

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 7](#_Toc336187899)

[1.1. SW2D Framework background 7](#_Toc336187900)

[1.1.1. Framework overview 8](#_Toc336187901)

[1.1.2. When to use SW2D? 8](#_Toc336187902)

[1.1.3. Technical logic design 8](#_Toc336187903)

[1.1.4. Technical physical design 9](#_Toc336187904)

[1.2. Getting started 9](#_Toc336187905)

[1.2.1. Download SW2D 9](#_Toc336187906)

[1.2.2. Veteran Tutorial 10](#_Toc336187907)

[2. Framework model 10](#_Toc336187908)

[2.1. Presentation 10](#_Toc336187909)

[2.2. View2D 11](#_Toc336187910)

[2.2.1. Create a view 11](#_Toc336187911)

[2.2.2. View2D API 11](#_Toc336187912)

[2.2.3. View Background 12](#_Toc336187913)

[2.2.3.1. Fill Background 12](#_Toc336187914)

[2.2.3.2. Image Background 12](#_Toc336187915)

[2.3. Window2D 13](#_Toc336187916)

[2.3.1. Create window2D 13](#_Toc336187917)

[2.3.2. Window API 14](#_Toc336187918)

[2.4. Plug-in 14](#_Toc336187919)

[3. Common Plug-ins 15](#_Toc336187920)

[3.1. Outline 15](#_Toc336187921)

[3.2. Device Image Background 16](#_Toc336187922)

[3.3. Text Legend 16](#_Toc336187923)

[3.4. Metrics 17](#_Toc336187924)

[3.4.1. Metrics Manager 18](#_Toc336187925)

[3.4.2. Metrics plug-in 18](#_Toc336187926)

[3.4.2.1. Axis Metrics Plug-ins 18](#_Toc336187927)

[3.4.2.2. Device Metrics Plug-ins 19](#_Toc336187928)

[3.4.2.3. Metrics Properties 19](#_Toc336187929)

[3.4.2.4. Metrics format 20](#_Toc336187930)

[3.4.3. Metrics samples 21](#_Toc336187931)

[3.4.3.1. Axis Free metrics 21](#_Toc336187932)

[3.4.3.2. Axis Static Metrics 22](#_Toc336187933)

[3.4.3.3. Axis Dynamic Metrics 23](#_Toc336187934)

[3.4.3.4. Axis Milli Metrics 24](#_Toc336187935)

[3.4.3.5. Device Milli Metrics 25](#_Toc336187936)

[3.4.3.6. Multiple Device Axis 26](#_Toc336187937)

[3.5. Grid 28](#_Toc336187938)

[3.5.1. Grid Manager 28](#_Toc336187939)

[3.5.2. Grid plug-in 28](#_Toc336187940)

[3.5.3. Grid properties 29](#_Toc336187941)

[3.5.4. Grid Samples 29](#_Toc336187942)

[3.5.4.1. Static grid 29](#_Toc336187943)

[3.5.4.2. Free Grid 30](#_Toc336187944)

[3.5.4.3. Flow grid 32](#_Toc336187945)

[3.5.4.4. Dynamic grid 33](#_Toc336187946)

[3.5.4.5. Free Grid on Log Projection 34](#_Toc336187947)

[3.6. Stripe 35](#_Toc336187948)

[3.6.1. Stripe Manager 35](#_Toc336187949)

[3.6.2. Stripe plug-in 36](#_Toc336187950)

[3.6.3. Stripe Sample 36](#_Toc336187951)

[3.6.3.1. Dynamic Stripes 36](#_Toc336187952)

[3.6.3.2. Textured Stripes 38](#_Toc336187953)

[4. Data oriented plug-ins 39](#_Toc336187954)

[4.1. Pie Chart 39](#_Toc336187955)

[4.1.1. Pie 40](#_Toc336187956)

[4.1.2. Slice 42](#_Toc336187957)

[4.1.3. Pie Label 42](#_Toc336187958)

[4.1.3.1. Pie Bound Label 43](#_Toc336187959)

[4.1.3.2. Pie Border Label 44](#_Toc336187960)

[4.1.3.3. Pie Radial Label 45](#_Toc336187961)

[4.1.3.4. Pie Path Label 46](#_Toc336187962)

[4.1.4. Pie Effect 46](#_Toc336187963)

[4.1.4.1. Pie Linear Effect 46](#_Toc336187964)

[4.1.4.2. Pie Radial Effect 47](#_Toc336187965)

[4.1.5. Pie Listener 47](#_Toc336187966)

[4.1.6. Pie Animator 47](#_Toc336187967)

[4.1.6.1. Pie Alpha animator 47](#_Toc336187968)

[4.1.6.2. Pie Divergence Animator 48](#_Toc336187969)

[4.1.6.3. Pie Flash Animator 48](#_Toc336187970)

[4.1.7. Pie Toolkit 48](#_Toc336187971)

[4.1.8. Multiple Pie 49](#_Toc336187972)

[4.2. Pie 3D Chart 52](#_Toc336187973)

[4.2.1. Pie 3D 52](#_Toc336187974)

[4.2.2. Pie 3D Slice 53](#_Toc336187975)

[4.2.3. Pie 3D Label 54](#_Toc336187976)

[4.2.3.1. Pie 3D Radial Label 54](#_Toc336187977)

[4.2.3.2. Pie 3D Border Label 55](#_Toc336187978)

[4.2.3.3. Pie 3D Path Label 56](#_Toc336187979)

[4.2.4. Pie 3D Paint 56](#_Toc336187980)

[4.2.5. Pie 3D Listener 57](#_Toc336187981)

[4.2.6. Pie 3D Animator 58](#_Toc336187982)

[4.2.6.1. Pie 3D Divergence Animator 58](#_Toc336187983)

[4.2.6.2. Pie 3D Label Animator 59](#_Toc336187984)

[4.2.7. Pie 3D Toolkit 59](#_Toc336187985)

[4.3. Donut 2D Chart 60](#_Toc336187986)

[4.3.1. Donut2D 60](#_Toc336187987)

[4.3.2. Donut 2D Slicing 61](#_Toc336187988)

[4.3.3. Donut 2D Label 62](#_Toc336187989)

[4.3.3.1. Donut 2D Radial Label 62](#_Toc336187990)

[4.3.3.2. Donut 2D Border Label 63](#_Toc336187991)

[4.3.3.3. Donut 2D Path Label 63](#_Toc336187992)

[4.3.4. Donut2D Fill 65](#_Toc336187993)

[4.3.4.1. Default Fill 65](#_Toc336187994)

[4.3.4.2. Radial Fill 65](#_Toc336187995)

[4.3.5. Donut 2D Listener 66](#_Toc336187996)

[4.3.6. Donut 2DAnimator 66](#_Toc336187997)

[4.3.6.1. Donut2D Divergence Animator 66](#_Toc336187998)

[4.3.6.2. Donut2D Label Animator 67](#_Toc336187999)

[4.3.7. Donut2D Toolkit 67](#_Toc336188000)

[4.3.8. Multiple Donut 2D 68](#_Toc336188001)

[4.4. Symbols 69](#_Toc336188002)

[4.4.1. Symbol Plug-in 70](#_Toc336188003)

[4.4.2. Bar Symbol Components 71](#_Toc336188004)

[4.4.2.1. Bar Layer 71](#_Toc336188005)

[4.4.2.2. Simple Bar Symbol 72](#_Toc336188006)

[4.4.2.3. Stacked Bar Symbol 73](#_Toc336188007)

[4.4.2.4. Bar Symbol Group 75](#_Toc336188008)

[4.4.2.5. Bar Symbol Effect 76](#_Toc336188009)

[4.4.2.6. Bar Symbol Label 78](#_Toc336188010)

[4.4.2.7. Bar Listener 84](#_Toc336188011)

[4.4.3. Point & Curve Symbol 85](#_Toc336188012)

[4.4.3.1. Point Layer 86](#_Toc336188013)

[4.4.3.2. Point Symbol 86](#_Toc336188014)

[4.4.3.3. Point Painter 86](#_Toc336188015)

[4.4.3.4. Polyline Symbol 87](#_Toc336188016)

[4.4.3.5. Polyline Painter 89](#_Toc336188017)

[4.4.3.6. Point Listener 90](#_Toc336188018)

[4.4.4. Symbol toolkit 91](#_Toc336188019)

[4.4.5. Sample Symbols 92](#_Toc336188020)

[4.4.5.1. Vertical Ascent and Descent Bar Symbol 92](#_Toc336188021)

[4.4.5.2. Horizontal Gant Type 93](#_Toc336188022)

[4.4.5.3. Country Age pyramid 94](#_Toc336188023)

[4.4.5.4. Bar Animator 99](#_Toc336188024)

[4.4.5.5. Climate Bar Symbol and point curve symbols 102](#_Toc336188025)

[4.5. Ray 111](#_Toc336188026)

[4.5.1. Ray Plug-in 111](#_Toc336188027)

[4.5.2. Ray 112](#_Toc336188028)

[4.5.3. Stacked ray 112](#_Toc336188029)

[4.5.4. Ray Group 113](#_Toc336188030)

[4.5.5. Ray Listener 114](#_Toc336188031)

[4.5.6. Ray Toolkit 115](#_Toc336188032)

[4.5.7. Ray versus Bar Symbol 116](#_Toc336188033)

[4.5.8. Ray Samples 116](#_Toc336188034)

[4.5.8.1. Simple Ray 116](#_Toc336188035)

[4.5.8.2. Stacked Ray 118](#_Toc336188036)

[4.6. Plot 122](#_Toc336188037)

[4.6.1. Serie2D 122](#_Toc336188038)

[4.6.1.1. Line Serie 122](#_Toc336188039)

[4.6.1.2. Interpolate Serie 122](#_Toc336188040)

[4.6.1.3. Linear regression serie 123](#_Toc336188041)

[4.6.2. Curve Metrics Path 123](#_Toc336188042)

[4.6.3. Plot Tool 124](#_Toc336188043)

[4.6.3.1. Curve tracker 124](#_Toc336188044)

[4.6.3.2. Peak Tracker 126](#_Toc336188045)

[4.7. Curve 126](#_Toc336188046)

[4.8. Area 127](#_Toc336188047)

[4.9. Scatter 128](#_Toc336188048)

[4.10. Radar 128](#_Toc336188049)

[4.10.1. Radar Plug-in 128](#_Toc336188050)

[4.10.2. Radar Dimension 128](#_Toc336188051)

[4.10.3. Surface 128](#_Toc336188052)

[5. Tool oriented plug-ins 128](#_Toc336188053)

[5.1. Zoom Box 128](#_Toc336188054)

[5.1.1. Zoom Box Frame Box 129](#_Toc336188055)

[5.1.2. Zoom Box Widget 129](#_Toc336188056)

[5.1.3. Zoom Box Device Context 130](#_Toc336188057)

[5.1.4. Zoom Synchronizer 130](#_Toc336188058)

[5.2. Zoom Objectif 131](#_Toc336188059)

[5.2.1. Objectif Widgets 131](#_Toc336188060)

[5.2.1.1. Bar Widgets 131](#_Toc336188061)

[5.2.1.2. Pad Widget 132](#_Toc336188062)

[5.2.2. Objectif device context 133](#_Toc336188063)

[5.3. Zoom Wheel 133](#_Toc336188064)

[5.4. Zoom Percent 133](#_Toc336188065)

[5.4.1. Zoom Percent device context 133](#_Toc336188066)

[5.5. Translate 134](#_Toc336188067)

[5.5.1. Translate Widgets 134](#_Toc336188068)

[5.5.1.1. Bars Widgets 134](#_Toc336188069)

[5.5.1.2. Pad Widget 135](#_Toc336188070)

[5.5.1.3. Compass Widget 136](#_Toc336188071)

[5.5.2. Translate device context 136](#_Toc336188072)

[5.6. Marker 137](#_Toc336188073)

[5.7. Serie Tracker 137](#_Toc336188074)

[5.8. Peack Tracker 137](#_Toc336188075)

[6. Instrument oriented Plug-ins 137](#_Toc336188076)

[6.1. Gauge 137](#_Toc336188077)

[6.2. General Metrics Path 137](#_Toc336188078)

[7. X2D 137](#_Toc336188079)

[8. JET 137](#_Toc336188080)

[9. Framework Development 138](#_Toc336188081)

[9.1. Plug-in 138](#_Toc336188082)

[9.2. Widget 138](#_Toc336188083)

[9.3. Device Context Menu 138](#_Toc336188084)

# Introduction

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

## SW2D Framework background

### 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.

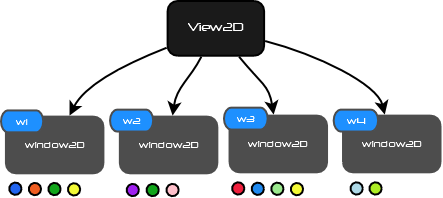
JET (JENSOFT Emitter Template) is a part of JENSOFT API which takes the responsibility of inflate xml document and emits view as various format like view, view as image, view as xml document, view as image file. X2D is the sw2d schema language and it used by JET to validate view document.

### When to use SW2D?

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

### 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!

### Technical physical design

As a component point of view, the *View2D* isa 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.

## 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.

### 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.

### 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.



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.

# Framework model

## Presentation

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

## View2D

View2D is the first API sw2d object that you have to create instance. The view is the end user chart scene. 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. Four outer parts components which are called south, north, west and east and one inner component part which is called the device.

### 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 or you must paint a lot of labels.



### View2D API

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



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.



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.

### View Background

All painting mechanism is managed by plug-ins but for the view background, framework provides view background painter. 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.

#### Fill Background

This background fills the view with a gradient. You can set properties like stroke, fill colors, outline color and corner arc radius.



#### 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.





## Window2D

The abstract window defines a user projection on X and Y dimensions. Windows 2D brings plus-ins that process painting operations in view or handle user interaction. Window instance has to be registered in a view. The window has different kind of nature:

* *Window2D.Linear*
* *Window2D.LogX*
* *Window2D.LogY*
* *Window2D.Log*
* *Window2D.TimeX*
* *Window2D.TimeY*
* *Window2D.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.

### Window2D - Type

#### Linear



#### LogX



#### LogY



#### Log



#### TimeX



#### TimeY



#### Map

*Incubator*

### Window API

In most of cases, developer creates one or more window and registers it a set of plug-ins. For plug-in developer the interesting window methods are projections methods that process projections from user system coordinate to pixel system coordinate and pixel 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.



## 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.

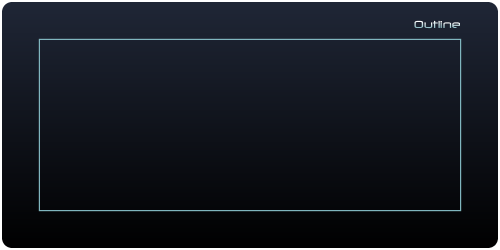
As a user point of view, existing plug-ins include in JenSoft API can be directly use to render such commons charts. Some plug in provides context menu and nice movable widget to execute some command for the host plug in. You can easily extends a context menu or create your own as an interface between 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.

# 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.

## 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.





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.



## Device Image Background

Image can be drawn in the device part component.



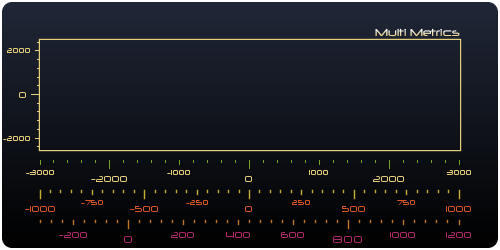
## Text Legend

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.



## 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).



### 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.

### 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.

#### Axis Metrics Plug-ins



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.

#### Device Metrics Plug-ins



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.

#### Metrics Properties

Properties like marker color, label color, label font, axis spacing and metric 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.
* Start Reference
* Factor multiplier
* Major and Median font.
* Marker and label color.
* Range fragment exclusion.

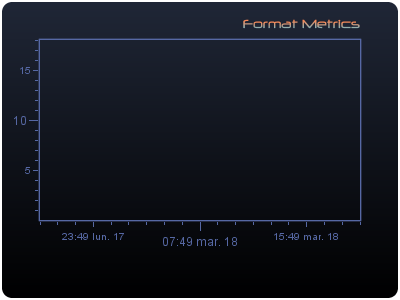
#### Metrics format

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





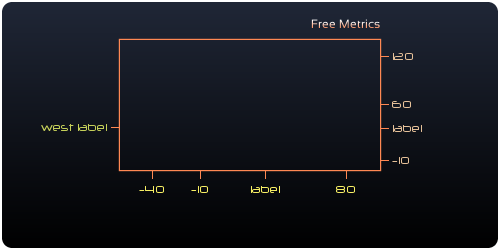
Here is a date format type.



### Metrics samples

#### 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.

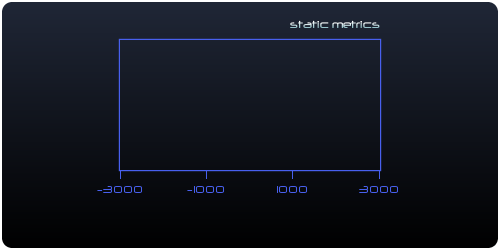






#### Axis Static Metrics

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.





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

#### Axis Dynamic Metrics

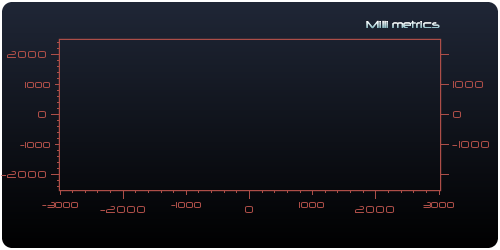
Dynamic metrics creates simple metrics marker for the entire current window user bound regarding the specified reference and metrics multiplier.





#### Axis Milli Metrics

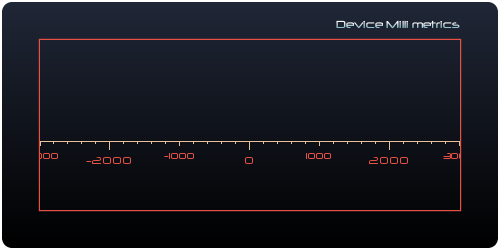
Milli metrics creates three metrics marker types for the entire current window user bound regarding the specified reference, and minor, median and major factors multiplier. Reference gives you a start point entry to obtain homogenous metrics values. The manager solves the maximum count metrics without 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.





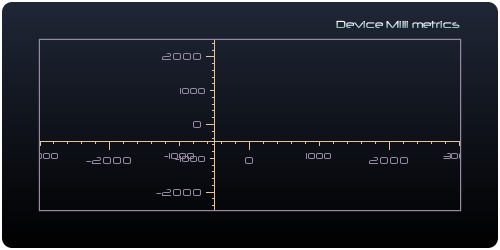
#### Device Milli Metrics

Device metrics is like any other axis in any outer parts but it is paint in the device with a specified value within one of two dimensions x or y. As show on the following view, the horizontal device axis is drawn at a given fixed y position. Behind the scene, it’s the same solvers like axis metrics which are used. You can choose between Dynamic, Milli, Flow, Static, Free manager.





#### Multiple Device Axis

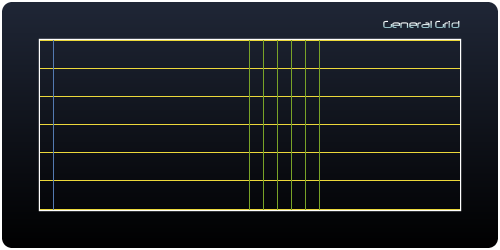






## Grid

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



### 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.

### 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.

 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.

### Grid properties

Some grid property can be set.

* Grid color
* Grid Stroke
* Grid label

### Grid Samples

#### Static grid

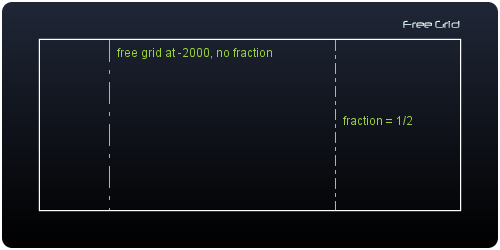






#### Free Grid

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









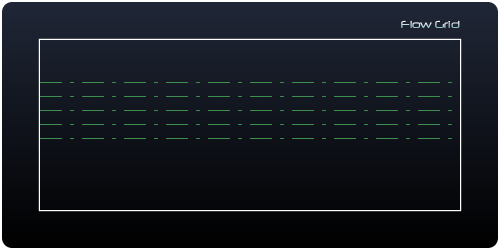






#### Flow grid

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.





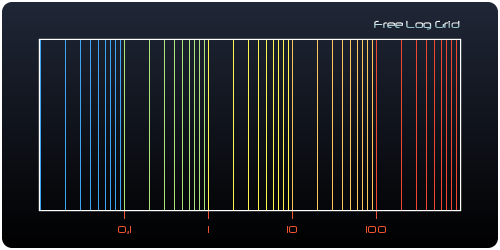


#### Dynamic grid



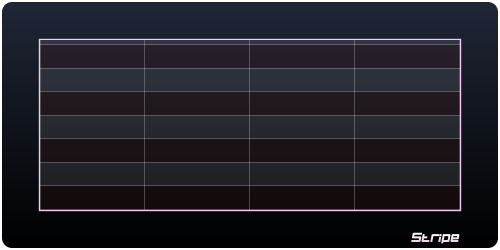


#### Free Grid on Log Window



## Stripe

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



### 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.

### 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.

 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.

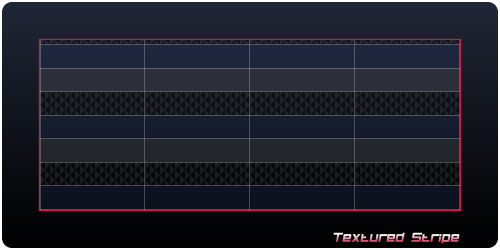
### Stripe Sample

#### Dynamic Stripes





#### Textured Stripes





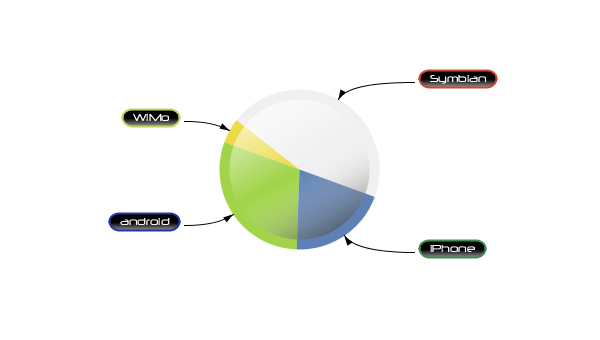
# Data oriented plug-ins

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.

## 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.



### 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)



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.



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



### 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.



### Pie Label

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

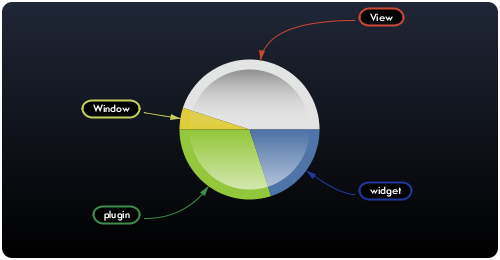
#### Pie Bound Label

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



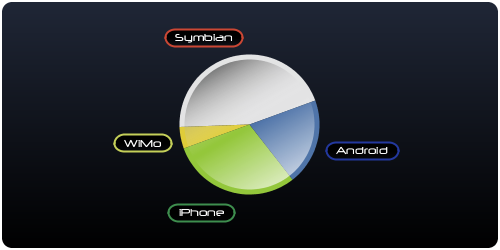


#### Pie Border Label



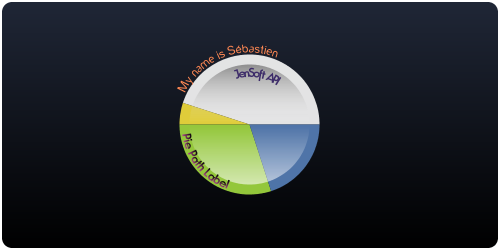


#### Pie Radial Label





#### Pie Path Label





### Pie Effect

#### Pie Linear Effect

#### Pie Radial Effect

### 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.



### Pie Animator

#### Pie Alpha animator

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



#### Pie Divergence Animator

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



#### Pie Flash Animator

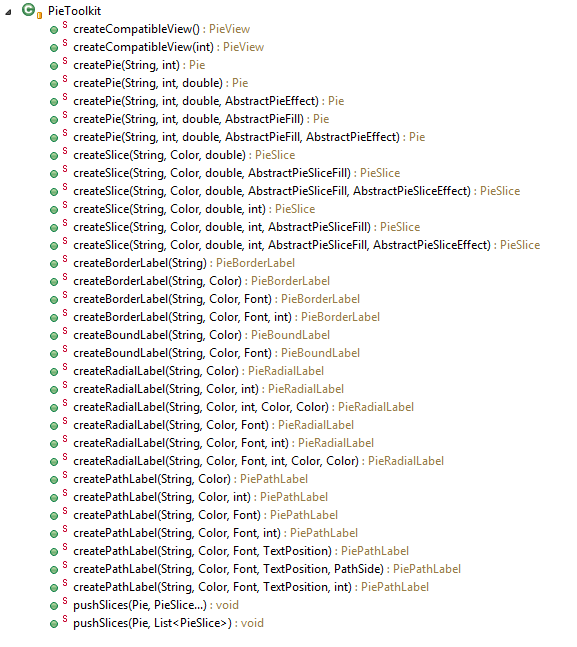
Pie flash run a cycle alpha transition on slice and create a flash effect for the slice.



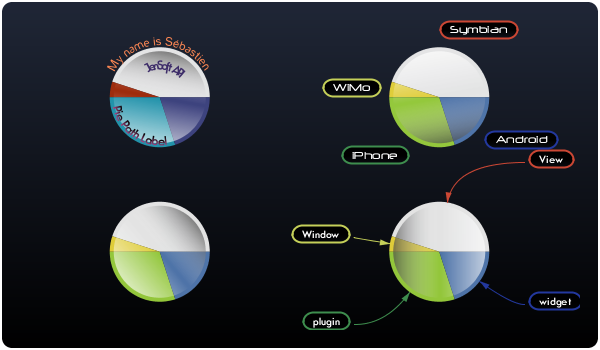
You can set the *ShowFlashBehavior* if you want flash start on a slice rollover or on a slice clicked and the flash velocity with *Flash Speed.*

### 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.



### Multiple Pie

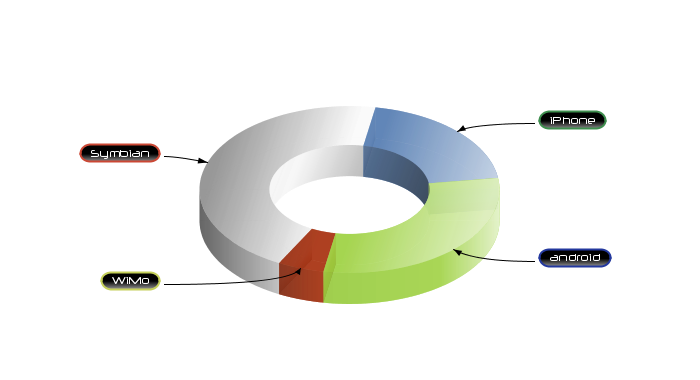




## Pie 3D Chart

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.



### Pie 3D

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.*





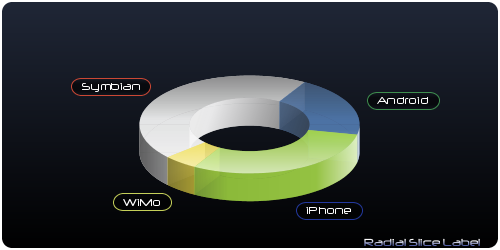
### Pie 3D Slice



### Pie 3D Label

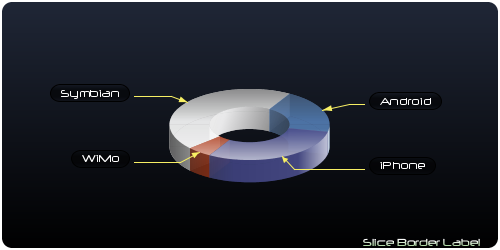
Various kind of pie 3D label helps you to represent data identification and decoration.

#### Pie 3D Radial Label



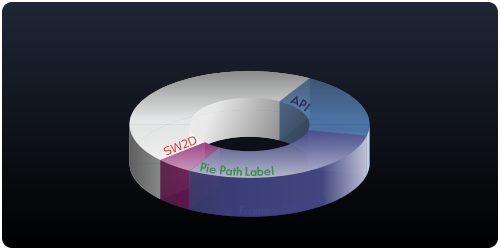


#### Pie 3D Border Label





#### Pie 3D Path Label





### 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.



### 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.



### Pie 3D Animator

#### Pie 3D Divergence Animator

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



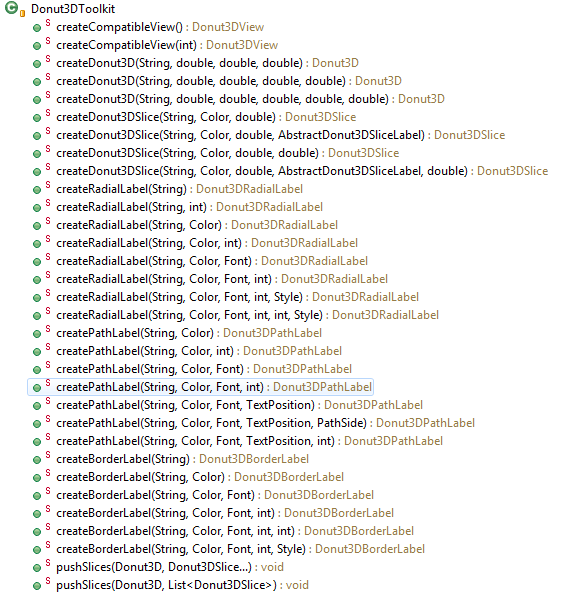
#### Pie 3D Label Animator

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



### 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.



## Donut 2D Chart

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.*

### Donut2D



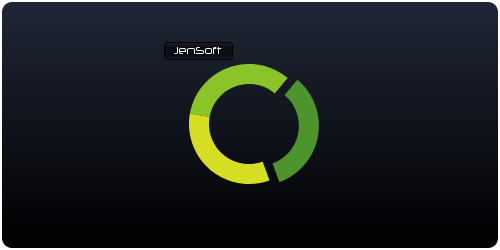


### Donut 2D Slicing



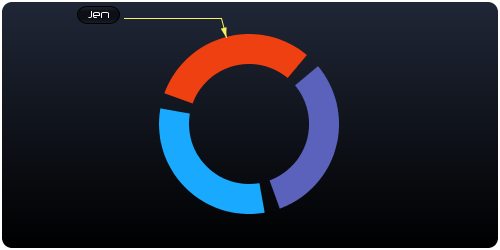
### Donut 2D Label

#### Donut 2D Radial Label



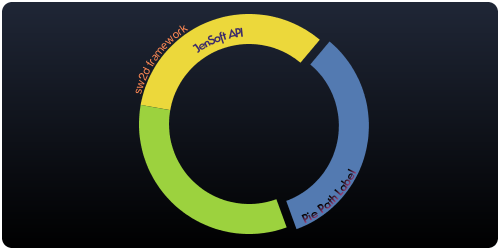


#### Donut 2D Border Label





#### Donut 2D Path Label



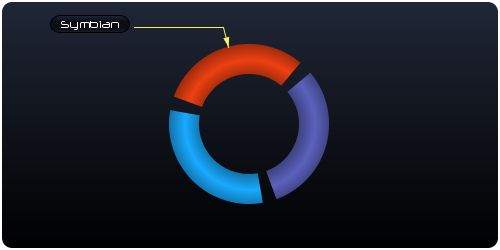


### Donut2D Fill

#### Default Fill

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

#### Radial Fill





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

### Donut 2D Listener

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



### Donut 2DAnimator

#### Donut2D Divergence Animator

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



#### Donut2D Label Animator

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



### 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.



### Multiple Donut 2D

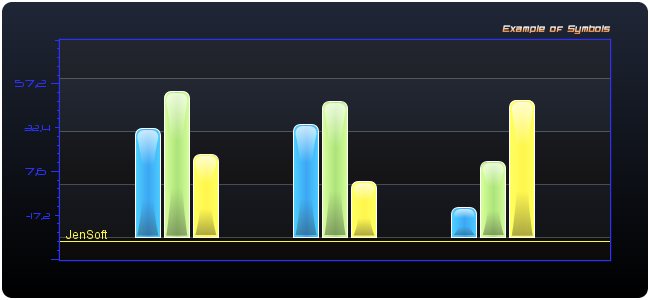
Multiple donut 2D can be achieve. Outer radius for the inner donut should be lesser than the inner radius for the outer donut2D. The building donut2D strategy is the same than for one donut.



## Symbols



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.

### 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.



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).



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



### 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.

#### 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.



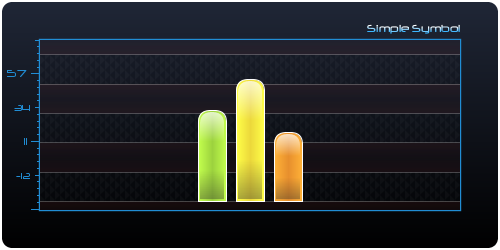
Glue and Strut are non visible symbol which contributes to the lay out. It’s a very simplest way to lay out symbol like you needs. Layer lays out on the entire device all of symbols children as a flow of symbol which has fixed size in pixel. It’s the symbolic dimension.

#### 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)



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.

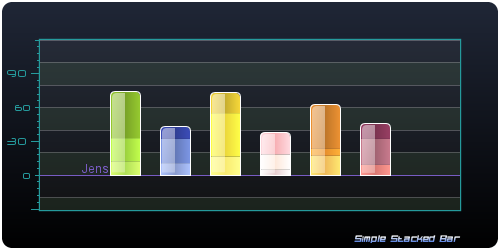


#### Stacked Bar Symbol

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.



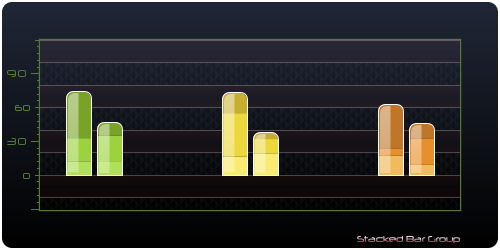
*StackedBarSymbol* is also a symbol which inherits from *BarSymbol. Stack* which is also a symbol is added as child to the stacked symbol. You have to create stacked symbol which commons properties like thickness, base value, fill, draw, effect, etc. and a ascent or descent value. Stacks are part of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



#### Bar Symbol Group

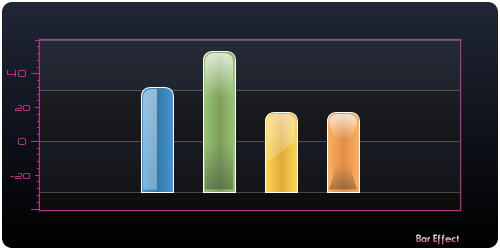
A bar symbol group *BarSymbolGroup* allows to lay out some children symbols with common property set without to have register property for each registered child symbol. Groups are quite different from layer because glue cannot be used in group. Only strut filler and symbol are allowed. Moreover group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



#### Bar Symbol Effect

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





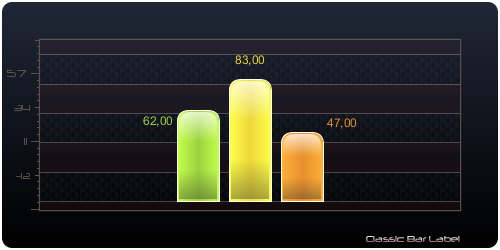


#### Bar Symbol Label

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

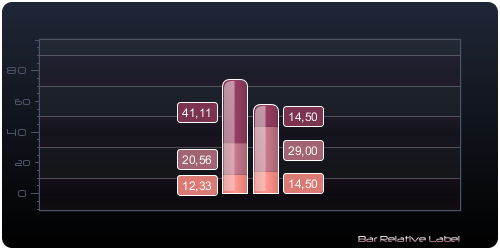
The default label draws the label at the top of the bar. Label can be shifted by using offset x and y properties. Here an example with 3 default label with shifting.



##### 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.

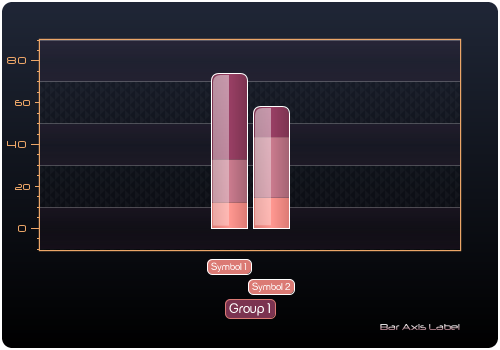
* *VerticalAlignment*
* *HorizontalAlignment*





##### Axis bar Label

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





#### 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.



### 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.

#### Point Layer

The point layer is a layer that hosts point symbol.



#### 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.



#### 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.



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.

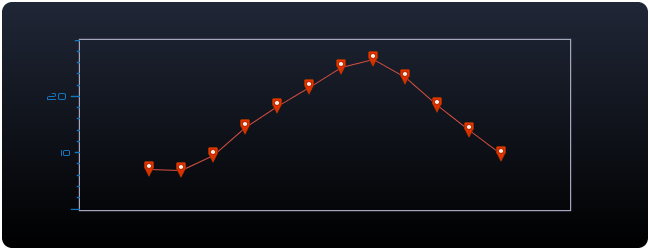


#### Polyline Symbol

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.



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.







#### 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.



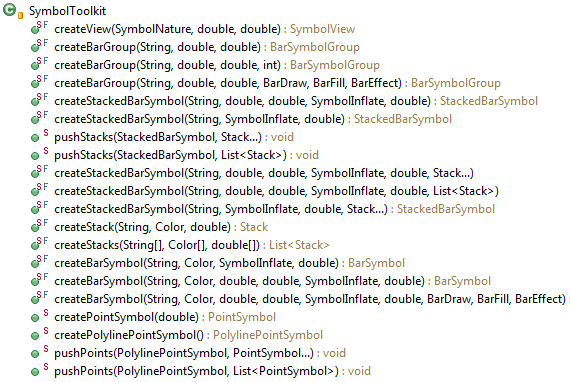
#### Point Listener

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.



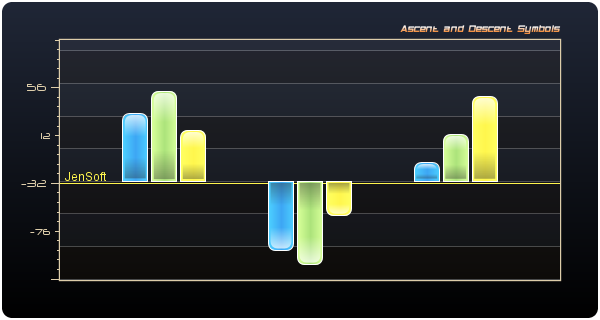
### Symbol toolkit

Here is the symbol toolkit methods outline.



### Sample Symbols

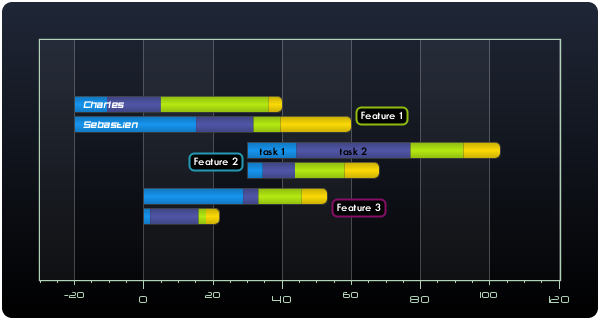
#### Vertical Ascent and Descent Bar Symbol





#### 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.

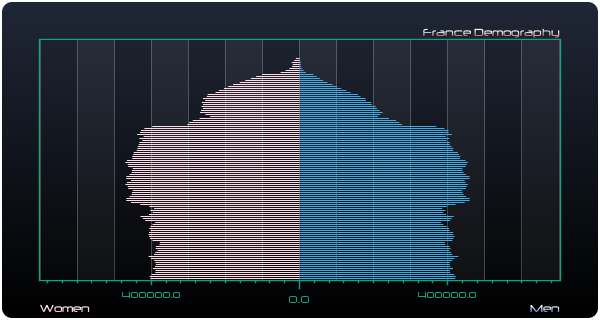


Here is naïve implementation of a Gantt chart which is shown. Symbol represent “feature” for different people. Each feature is divided into task. The axis should be symbolized by time or points.

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.

#### 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.









#### 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:



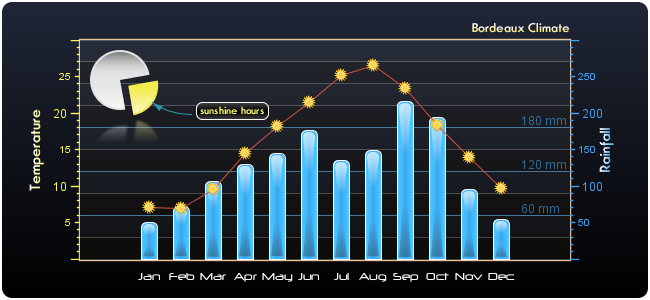






#### Climate Bar Symbol and point curve symbols

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.































## 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)

### Ray Plug-in

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



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.



### 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)*.*



### 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.



### Ray Group

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



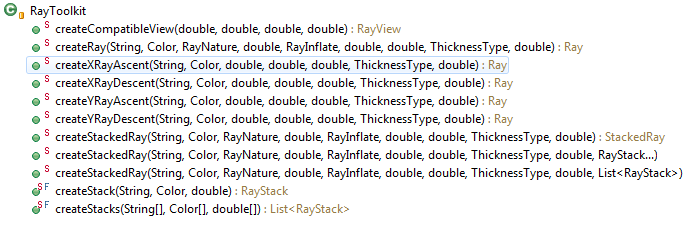
### Ray Listener

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



### Ray Toolkit

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



### Ray versus Bar Symbol

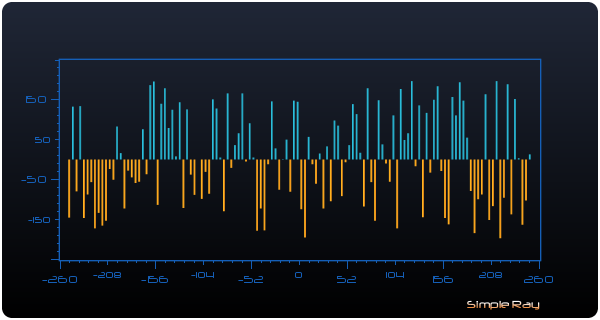
Sometimes charts can be made with symbol plug-in or ray plug-in as soon as you should have bar combo and sometimes you do not. For example for a scan component where you have to show some detected power channels for a specified frequency, you have to use ray plug-in but age people distribution chart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

### Ray Samples

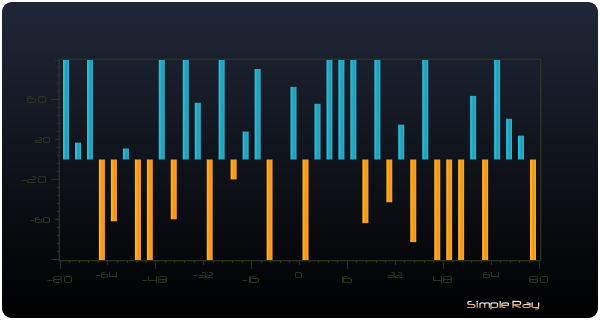
#### Simple Ray

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



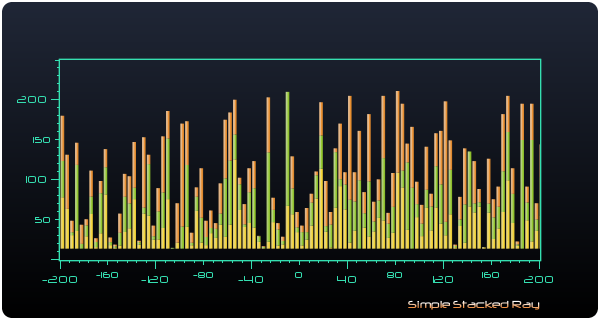


The same ray graph with a window lower bound (achieve by zoom plug-in).



#### Stacked Ray

The same type of randomized ray can be made.







## 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.

### 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](http://en.wikipedia.org/wiki/Linear_predictor_function).
* *InterpolateSerie2D* - Interpolate serie defines a set of point which is fitting with spline curve interpolator.



#### Line Serie

Create simple serie with *SerieToolkit* is quite simple.



#### Interpolate Serie



#### Linear regression serie



### 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.



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.

### Plot Tool

#### 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.



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



#### Peak Tracker

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

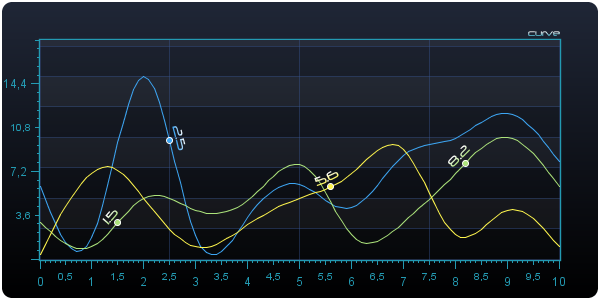
## Curve



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.

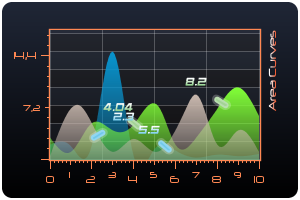
The figure shows the case of a multiple curves chart in the same window projection and some label annotations. The metrics glyph corresponds to the x projection of a specified glyph metrics. The label is default set to the y value and can be choose by user.



## Area

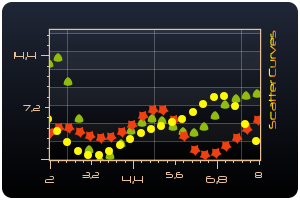
An area chart or area graph displays graphically quantitive data. It is based on the line chart. The area between axis and line are commonly emphasized with colors, textures and hatchings. Commonly one compares with an area chart two or more quantities. Area charts are used to represent cumulated totals 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.



## 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.



## Radar

### Radar Plug-in

### Radar Dimension

### Surface

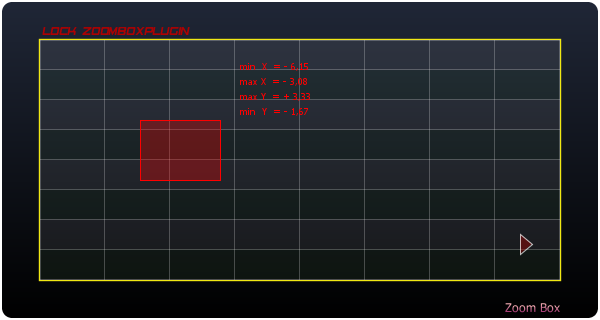
# Tool oriented plug-ins

## Zoom Box

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

### Zoom Box Frame Box

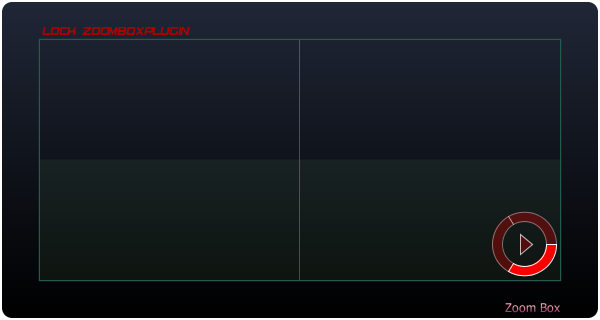
*ZoomBoxPlugin* is a selectable plug-in that need a lock to allow zooming operation. The lock can be lock with *lockSelected* method or with the menu item of the plug in device context menu. When the plug in is selected, a view area can be zoomed. On left click on the view and drag cursor, you can create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



### Zoom Box Widget

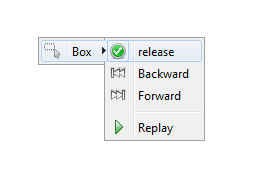
The zoom box device widget is a donut morphe widget. Each slice represents a zoom bounding box which can be shown on a slice clicked. You can play the zoom sequence with the center button.





### 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.

### Zoom Synchronizer

Zoom synchronizer provides zoom box synchronization between view.



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

## Zoom Objectif

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



### 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.

#### Bar Widgets





#### Pad Widget





### Objectif device context

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



## Zoom Wheel

Zoom wheel provides a zoom when mouse wheel occurs.



You can create zoom wheel synchronizer for different views.



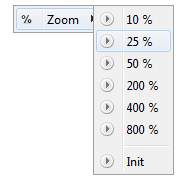
## Zoom Percent

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



### Zoom Percent device context





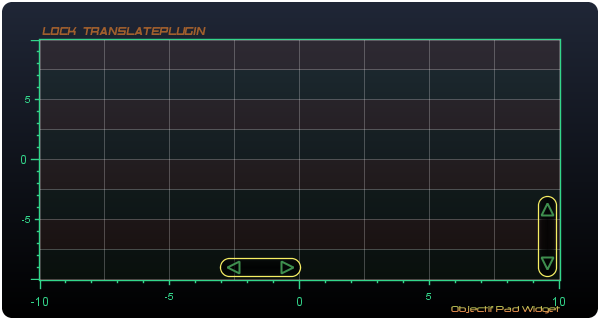
## Translate

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.

### 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.

#### Bars Widgets





#### Pad Widget

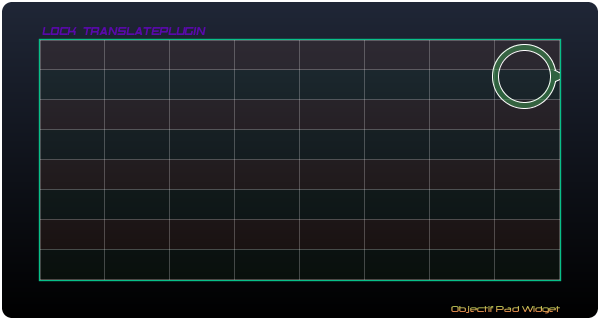




#### Compass Widget

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

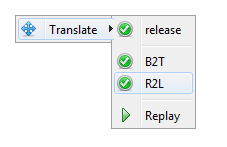




### Translate device context

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.





## Marker

## Serie Tracker

## Peack Tracker

# Instrument oriented Plug-ins

## Gauge

## General Metrics Path

# X2D

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

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

# JET

# Framework Development

## 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.

## Widget

## Device Context Menu