMiliMetrics.setMajor(100);
axisMiliMetrics.setMedian(50);
axisMiliMetrics.setMinor(10);
axisMiliMetrics.setMetricsLabelColor(blue);
axisMiliMetrics.setMetricsMarkerColor(blue);

window2DRainfall.registerPlugin(axisMiliMetrics);
window2DRainfall.setThemeColor(RosePalette.MELON);

SymbolPlugin rainFallSymbolPlugin = new SymbolPlugin();
rainFallSymbolPlugin.setNature(SymbolNature.Vertical);
rainFallSymbolPlugin.setPriority(100);
window2DRainfall.registerPlugin(rainFallSymbolPlugin);
```

```java
/**
* Climate view, create rainfall symbols
*/
BarSymbol b1 = SymbolToolkit.createBarSymbol("January", blue,
             SymbolInflate.Ascent, 49.9);
BarSymbol b2 = SymbolToolkit.createBarSymbol("Februry", blue,
             SymbolInflate.Ascent, 71.5);
BarSymbol b3 = SymbolToolkit.createBarSymbol("March", blue,
             SymbolInflate.Ascent, 106.4);
BarSymbol b4 = SymbolToolkit.createBarSymbol("April", blue,
             SymbolInflate.Ascent, 129.2);
BarSymbol b5 = SymbolToolkit.createBarSymbol("May", blue,
             SymbolInflate.Ascent, 144);
BarSymbol b6 = SymbolToolkit.createBarSymbol("June", blue,
             SymbolInflate.Ascent, 176);
BarSymbol b7 = SymbolToolkit.createBarSymbol("July", blue,
             SymbolInflate.Ascent, 135.6);
BarSymbol b8 = SymbolToolkit.createBarSymbol("August", blue,
             SymbolInflate.Ascent, 148.5);
BarSymbol b9 = SymbolToolkit.createBarSymbol("September", blue,
             SymbolInflate.Ascent, 216.4);
BarSymbol b10 = SymbolToolkit.createBarSymbol("October", blue,
             SymbolInflate.Ascent, 194.1);
BarSymbol b11 = SymbolToolkit.createBarSymbol("November", blue,
             SymbolInflate.Ascent, 95.6);
BarSymbol b12 = SymbolToolkit.createBarSymbol("December", blue,
             SymbolInflate.Ascent, 54.4);

BarSymbol[] rainfalls = { b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11,
             b12 };



/**
* Climate view, create axis month label for this rainfalls symbols
*/
for (int i = 0; i < rainfalls.length; i++) {
      BarDefaultAxisLabel rl = new
       BarDefaultAxisLabel("month",Color.WHITE);
      rl.setFont(InputFonts.getNeuropol(12));
      rl.setTextColor(NanoChromatique.WHITE);
      rl.setText(rainfalls[i].getSymbol().substring(0, 3));
      rl.setTextPaddingY(1);
      rl.setOffsetY(10);
      rainfalls[i].setAxisLabel(rl);
}
```

```
/**
* Climate view, lay out this symbol in a group which defines common
properties for all symbols, fixed thickness in pixels, base from zero, etc.
*/
// or use bar factory
// BarFactory.createGroup(nameSymbol, base, thickness);
// BarFactory.createGroup(nameSymbol, base, thickness, round)
// BarFactory.createGroup(nameSymbol, base, thickness, barDraw, barFill,
             // barEffect);

// create a group for properties
BarSymbolGroup group1 = new BarSymbolGroup("G1");
group1.setBase(0);
group1.setThickness(16);
group1.setRound(6);
group1.setMorpheStyle(MorpheStyle.Round);
group1.setBarDraw(new BarDefaultDraw(Color.WHITE));
group1.setBarFill(new BarFill1());
group1.setBarEffect(new BarEffect4());



/**
* Climate view, lay out this group in a bar symbol layer
*/
// lay out symbol in the group with strut filler

for (int i = 0; i < rainfalls.length; i++) {
       group1.addSymbol(rainfalls[i]);
       if (i < 11)
             group1.addSymbol(SymbolComponent.createStrut(BarSymbol.class,
                         16));
}


// lay out this group in a layer with glue filler from west and east side
BarSymbolLayer rainFallLayer = new BarSymbolLayer();
rainFallSymbolPlugin.addLayer(rainFallLayer);

rainFallLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
rainFallLayer.addSymbol(group1);
rainFallLayer.addSymbol(SymbolComponent.createGlue(BarSymbol.class));
```

```java
/**
* Climate view, add decorators
*/
// HORIZONTAL BAND DECORATOR
DynamicStripeManager dbm = new DynamicStripeManager(
             StripeOrientation.Horizontal, 0, 30);
StripePlugin bandLayout = new StripePlugin(dbm);
StripePalette bp = new StripePalette();
bp.addPaint(new Color(255, 255, 255, 40));
bp.addPaint(ColorPalette.alpha(TangoPalette.ORANGE3, 40));
dbm.setBandPalette(bp);
bandLayout.setAlpha(0.3f);
window2DRainfall.registerPlugin(bandLayout);

// HORIZONTAL AND VERTICAL GRID DECORATOR
DynamicGridManager dgm = new DynamicGridManager(
             GridOrientation.Horizontal, 0, 30);
dgm.setGridColor(new Color(255, 255, 255, 60));
GridPlugin gridLayout = new GridPlugin(dgm);
window2DRainfall.registerPlugin(gridLayout);

// a free grid for reference level rainfalls
FreeGridManager fgm = new FreeGridManager();
fgm.setGridOrientation(GridOrientation.Horizontal);
fgm.addGrid(60, "60 mm", new Alpha(blue, 150), 0.9f);
fgm.addGrid(120, "120 mm", new Alpha(blue, 150), 0.9f);
fgm.addGrid(180, "180 mm", new Alpha(blue, 150), 0.9f);
fgm.setGridStroke(new BasicStroke(0.8f));
window2DRainfall.registerPlugin(new GridPlugin(fgm));



/**
* Climate view, create window for temperature (which is another window
* projection for display in the same view chart)
*/
Window2D temperatureWindow = new Window2D(0, 0, 0, 30);
climateView.registerWindow2D(temperatureWindow);

// metrics for sunshine window
AxisMilliMetrics sunshineMetrics = new AxisMilliMetrics(0,
             Axis.AxisWest);
sunshineMetrics.setMajor(10);
sunshineMetrics.setMedian(5);
sunshineMetrics.setMinor(1);
sunshineMetrics.setMetricsLabelColor(orange);
sunshineMetrics.setMetricsMarkerColor(orange);
temperatureWindow.registerPlugin(sunshineMetrics);

SymbolPlugin temperatureSymbolPlugin = new SymbolPlugin();
temperatureSymbolPlugin.setNature(SymbolNature.Vertical);
temperatureSymbolPlugin.setPriority(100);
temperatureWindow.registerPlugin(temperatureSymbolPlugin);
```

```java
/**
* Climate view, create temperature point symbol
*/
PointSymbol ps1 = new PointSymbol(7);
PointSymbol ps2 = new PointSymbol(6.9);
PointSymbol ps3 = new PointSymbol(9.5);
PointSymbol ps4 = new PointSymbol(14.5);
PointSymbol ps5 = new PointSymbol(18.2);
PointSymbol ps6 = new PointSymbol(21.5);
PointSymbol ps7 = new PointSymbol(25.2);
PointSymbol ps8 = new PointSymbol(26.5);
PointSymbol ps9 = new PointSymbol(23.3);
PointSymbol ps10 = new PointSymbol(18.3);
PointSymbol ps11 = new PointSymbol(13.9);
PointSymbol ps12 = new PointSymbol(9.6);

PointSymbol[] sunshines = { ps1, ps2, ps3, ps4, ps5, ps6, ps7, ps8,
            ps9, ps10, ps11, ps12 };



/**
* Climate view, lay out point symbol in a point symbols layer
* with adjust filler to be at the same location along x axis (symbolic
* dimension)
*/

PointSymbolLayer temperatureLayer = new PointSymbolLayer();
temperatureSymbolPlugin.addLayer(temperatureLayer);

temperatureLayer.addSymbol(SymbolComponent
            .createGlue(PointSymbol.class));
for (int i = 0; i < sunshines.length; i++) {

      ImageIcon ic = SharedIcon.getWeather(Weather.WEATHER_SUN_16);
      PointSymbolImage psi = new PointSymbolImage(ic.getImage());
      sunshines[i].addPointSymbolPainter(psi);

      psi.setOffsetX(-ic.getIconWidth()/2);
      psi.setOffsetY(-ic.getIconHeight()/2);

      temperatureLayer.addSymbol(sunshines[i]);
      if (i < 11)
            temperatureLayer.addSymbol(SymbolComponent.createStrut(
                        PointSymbol.class, 32));
}
temperatureLayer.addSymbol(SymbolComponent
            .createGlue(PointSymbol.class));
```

```java
/**
* Climate view, register a curve symbol
*/

/**
* polyline does not contribute itself to layer constraints solving for location,
* it only solve by its points elements coordinates
* which have to solved in the same layer. you can add without constraint at any *
* time. Assume that
* all registered point in polyline are registered in point layer.
*/
PolylinePointSymbol polylineSymbol = new PolylinePointSymbol("sunshine polyline");


for (int i = 0; i < sunshines.length; i++) {
      polylineSymbol.addSymbol(sunshines[i]);
}
temperatureLayer.addSymbol(polylineSymbol);
polylineSymbol.setPolylinePainter(new PolylinePointSymbolDefaultPainter());



/**
* Climate view, register text legends (legend can be registered in any of
* registered windows)
*/
Legend legend = new Legend("Bordeaux Climate");
legend.setLegendFill(new LegendFill1(Color.WHITE, FilPalette.YELLOW2));
legend.setFont(InputFonts.getFont(InputFonts.NO_MOVE, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.North, 0.2f,
             LegendAlignment.Rigth));

Legend rainfalllegend = new Legend("Rainfall");
rainfalllegend.setLegendFill(new LegendFill1(Color.WHITE, blue));
rainfalllegend.setFont(InputFonts.getFont(InputFonts.NO_MOVE, 14));
rainfalllegend.setConstraints(new LegendConstraints(
             LegendPosition.East, 0.5f, LegendAlignment.Middle));

Legend sunshinelegend = new Legend("Temperature");
sunshinelegend.setLegendFill(new LegendFill1(Color.WHITE, orange));
sunshinelegend.setFont(InputFonts.getFont(InputFonts.NO_MOVE, 14));
sunshinelegend.setConstraints(new LegendConstraints(
             LegendPosition.West, 0.5f, LegendAlignment.Middle));

LegendPlugin legendPlugin = new LegendPlugin();
legendPlugin.addLegend(legend);
legendPlugin.addLegend(rainfalllegend);
legendPlugin.addLegend(sunshinelegend);
window2DRainfall.registerPlugin(legendPlugin);
```

```
/**
* Climate view, create pie for sunshine. Pie is project in component system
* coordinate to stay in corner top left of the device even if you zoom the
* windows.
*/
PiePlugin piePlugin = new PiePlugin();

Pie pie = PieToolkit.createPie("sunshine pie", 30);
PieLinearEffect fx1 = new PieLinearEffect();
PieReflectionEffect fx2 = new PieReflectionEffect();
PieCompoundEffect c = new PieCompoundEffect(fx1, fx2);
fx1.setOffsetRadius(2);
pie.setPieEffect(c);
pie.setStartAngleDegree(280);
pie.setPieNature(PieNature.PieDevice);
pie.setCenterX(40);
pie.setCenterY(40);
piePlugin.setPriority(100);// paint last



/**
* Climate view, create pie slices
*/
PieSlice s1 = PieToolkit.createSlice("s1",
             ColorPalette.alpha(orange, 240), 25, 0);
PieSlice s2 = PieToolkit.createSlice("s2",
             new Color(240, 240, 240, 240), 75, 0);
s1.setDivergence(10);

PieBorderLabel label2 = new PieBorderLabel();
label2.setLabelFont(InputFonts.getNoMove(10));
label2.setFillColor(Color.BLACK);
label2.setOutlineColor(Color.WHITE);
label2.setLinkColor(Spectral.SPECTRAL_BLUE1);
label2.setLabelPaddingY(2);
label2.setLabelPaddingX(4);
label2.setOutlineRound(12);
float[] fractions = { 0f, 0.5f, 1f };
Color[] colors = { Color.DARK_GRAY, Color.BLACK, Color.DARK_GRAY };
label2.setShader(fractions, colors);
label2.setLabel("sunshine hours");
s1.addSliceLabel(label2);

PieToolkit.pushSlices(pie, s1, s2);
piePlugin.addPie(pie);



/**
* Climate view, register pie plug-in in any of the registered windows (pie
will be not project, and then any of existing windows can bring the pie.)
*/

window2DRainfall.registerPlugin(piePlugin);
```

4.5. Ray

Ray is not a symbol. Only ray morphe looks like a symbol but unlike symbol, a ray have two metrics dimension and not only one like symbol. For example a vertical ray inflates from base to the ascent or descent value but has an x coordinate and it is not lay out by a layer. The thickness can be in pixel or project in a user space because this dimension is also scalar metrics. For example if you need is to make a “scan” component, ray is the appropriate component to achieve it. Suppose you scan frequencies on a bandwidth. The detected channel is a measure of physical value (electromagnetic field) at the specified frequency. That is a ray. The ray has a fixed value in pixel if you decide to make a pseudo symbol or the thickness can be project if you desire which should be have a particular thickness in the user coordinate system (for example, a scanned channel should be an exact frequency symbolized by this ray with a fixed size in pixel, 1, 2, 3, etc. pixels or the ray has a frequency bandwidth and the thickness value is project in the user coordinate system)

4.5.1. Ray Plug-in

The ray plug-in knows how to design ray graph.

```
/**
* Snippet code: create a ray view with a view, a window and  ray
* plug in.
*/

View2D view = new View2D();

view.setPlaceHolderAxisSouth(80);
view.setPlaceHolderAxisWest(120);
view.setPlaceHolderAxisEast(120);

Window2D w2d = new Window2D(-100, 150, 0, 100);
w2d.setName("ray window");
RayPlugin rayPlugin = new RayPlugin();
w2d.registerPlugin(rayPlugin);

w2d.registerPlugin(new OutlinePlugin());
```

You should be use *RayToolkit* to get a new ray view instance which a compatible view with ray. Here is an example which create a ray compatible view with `[x min, x max, y min, y max]` user bounds projection.

```
/**
* Snippet code: create a ray compatible view
*/

RayView view = RayToolkit.createCompatibleView(-200, 200, -200, 200);
```

4.5.2. Ray

A ray is a kind of rectangle which inflates from a base to a value in ascent or decent mode (positive or negative value). The ray is centered on the ray value in the other dimension. For figure out a ray in a physical sample, here is following example. The scanned channel is a ray which is detected as a peak defines by the measured value for a specified frequency. The peak is the *ascent or descent value* (like bar symbol) and moreover the frequency (in Hertz) which is the *ray value* for the detected peak (maybe in joule or µeV).

```
/**
* Snippet code: create a ray in device coordinate system (in pixel) with a
* fixed thickness of 20 pixel.
*/

Ray ray = new Ray();
ray.setName("ray 1");
ray.setThicknessType(ThicknessType.Device);
ray.setThickness(20);
ray.setRayNature(RayNature.XRay);
ray.setRayBase(0);
ray.setAscentValue(40);
ray.setRay(60);
ray.setThemeColor(blue);
ray.setRayDraw(new RayDefaultDraw(Color.WHITE));
ray.setRayFill(new RayFill1());
ray.setRayEffect(new RayEffect1());

rayPlugin.addRay(ray);
```

4.5.3. Stacked ray

A stacked ray is a ray which contains stacks. A stack is added to the host ray with a relative value. That value is normalized to the host ray ascent or descent value.

```
/**
* Snippet code: create a stacked ray in thickness device system coordinate
* with a fixed size in pixel of 32.
*/
View2D view = new View2D();

view.setPlaceHolderAxisSouth(80);
view.setPlaceHolderAxisWest(120);
view.setPlaceHolderAxisEast(120);

Color red = new Color(254, 206, 12);
Color green = new Color(125, 186, 39);
Color orange = new Color(223, 167, 59);

Window2D w2d = new Window2D(-100, 100, -20, 100);
w2d.setName("bar window");

AxisMilliMetrics miliSouth = new AxisMilliMetrics(0, Axis.AxisSouth);
miliSouth.setMajor(40);
miliSouth.setMedian(20);
miliSouth.setMinor(5);
w2d.registerPlugin(miliSouth);

AxisMilliMetrics miliWest = new AxisMilliMetrics(0, Axis.AxisWest);
miliWest.setMajor(40);
miliWest.setMedian(20);
miliWest.setMinor(5);
w2d.registerPlugin(miliWest);

RayPlugin rayPlugin = new RayPlugin();

StackedRay sray = new StackedRay();
sray.setName("stacked ray 1");
sray.setThicknessType(ThicknessType.Device);
sray.setThickness(36);
sray.setRayNature(RayNature.XRay);
sray.setRayBase(20);
sray.setAscentValue(40);
sray.setRay(90);
sray.setThemeColor(Color.WHITE);
sray.setRayDraw(new RayDefaultDraw(Color.WHITE));
RayStack rs11 = new RayStack("ray 1 stack 1", TangoPalette.BUTTER3, 20);
rs11.setRayFill(new RayFill2());
RayStack rs12 = new RayStack("ray 1 stack 2", TangoPalette.CHAMELEON3,
             40);
rs12.setRayFill(new RayFill1());

sray.addStack(rs11);
sray.addStack(rs12);
rayPlugin.addRay(sray);
```

4.5.4. Ray Group

A ray group allows setting some of common properties for a ray set which shared these properties (base, fill, stroke, etc.)

```
/**
* Snippet code: create a ray group.
*/
RayPlugin rayPlugin = new RayPlugin();

// ray 1
Ray ray = new Ray("ray1");
ray.setAscentValue(40);
ray.setRay(60);
ray.setThemeColor(PetalPalette.PETAL1_HC);

// ray 2
Ray ray2 = new Ray("ray2");
ray2.setAscentValue(40);
ray2.setRay(80);
ray2.setThemeColor(PetalPalette.PETAL2_HC);

// ray 3
Ray ray3 = new Ray("ray3");
ray3.setAscentValue(40);
ray3.setRay(100);
ray3.setThemeColor(PetalPalette.PETAL3_HC);

RayGroup group = new RayGroup();
group.setGroupNature(RayNature.XRay);
group.setGroupThicknessType(ThicknessType.Device);
group.setGroupThickness(20);
group.setRayBase(0);
group.setGroupDraw(new RayDefaultDraw(Color.WHITE));
group.setGroupFill(new RayFill1());
group.setGroupEffect(new RayEffect1());

group.addRay(ray);
group.addRay(ray2);
group.addRay(ray3);

rayPlugin.addRay(group);
```

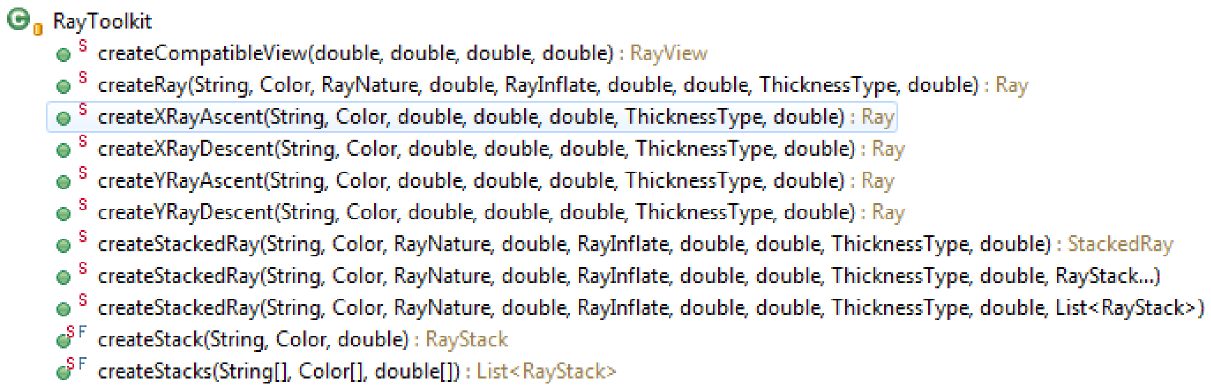
4.5.5. Ray Listener

A ray listener allows putting action code when ray event like enter, exit, pressed, released and click occurs on a ray.

```
/**
* Snippet code: create a ray listener.
*/

rayPlugin.addRayListener(new RayListener() {

	@Override
	public void rayReleased(RayEvent e) {
		System.out.println("ray released : " + e.getRay().getName());
	}

	@Override
	public void rayPressed(RayEvent e) {
		System.out.println("ray pressed : " + e.getRay().getName());
	}

	@Override
	public void rayExited(RayEvent e) {
		System.out.println("ray exited : " + e.getRay().getName());
	}

	@Override
	public void rayEntered(RayEvent e) {
		System.out.println("ray entered : " + e.getRay().getName());
	}

	@Override
	public void rayClicked(RayEvent e) {
		System.out.println("ray clicked : " + e.getRay().getName());
	}
});
```

4.5.6. Ray Toolkit

Ray toolkit helps to create related ray object with static factory methods.



```
RayToolkit
  createCompatibleView(double, double, double, double) : RayView
  createRay(String, Color, RayNature, double, RayInflate, double, double, ThicknessType, double) : Ray
  createXRayAscent(String, Color, double, double, double, ThicknessType, double) : Ray
  createXRayDescent(String, Color, double, double, double, ThicknessType, double) : Ray
  createYRayAscent(String, Color, double, double, double, ThicknessType, double) : Ray
  createYRayDescent(String, Color, double, double, double, ThicknessType, double) : Ray
  createStackedRay(String, Color, RayNature, double, RayInflate, double, double, ThicknessType, double) : StackedRay
  createStackedRay(String, Color, RayNature, double, RayInflate, double, double, ThicknessType, double, RayStack...)
  createStackedRay(String, Color, RayNature, double, RayInflate, double, double, ThicknessType, double, List<RayStack>)
  createStack(String, Color, double) : RayStack
  createStacks(String[], Color[], double[]) : List<RayStack>
```

4.5.7. Ray versus Bar Symbol

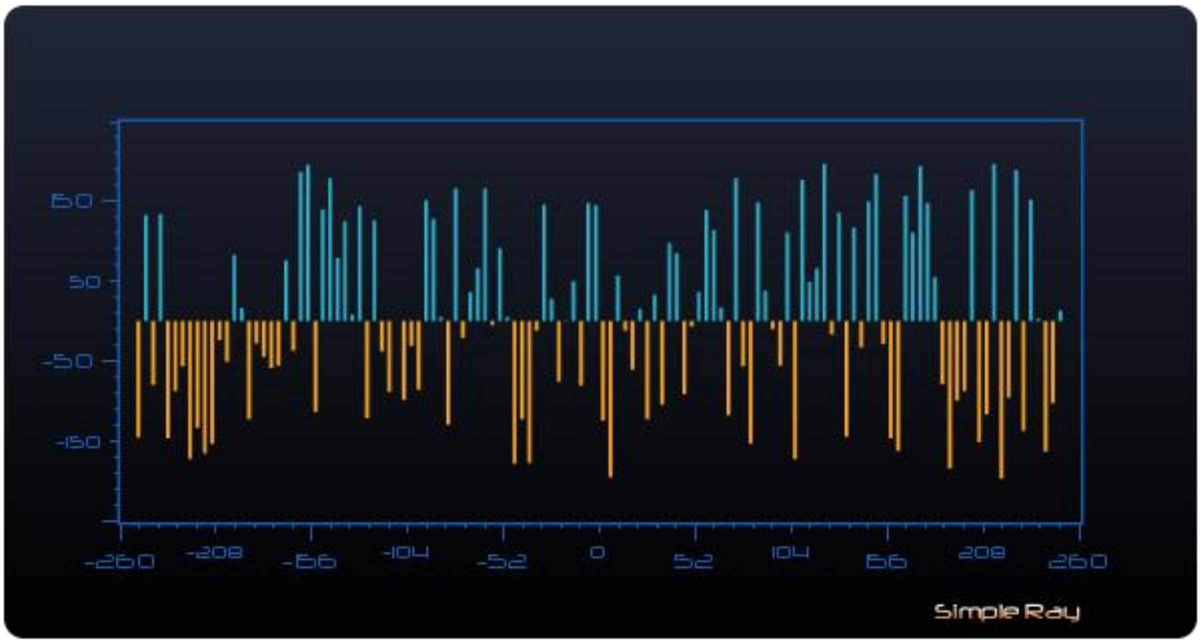
Sometimes charts can be made with symbol plug-in or ray plug-in as soon as you should have bar combo and sometimes you do not. For example for a scan component where you have to show some detected power channels for a specified frequency, you have to use ray plug-in but age people distribution chart can be achieve with ray plug-in or symbol plug-in.

4.5.8. Ray Samples

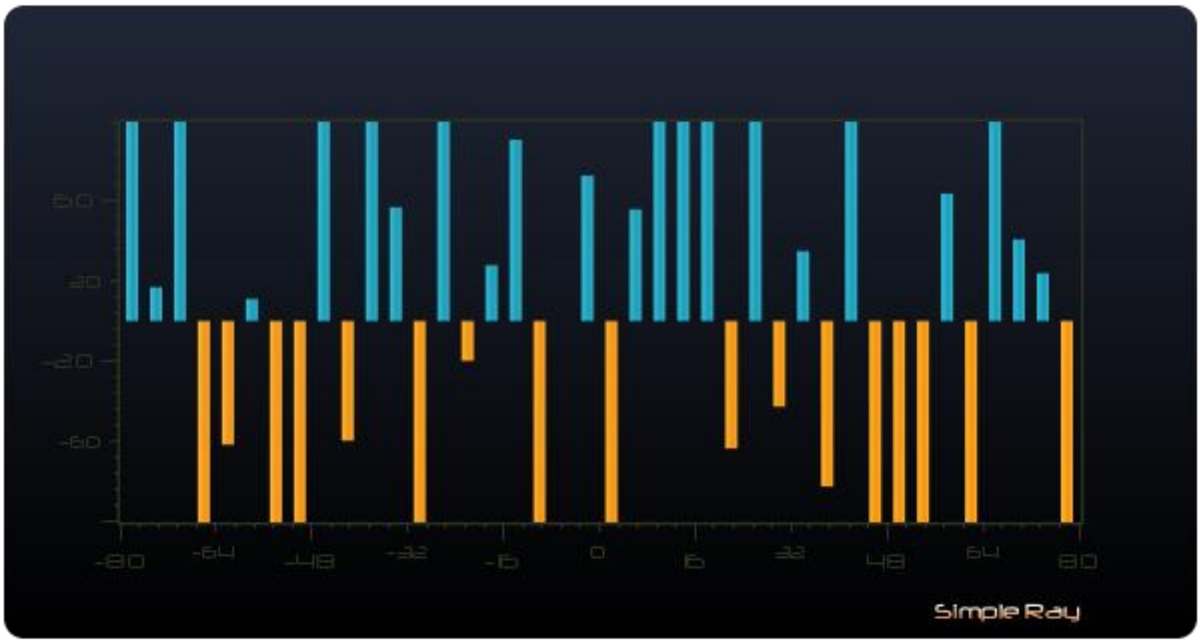
4.5.8.1.Simple Ray

Here an example with simple ray in ascent and descent mode and ray in user system projection.

```
/**
* Snippet code: create a ray in user coordinate system (thickness depends
* on the window bound)
*/
RayView view = RayToolkit.createCompatibleView(-260, 260, -250, 250);
Random r = new Random();
for (int i = -250; i <= 250; i = i + 4) {
	Ray ray = new Ray();
	ray.setName("ray" + i);
	ray.setThicknessType(ThicknessType.User);
	ray.setThickness(2);
	ray.setRayNature(RayNature.XRay);
	ray.setRayBase(0);
	int style = r.nextInt(2);
	if (style == 0) {
		ray.setAscentValue(r.nextInt(200));
		ray.setThemeColor(Spectral.SPECTRAL_BLUE1);
	} else {
		ray.setDescentValue(r.nextInt(200));
		ray.setThemeColor(NanoChromatique.ORANGE);
	}
	ray.setRay(i);
	ray.setRayFill(new RayFill1());
	view.addRay(ray);
}

view.registerPlugin(new OutlinePlugin());

// LEGEND
Legend legend = new Legend("Simple Ray");
legend.setLegendFill(new LegendFill1(Color.WHITE, JennyPalette.JENNY3));
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.South, 0.8f,
		LegendAlignment.Rigth));
LegendPlugin lgendL = new LegendPlugin();
lgendL.addLegend(legend);
view.registerPlugin(lgendL);

// TOOLS
ZoomBoxPlugin zoomTool = new ZoomBoxPlugin();
view.registerPlugin(zoomTool);

TranslatePlugin toolTranslate = new TranslatePlugin();
view.registerPlugin(toolTranslate);

ZoomWheelPlugin zoomWheel = new ZoomWheelPlugin();
view.registerPlugin(zoomWheel);
```

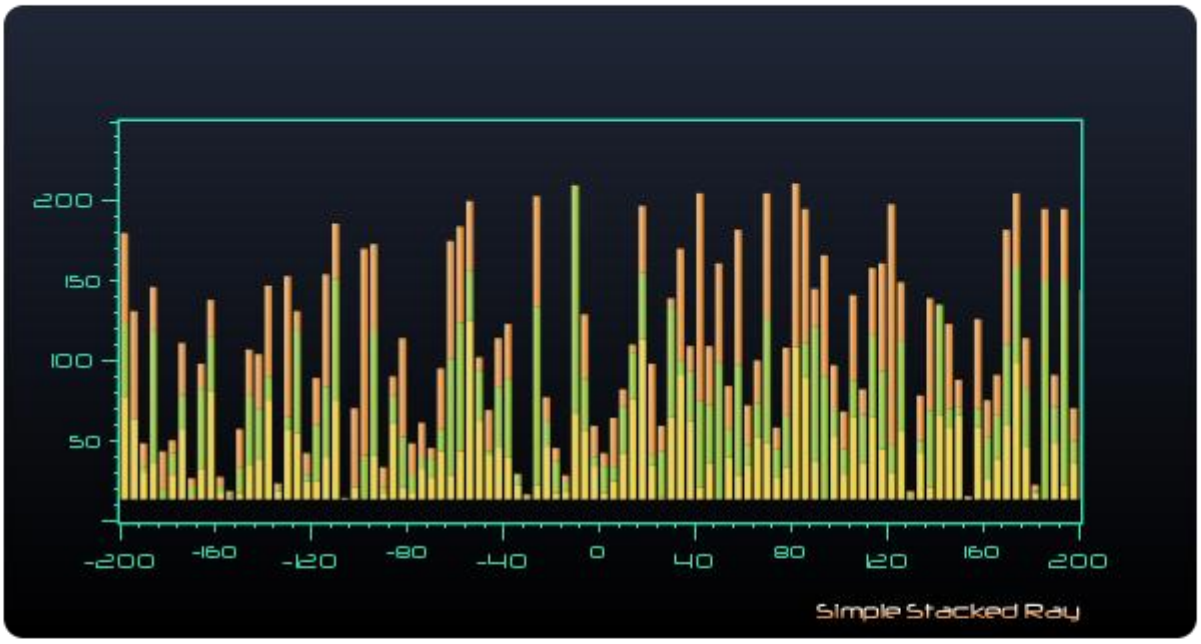


The same ray graph with a window lower bound (achieve by zoom plug-in).



4.5.8.2.Stacked Ray

The same type of randomized ray can be made.


200
150
100
50
-200
-160
-120
-80
-40
0
40
80
120
160
200
Simple Stacked Ray

```java
/**
* Snippet code: create a stacked ray chart
*/
RayView view = RayToolkit.createCompatibleView(-200, 200, 0, 250);

Random r = new Random();
Random r2 = new Random();

// CREATE STACKED RAY WITH RANDOM
for (int i = -250; i <= 250; i = i + 4) {

        StackedRay sray = new StackedRay();
        sray.setThicknessType(ThicknessType.User);
        sray.setThickness(3);
        sray.setRayNature(RayNature.XRay);
        sray.setRayBase(13);
        sray.setAscentValue(r.nextInt(200));
        sray.setRay(i);
        sray.setThemeColor(Color.WHITE);

        RayStack rs11 = new RayStack("ray 1 stack 1", TangoPalette.BUTTER3,
                    r2.nextInt(20));
        rs11.setRayFill(new RayFill2());

        RayStack rs12 = new RayStack("ray 1 stack 2",
                    TangoPalette.CHAMELEON3, r2.nextInt(20));
        rs12.setRayFill(new RayFill2());

        RayStack rs13 = new RayStack("ray 1 stack 3", TangoPalette.ORANGE3,
                    r2.nextInt(20));
        rs13.setRayFill(new RayFill2());

        sray.addStack(rs11);
        sray.addStack(rs12);
        sray.addStack(rs13);

        sray.setThemeColor(TangoPalette.BUTTER1);
        sray.setRayEffect(new RayEffect1());
        view.addRay(sray);
}
```

```java
/**
* Snippet code: create a stacked ray
*/
RayView view = RayToolkit.createCompatibleView(-200, 200, 0, 250);
Random r = new Random();
Random r2 = new Random();
for (int i = -250; i <= 250; i = i + 4) {
	StackedRay sray = new StackedRay();
	sray.setName("stacked ray 1");
	sray.setThicknessType(ThicknessType.User);
	sray.setThickness(3);
	sray.setRayNature(RayNature.XRay);
	sray.setRayBase(13);
	sray.setAscentValue(r.nextInt(200));
	sray.setRay(i);
	sray.setThemeColor(Color.WHITE);
	// sray.setRayDraw(new RayDefaultDraw(Color.WHITE));
	RayStack rs11 = new RayStack("ray 1 stack 1", TangoPalette.BUTTER3,
			r2.nextInt(20));
	rs11.setRayFill(new RayFill2());
	RayStack rs12 = new RayStack("ray 1 stack 2",
			TangoPalette.CHAMELEON3, r2.nextInt(20));
	rs12.setRayFill(new RayFill2());
	RayStack rs13 = new RayStack("ray 1 stack 3", TangoPalette.ORANGE3,
			r2.nextInt(20));
	rs13.setRayFill(new RayFill2());
	sray.addStack(rs11);
	sray.addStack(rs12);
	sray.addStack(rs13);
	sray.setThemeColor(TangoPalette.BUTTER1);
	sray.setRayEffect(new RayEffect1());
	view.addRay(sray);
}
// OUTLINE
view.registerPlugin(new OutlinePlugin());
// LEGEND
Legend legend = new Legend("Simple Stacked Ray");
legend.setLegendFill(new LegendFill1(Color.WHITE, JennyPalette.JENNY3));
legend.setFont(InputFonts.getFont(InputFonts.NEUROPOL, 12));
legend.setConstraints(new LegendConstraints(LegendPosition.South, 0.8f,
		LegendAlignment.Rigth));
LegendPlugin lgendL = new LegendPlugin();
lgendL.addLegend(legend);
view.registerPlugin(lgendL);

// TOOLS
ZoomBoxPlugin zoomTool = new ZoomBoxPlugin();
view.registerPlugin(zoomTool);

TranslatePlugin toolTranslate = new TranslatePlugin();
view.registerPlugin(toolTranslate);

ZoomWheelPlugin zoomWheel = new ZoomWheelPlugin();
view.registerPlugin(zoomWheel);
```

4.6. Plot

This core package is used by curve, area and scatter plug-ins. It provides some core concept like serie which is a set of points. There are some kinds of serie like interpolate or regression which use interpolator and curve spline fitting. This package provides an implementation abstract concept of metrics path, the *CurveMetricsPath* that draws metrics label along affine function path. Another spline path objects are provided.

4.6.1. Serie2D

A Serie is a set of points which can be used by curve, area or scatter curve. There are several kind of serie.

- *Serie2D* - Simple serie defines a set of point.
- *LinearRegressionSerie2D* - Linear regression serie defines a set of point which is modelled using linear predictor functions.
- *InterpolateSerie2D* - Interpolate serie defines a set of point which is fitting with spline curve interpolator.

```
/**
* Snippet code: example of point array
*/
double[] xValues1 = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
double[] yValues1 = { 6, 1.8, 15, 1.9, 3.4, 6.1, 4.2, 8.5, 9.9, 12, 8 };

double[] xValues2 = { 0, 1, 2, 3, 4, 5.2, 6, 7, 8, 9, 10 };
double[] yValues2 = { 3, 1, 5, 4, 4.8, 7.3, 2, 3, 7, 10, 6 };

double[] xValues3 = { 0, 1.4, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
double[] yValues3 = { 0.4, 7.5, 5, 1, 2.8, 5, 7, 9, 2, 4, 1 };
```

4.6.1.1.Line Serie

Create simple serie with *SerieToolkit* is quite simple.

```
/**
* Snippet code: Create line serie from point array
*/
Serie2D lineserie1 = SerieToolkit.createSerieFromArray(xValues1, yValues1);
Serie2D lineserie2 = SerieToolkit.createSerieFromArray(xValues2, yValues2);
Serie2D lineserie3 = SerieToolkit.createSerieFromArray(xValues3, yValues3);
```

4.6.1.2.Interpolate Serie

```
/**
* Snippet code: Create interpolate serie from point array
* and the epsilon step for interpolate serie.
*/
Serie2D serie1 = SerieToolkit.createInterpolateSerieFromArray(xValues1,
             yValues1, 0.1);
Serie2D serie2 = SerieToolkit.createInterpolateSerieFromArray(xValues2,
             yValues2, 0.1);
Serie2D serie3 = SerieToolkit.createInterpolateSerieFromArray(xValues3,
             yValues3, 0.1);
```

4.6.1.3.Linear regression serie

```
/**
* Snippet code: Create linear regression serie from point array
* and the epsilon to solve the simple regression
*/
Serie2D serie1 = SerieToolkit.createLinearRegressionSerieFromArray (xValues1,
             yValues1, 0.1);
Serie2D serie2 = SerieToolkit.createLinearRegressionSerieFromArray (xValues2,
             yValues2, 0.1);
Serie2D serie3 = SerieToolkit.createLinearRegressionSerieFromArray (xValues3,
             yValues3, 0.1);
```

4.6.2. Curve Metrics Path

A curve metrics path or *CurveMetricsPath* inherits from *AbstractMetricsPath.* The curve metrics path is created with a serie parameter argument constructor and it provides metrics label drawing mechanism along the curve path function. This object can be used by plug-in curve developers to make plug-in which use a curve metrics path and draw some labels annotations along affine curve path function.

```
/**
* Snippet code: Curve metrics path function
*/

Serie2D serie = new Serie2D(serie1);
InterpolateSerie2D io = new InterpolateSerie2D(serie1, 50);

GlyphFill gradientWhiteRed = new GlyphFill(Color.WHITE, Color.RED);
DefaultMarker markerWhite = new DefaultMarker(NanoChromatique.PURPLE);

CurveMetricsPath metricsPath = new CurveMetricsPath(io);
metricsPath.setPathPainter(new MetricsPathDefaultDraw(Color.YELLOW));

GlyphMetric metric = new GlyphMetric();
metric.setValue(0);
metric.setStylePosition(StylePosition.Default);
metric.setMetricsNature(GlyphMetricsNature.Median);
metric.setMetricsLabel("0");
metric.setDivergence(20);
metric.setGlyphMetricFill(gradientWhiteRed);
metric.setGlyphMetricMarkerPainter(markerWhite);
metric.setFont(InputFonts.getFont(InputFonts.ELEMENT, 14));

metricsPath.addMetrics(metric);
```

Line curve, area curve and scatter curve plug-ins already embeds this kind of metrics path, you can register glyph metrics on curve typed path. A glyph metrics is a glyph which is project on the path in the user projection system. For a x coordinate value, the y value is the projection f(x) for the specified serie. Plot tools like *CurveTracker* uses curve metrics as behind scene worker.

4.6.3. Plot Tool

4.6.3.1.Curve tracker

CurveTracker is a plug-in that takes the responsibility of some interactions with a serie like shows the y intercept coordinate for the current mouse drag or press position.

```java
/**
* Snippet code: serie tracker
*/
double[] xValues1 = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
double[] yValues1 = { 6, 1.8, 15, 1.9, 3.4, 6.1, 4.2, 8.5, 9.9, 12, 8 };

double[] xValues2 = { 0, 1, 2, 3, 4, 5.2, 6, 7, 8, 9, 10 };
double[] yValues2 = { 3, 1, 5, 4, 4.8, 7.3, 2, 3, 7, 10, 6 };

double[] xValues3 = { 0, 1.4, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
double[] yValues3 = { 0.4, 7.5, 5, 1, 2.8, 5, 7, 9, 2, 4, 1 };

// create a series
Serie2D serie1 = SerieToolkit.createInterpolateSerieFromArray(xValues1,
            yValues1, 0.1);
Serie2D serie2 = SerieToolkit.createInterpolateSerieFromArray(xValues2,
            yValues2, 0.1);
Serie2D serie3 = SerieToolkit.createInterpolateSerieFromArray(xValues3,
            yValues3, 0.1);

SerieTrackerPlugin curveTrackerPlugin = new SerieTrackerPlugin();
curveTrackerPlugin.registerSerie(serie1);
curveTrackerPlugin.registerSerie(serie2);
curveTrackerPlugin.registerSerie(serie3);

curveTrackerPlugin.registerContext(new SerieTrackerDeviceContext());
window.registerPlugin(curveTrackerPlugin);
```

If you are interesting to handles some events from tracker, you should have to register a serie tracker listener like shown above.

```java
/**
* Snippet code: serie tracker listener
*/

curveTrackerPlugin.addSerieTrackerListener(new SerieTrackerListener() {

      @Override
      public void serieTracked(SerieTrackerEvent event) {
            System.out.println("serieTracked ::"+event.getSerie().getName());
      }

      @Override
      public void serieRegistered(SerieTrackerEvent event) {
            System.out.println("serieRegistered ::"+event.getSerie().getName());

      }

      @Override
      public void currentTrack(SerieTrackerEvent event) {
            System.out.println("currentTrack::"+event.getSerie().getName());
            Point2D devicePoint = curveTrackerPlugin.getCurrentTrackDevice();
            Point2D userPoint = curveTrackerPlugin.getCurrentTrackUser();
            System.out.println("serieTracked device::"+devicePoint);
            System.out.println("serieTracked user::"+userPoint);
      }
});
```

4.6.3.2.Peak Tracker

Peak track is a variant of the curve metrics tracker which track peak with peak frame painting.

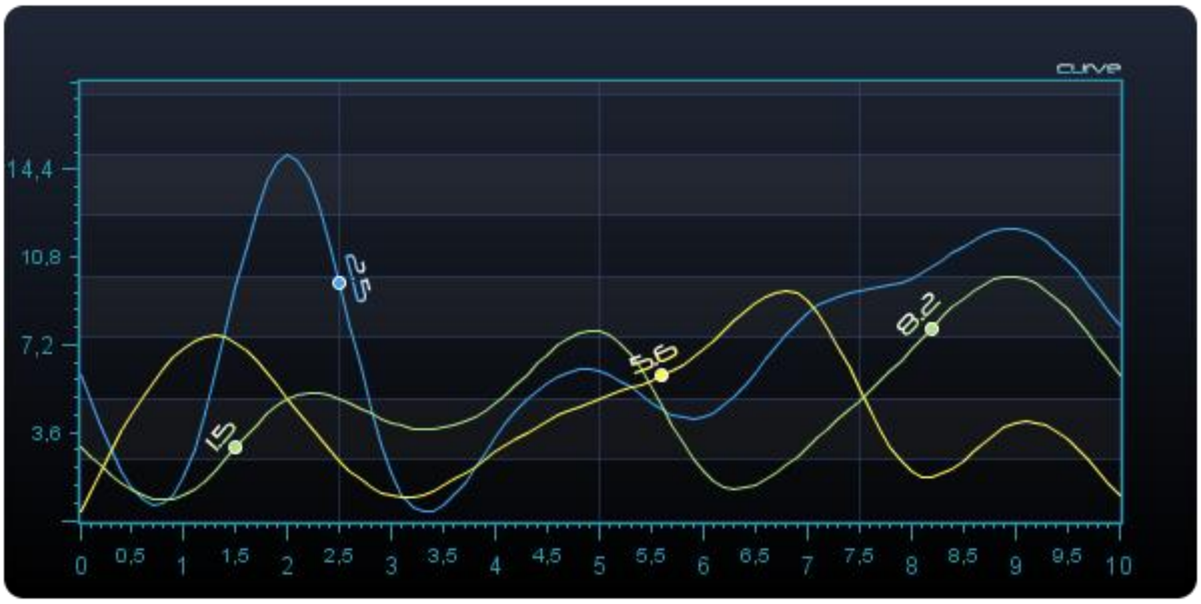
4.7. Curve

```
package com.jensoft.sw2d.core.plugin.curve;
```

A curve chart is a type of chart, which displays information as a series of data points connected by straight line segments (line chart) or smoothing segment (spline chart). It is an extension of a scatter chart, and is created by connecting a series of points that represent individual measurements with line or spline segments. In the experimental sciences, data collected from experiments are often visualized by a graph that includes an overlaid mathematical function depicting the best-fit trend of the scattered data. This layer is referred to as a best-fit layer and the graph containing this layer is often referred to as a line graph.

To draw fast, use simple serie. Obtain a curve chart with good looking, use spline interpolation. In computer graphics splines are popular curves because of the simplicity of their construction, their ease and accuracy of evaluation, and their capacity to approximate complex shapes through curve fitting and interactive curve design.

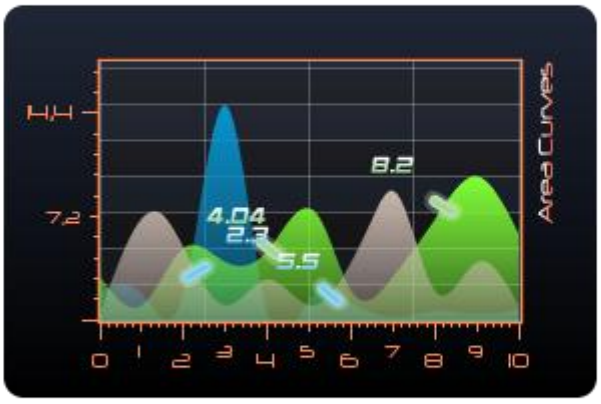
The figure shows the case of a multiple curves chart in the same window projection and some label annotations. The metrics glyph corresponds to the x projection of a specified glyph metrics. The label is default set to the y value and can be choose by user.



4.8. Area

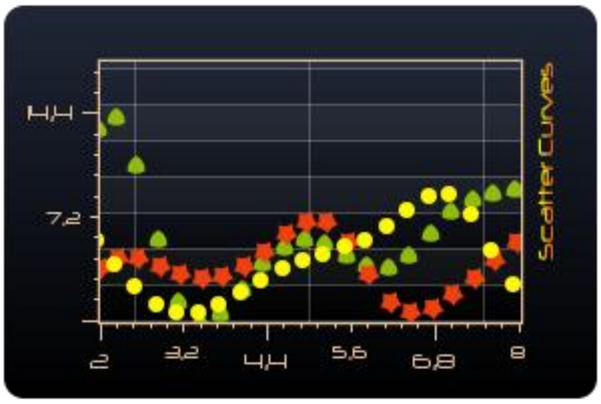
An area chart or area graph displays graphically quantitive data. It is based on the line chart. The area between axis and line are commonly emphasized with colors, textures and hatchings. Commonly one compares with an area chart two or more quantities. Area charts are used to represent cumulated totals using numbers or percentages (stacked area charts in this case) over time.

To draw fast, use simple serie. Obtain a curve chart with good looking, use spline interpolation. In computer graphics splines are popular curves because of the simplicity of their construction, their ease and accuracy of evaluation, and their capacity to approximate complex shapes through curve fitting and interactive curve design. The most commonly used splines are cubic spline, i.e., of order 3 - in particular, cubic B-spline and cubic Bézier spline. They are common, in particular, in spline interpolation simulating the function of flat splines.



4.9. Scatter

A scatter plot or scatter graph is a type of mathematical diagram using Cartesian coordinates to display values for two variables for a set of data. The data is displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis. This kind of plot is also called a scatter chart, scatter gram, scatter diagram or scatter graph.



4.10. Radar

4.10.1. Radar Plug-in

4.10.2. Radar Dimension

4.10.3. Surface

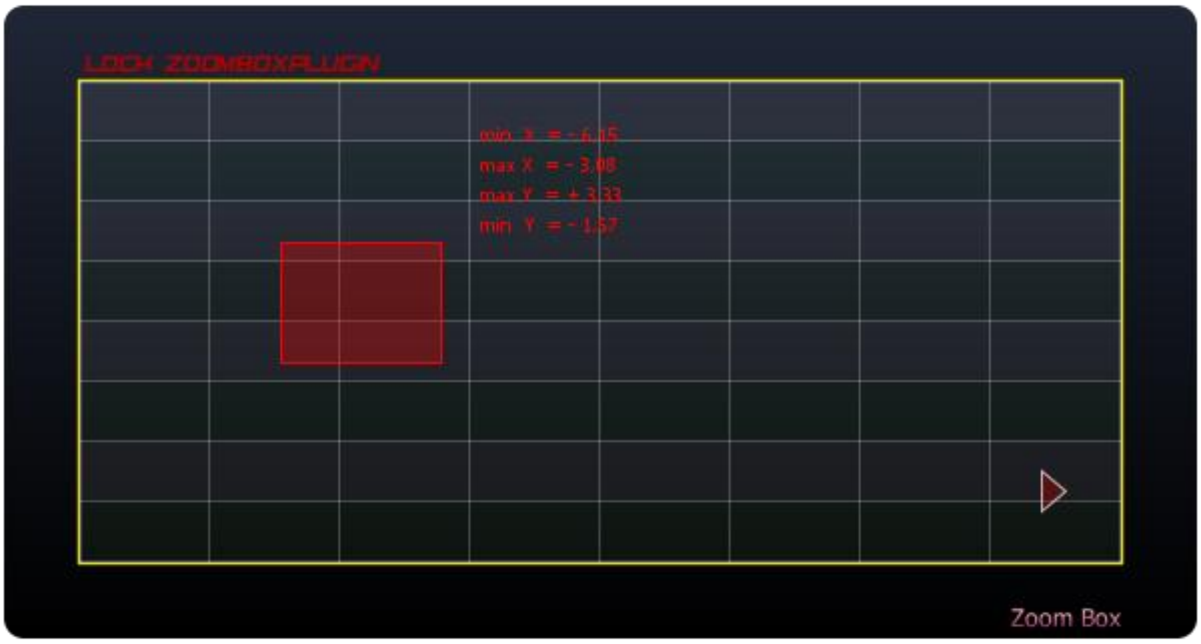
5. Tool oriented plug-ins

5.1. Zoom Box

Zoom Box Plug in provides a zooming box that allow to choose a window area to zoom in.

5.1.1. Zoom Box Frame Box

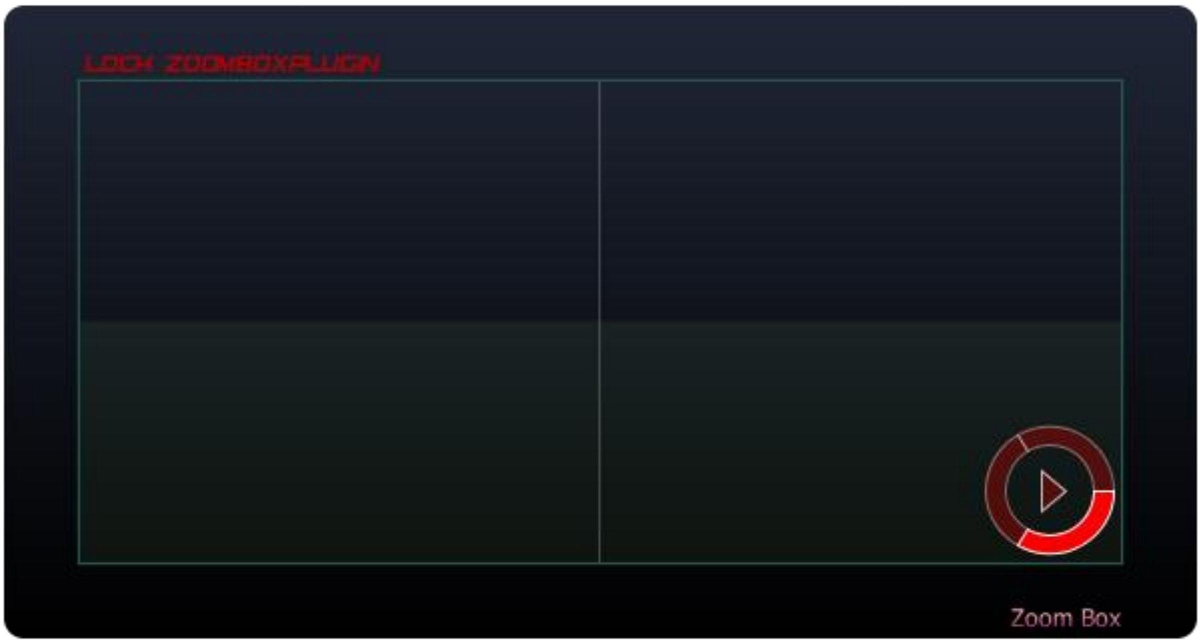
ZoomBoxPlugin is a selectable plug-in that need a lock to allow zooming operation. The lock can be lock with *lockSelected* method or with the menu item of the plug in device context menu. When the plug in is selected, a view area can be zoomed. On left click on the view and drag cursor, you can create a bound box which will be zoomed on mouse released.



5.1.2. Zoom Box Widget

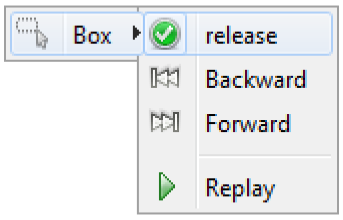
The zoom box device widget is a donut morphe widget. Each slice represents a zoom bounding box which can be shown on a slice clicked. You can play the zoom sequence with the center button.

```
/**
* Snippet code: create zoom box widget
*/
ZoomBoxPlugin zoomBox = new ZoomBoxPlugin();
zoomBox.setThemeColor(Color.RED);
ZoomBoxDonutWidget zoomBoxDonutWidget = new ZoomBoxDonutWidget();
zoomBox.registerWidget(zoomBoxDonutWidget);
w2d.registerPlugin(zoomBox);
```



5.1.3. Zoom Box Device Context

The figure below shows the default zoom box device context. The box menu item provides the lock selected action. Zoom box plug-in is a selectable plug-in that required a lock to make zoom operation available. You can backward and forward zoom and replay all zoom sequence.



Labels for device context menu can be set and default label value will be replaced by the provided labels.

5.1.4. Zoom Synchronizer

Zoom synchronizer provides zoom box synchronization between view.

```
/**
* Snippet code: create zoom box synchronizer.
* Assume you have two views and a zoom instance on each window
*/

ZoomBoxSynchronizer s = ZoomBoxPlugin.createSynchronizer(zoomBox1,zoomBox2);
```

The static method returns the synchronizer which can be enabled or disabled.

5.2. Zoom Objectif

Zoom objectif provides lazy zoom for each dimension x and y.

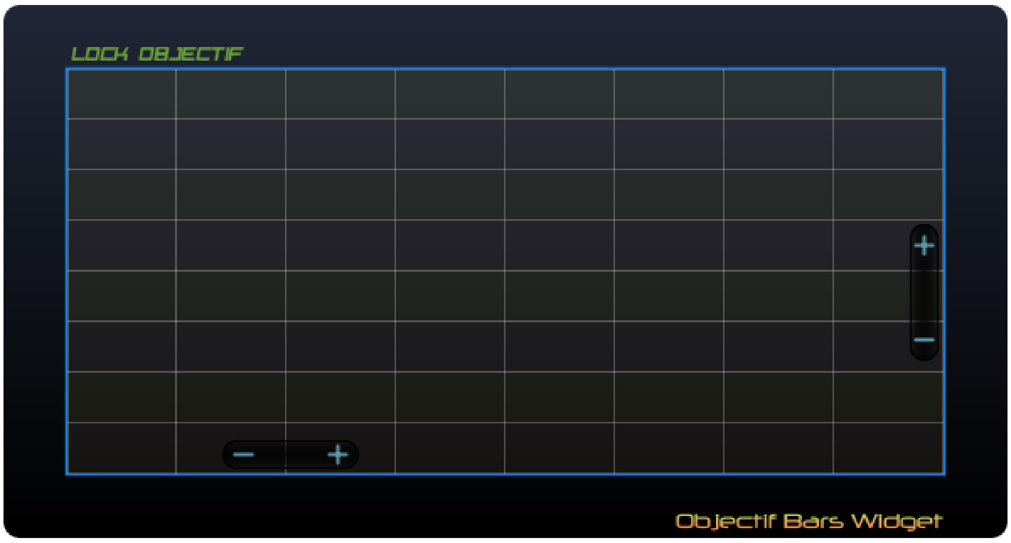
```
/**
* Snippet code: register zoom objectif
*/
ZoomObjectifPlugin zoomObjectif = new ZoomObjectifPlugin();
w2d.registerPlugin(zoomObjectif);
```

5.2.1. Objectif Widgets

Objectif provides two type of widgets, the bars widget for each x and y dimensions and a pad for x and y dimensions.

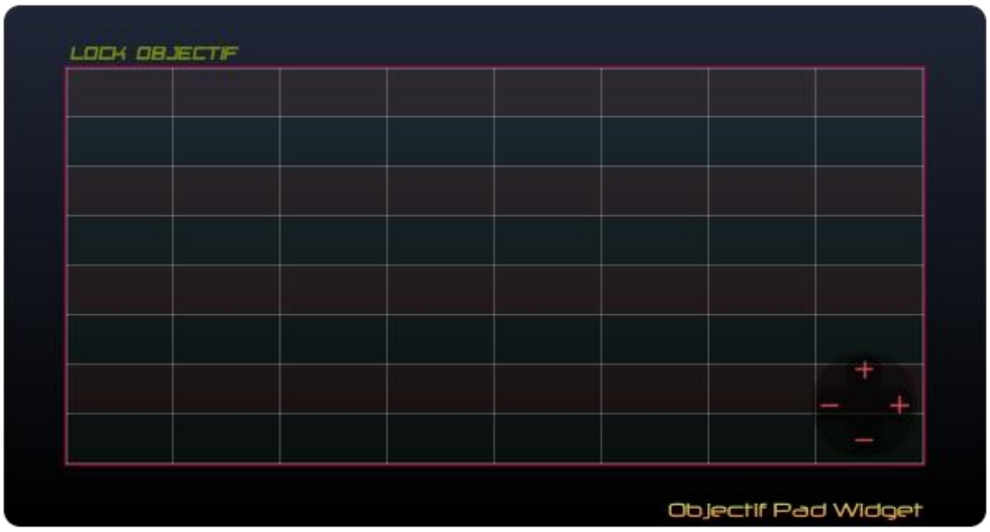
5.2.1.1.Bar Widgets

```
/** Snippet code: register bar widget on objectif widget */
ZoomObjectifPlugin zoomObjectif = new ZoomObjectifPlugin();
// create two objectif for x and y dimension
ObjectifX ox = new ObjectifX();
ObjectifY oy = new ObjectifY();
// register widget in zoom objectif plugin
zoomObjectif.registerWidget(ox);
zoomObjectif.registerWidget(oy);

ox.setOutlineColor(Color.BLACK);
ox.setButton1DrawColor(RosePalette.CALYPSOBLUE);
ox.setButton2DrawColor(RosePalette.CALYPSOBLUE);
oy.setOutlineColor(Color.BLACK);
oy.setButton1DrawColor(RosePalette.CALYPSOBLUE);
oy.setButton2DrawColor(RosePalette.CALYPSOBLUE);

w2d.registerPlugin(zoomObjectif);
```



5.2.1.2.Pad Widget

```
/**
* Snippet code: register pad widget on objectif widget
*/
ZoomObjectifPlugin zoomObjectif = new ZoomObjectifPlugin();
w2d.registerPlugin(zoomObjectif);
// use pad widget with objectif
ObjectifPad opad = new ObjectifPad();
opad.setFillBaseColor(new Color(0, 0, 0, 90));
opad.setFillControlColor(new Color(0, 0, 0, 100));
opad.setDrawBaseColor(null);
opad.setDrawControlColor(null);
zoomObjectif.registerWidget(opad);
// pre lock selected
zoomObjectif.lockSelected();
```



5.2.2. Objectif device context

The objectif device context menu only provides the lock select for objectif.

```
/**
* Snippet code: register objectif device context menu
*/
zoomObjectif.registerContext(new ObjectifDefaultDeviceContext());
```

5.3. Zoom Wheel

Zoom wheel provides a zoom when mouse wheel occurs.

```
/**
* Snippet code: register zoom wheel
*/
ZoomWheelPlugin zoomWheel = new ZoomWheelPlugin();
w2d.registerPlugin(zoomWheel);
```

You can create zoom wheel synchronizer for different views.

```
/**
* Snippet code: create zoom wheel synchronizer
*/
ZoomWheelSynchronizer s = ZoomWheelPlugin.createSynchronizer(wheel1,wheel2);
```

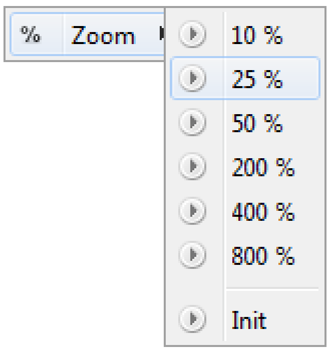
5.4. Zoom Percent

Zoom percent provides a zoom based on percent current based dimension.

```
/**
* Snippet code: register zoom percent
*/

ZoomPercentPlugin zoomPercent = new ZoomPercentPlugin();
w2d.registerPlugin(zoomPercent);
```

5.4.1. Zoom Percent device context

```
/**
* Snippet code: register zoom percent device context
*/
ZoomPercentPlugin zoomPercent = new ZoomPercentPlugin();
zoomPercent.registerContext(new ZoomPercentDefaultDeviceContext());
w2d.registerPlugin(zoomPercent);
```



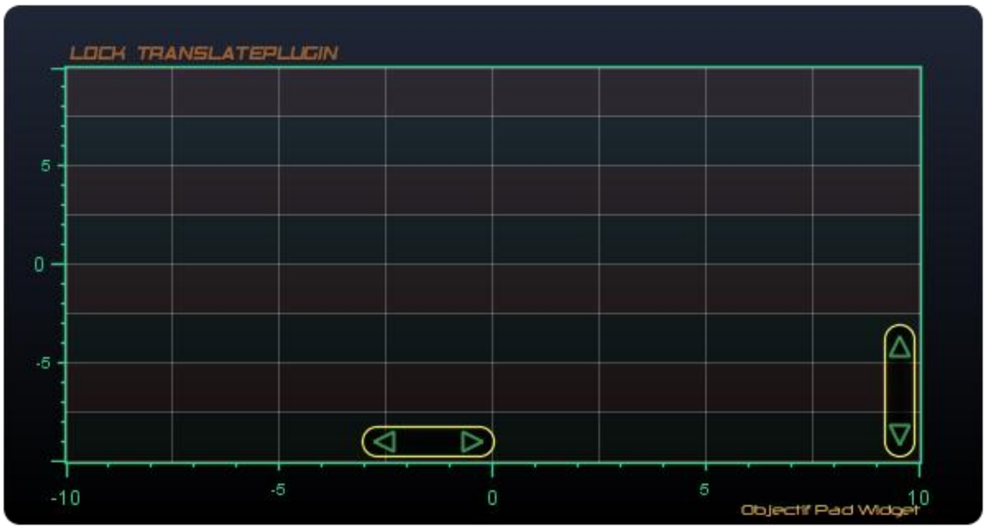
5.5. Translate

Translate plug-in provides user translate operation. Translate is a selectable plug-in that needs a lock to work. The lock can be acquired be calling programmatically *lockSelected*() method or use the lock item action in the in the device context menu.

5.5.1. Translate Widgets

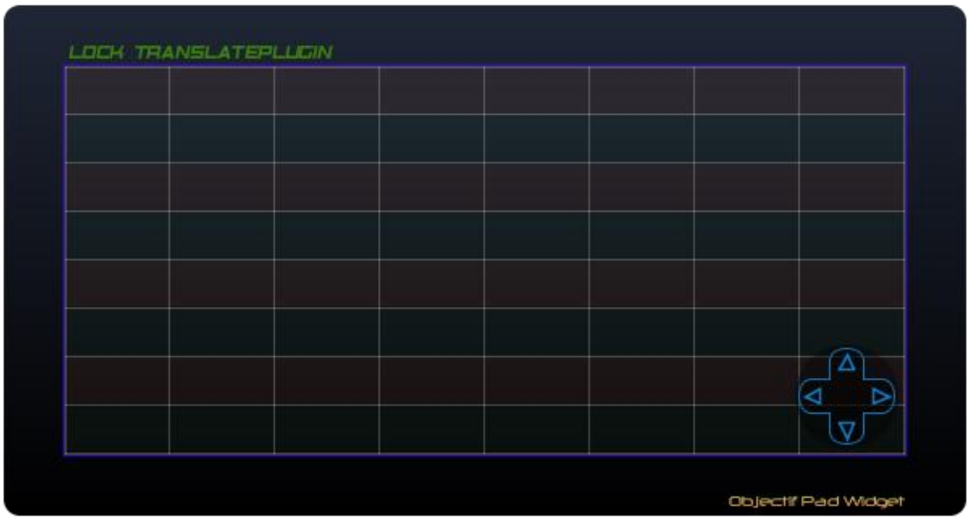
Translate provides two widgets, the bars widgets and a pad widget. Widget allows translate window shifting. You can set colors for outline, background, and draw or fill button (default, rollover) etc.

5.5.1.1.Bars Widgets



```
/**
* Snippet code: register translate bars widgets
*/

// TRANSLATE
TranslatePlugin translatePlugin = new TranslatePlugin();
w2d.registerPlugin(translatePlugin);

// use pad widget with objectif
TranslateX tx = new TranslateX();
TranslateY ty = new TranslateY();
translatePlugin.registerWidget(tx, ty);

// pre lock selected
translatePlugin.lockSelected();
```

5.5.1.2.Pad Widget

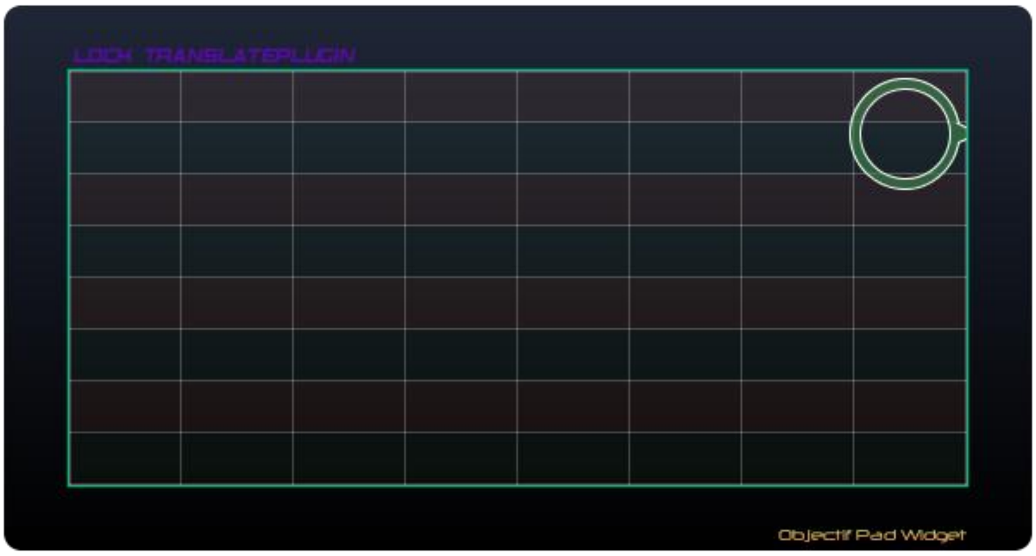


```
/**
* Snippet code: register translate pad widget
*/

// TRANSLATE
TranslatePlugin translatePlugin = new TranslatePlugin();
w2d.registerPlugin(translatePlugin);

// use pad widget with objectif
TranslatePad tPad = new TranslatePad();
tPad.setFillBaseColor(new Color(0, 0, 0, 90));
tPad.setFillControlColor(new Color(0, 0, 0, 100));
translatePlugin.registerWidget(tPad);
tPad.setDrawControlColor(Spectral.SPECTRAL_BLUE2);
tPad.setDrawButtonColor(Spectral.SPECTRAL_BLUE2);
// pre lock selected
translatePlugin.lockSelected();
```

5.5.1.3.Compass Widget

During translation a compass widget appears if you have registered the compass widget.

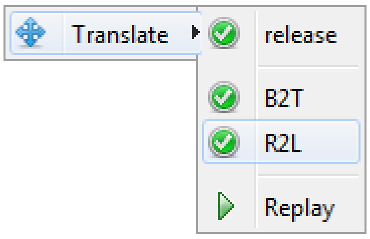
```
/**
* Snippet code: register translate compass widget
*/
TranslatePlugin translatePlugin = new TranslatePlugin();
w2d.registerPlugin(translatePlugin);


TranslateCompass compass = new TranslateCompass();
translatePlugin.registerWidget(compass);
compass.setCompassStyle(CompassStyle.Merge);
compass.setRingFillColor(new Alpha(RosePalette.EMERALD, 150));
compass.setRingDrawColor(Color.WHITE);
compass.setRingNeedleFillColor(new Alpha(RosePalette.EMERALD, 150));
compass.setRingNeedleDrawColor(Color.WHITE);
```



5.5.2. Translate device context

The translate device context provides followings actions: lock translate, allow bottom to top translate (only vertical translation), right to left translate (horizontal translation) and replay last translate operation.

```
/**
* Snippet code: register translate device context
*/
TranslatePlugin translatePlugin = new TranslatePlugin();
w2d.registerPlugin(translatePlugin);

translatePlugin.registerContext(new TranslateDefaultDeviceContext());
```



5.6. Marker

5.7. Serie Tracker

5.8. Peack Tracker

6. Instrument oriented Plug-ins

6.1. Gauge

6.2. General Metrics Path

7. X2D

To see to x2d schema, click on the link below:

http://www.jensoft.org/jensoft/schema/x2d/x2d.xsd

8. JET

9. Framework Development

9.1. Plug-in

Create a plug-in in JenSoft API is quiet simple. You have to create a class that inherits from *AbtractPlugin* which provides an abstract plug-in definition. An implementation of this class should be supply the method paint plug-in method. This method is called by each window part component. Each plug-in can be paint the related part component according to rules it defines and general plug-in behaviors. For example, *ZoomBox, ZoomObjectif, Translate* plug-ins are selectable plug-in that use a lock. If a selectable plug-in get the lock, if another plug-in has a lock, it is released. This mechanism allows avoiding plug-ins conflicts. For example *ZoomBox* and *Translate* are listening a mouse pressed event for starting operation, then they cannot work together.

9.2. Widget

9.3. Device Context Menu