GraphEngineDev

Introduction

The GraphEngine is a generic chart rendering library. The primary use will be to generate charts for TikiSheet, but usage could be extended to other features in Tiki and other projects. The library should be working and integrated to TikiSheet by the 1.9 release. This page indicates the development status, decisions and various specification rules. It's written for those using the GraphEngine library (not the user interfaces that will use it) or extending it.

Integration made in time for 1.9! Enable feature_charts.

The library aims to be output independant. At the moment, multiple renderers are available:

- GD (PNG, JPEG)
- PDF (Using PDFLib)
- PostScript (Using PSLib, in PECL)

Other renderers such as SVG and PDF using other libraries can be added without any major changes. Those renderers are planned in a post-1.9 future. To ensure the independance of the renderers, all renderers must extent the GRenderer class and follow the base guidelines and follow the expected behaviors.

GRenderer Specifications

- General Rules
  - Renderer clients only rely on the base interface defined by GRenderer.
  - Methods starting with an underscore are used internally and are not part of the interface.
  - Lengths and positions are given as float values between 0 and 1. The values are proportions of the width or height.
  - Output parameters are given using the style parameters.
  - Coordinates begin from the top-left corner as (0,0)
- Style parameters
  - Renderer-specific.
  - Clients should not make any assumptions concerning the structure.
  - Implementers choose the structure and can modify it at any time (as long as everything still works).
  - Are created by the getStyle($name) method. The names are common to all renderers. Implementers are responsible of making sure the styles returned behave properly.
- Text
  - Location is based on left, right and height.
  - Height is the top of the string.
  - In the case of vertical text: height becomes left, left becomes bottom and right becomes top.
  - Alignment and orientation is handled by styles.
  - getTextWidth() and getTextHeight() return values between 0 and 1 to make sure the clients can perform appropriate calculations.
- Shapes
  - Lines use the LineStroke styles.
  - Rectangles can use the LineStroke and FillStroke styles.
  - Pies can use the LineStroke and FillStroke styles.
  - Angles in pies start at 0 RAD, (1,0), or whatever you call that location at the right of the circle.
  - Angles are counter-clockwise.
  - Angles are given in degrees.
  - Radius is a length and will be transformed against the smallest of available width or height. (consider a landscape sheet of paper, height would be used)

The _getRawColor() function recieves a color name as the parameter and returns an array containing the color codes as RGB. The color names are all lower case and the returned array contains three values associated to 'r', 'g' and 'b'. The values are 0-255. Available colors are:

- red
- green
- blue
- yellow
- orange
- lightgreen
- lightblue
- gray
- black
- white

Graphics

Feeding Data

Data is feed using series: single dimension arrays (keys will be ignored). Each graphic has to identify which series it uses. When called, setData() will verify if all required series are present and make sure they are valid. The series list is taken from getRequiredSeries(), which returns an associative array with the series name as the key and a boolean value to indicate if's mandatory. A non-mandatory series means the graphic will come up with replacement values.

setData() recieves an associative array as the only parameter. The key is the name of the series and the value is an array containing the values. If the data is valid, _handleData() will be called so the graphic can handle the values and build proper internal structures.

Some graphics might require unlimited amounts of series, such as a line graphic having n amount of lines. It's recommanded to create a single entry in getRequiredSeries() called 'y0'. setData() will make sure all the series listed are present, but will accept non-listed series as long as they are valid arrays. Examples of this will be created soon.

Currently Available Graphics

- PieChartGraphic
- MultilineGraphic
- BarStackGraphic
- MultibarGraphic

More can be added, name it.

Parameters

The output of graphics can be customized using parameters. All parameters have default values that can be overloaded. The setParam() method allows to do this operation.

About Fake_GRenderer

The Fake_GRenderer class is a decorator class over any kind of renderer. It allows to create a renderer over a portion of the other renderer. The fake renderer is used internally to force methods to draw in specific portions of the graphic without sending 4 position parameters.

For example, the _drawContent() abstract method recieves a renderer as the first argument. The draw() method initialize in Fake_GRenderer to the content area of the graphic after determining the size when title and legend have been positionned. This way, the child classes are totally independant and render their content in a 0 to 1 area, no matter what the real position is.

The renderer could be used by clients to fit multiple different graphics in a same canevas without having to modify any of the graphics being rendered.

The fake renderer forwards the method calls to the real renderer after scaling the positions and adding the offsets. All components are scaled down except the text. Since in all graphic libraries used the fonts have a static size, the size of the font cannot be scaled down. To keep consistency, opposite operations are applied in the calls to getTextWidth() and getTextHeight(). (You dont have to understand this, it magically works)

Style List

TODO

Layout Parameters

TODO