Guile Charting

version 0.2.0, updated 19 September 2014

Andy Wingo (wingo@pobox.com)

This manual is for Guile Charting (version 0.2.0, updated 19 September 2014)

Short Contents

1 (charting)

1.1 Overview

1.2 Usage

`make-bar-chart` [*title*] [*data*] [*write-to-png*] [*bar-width*] [*group-spacing*] [*chart-height*] [*max-y*] [*chart-params*] [*legend-params*] [*ytick-label-formatter*] [*bar-value-formatter*] [Function]

Make a bar chart.

The format of *data* is defined as follows:

data (*group*+)

group (*group-label bar*+)

group-label
A string, to be written to the X axis.

bar (*height bar-params*?)

height The bar height, as a number.

bar-params
A property list suitable to passing to [charting draw draw-bar], page 5.

This function returns the cairo surface. By default, make-chart will create an image surface, but you may override this by passing a `#:make-surface` function in the *chart-params*. In this way you can render charts to any surface supported by Cairo, e.g. PS, PDF, SVG, GDK, etc.

The #:write-to-png option will write the chart out to the PNG file that you name.

An example invocation might look like:

```
(make-bar-chart "Average Height at Iihenda JSS"
                '(("Grade 9" (150 "Boys") (140 "Girls"))
                  ("Grade 10" (150 "Boys")
                              (140 "Girls" (#:y+-bracket 5 #:y--bracket 4.5))))
                #:write-to-png "/tmp/graph.png")
```

`make-chart` [*title*] [*chart-height*] [*chart-width*] [*font-family*] [*line-width*] [*title-text-height*] [*axis-text-height*] [*x-axis-label*] [*y-axis-label*] [*tick-size*] [*y-axis-ticks*] [*x-axis-ticks*] [*y-axis-tick-labels*] [*x-axis-tick-labels*] [*x-axis-tick-mode*] [*y-axis-tick-mode*] [*chart-margin*] [*margin*] [*padding-left*] [*padding-right*] [*padding-top*] [*padding-bottom*] [*make-surface*] [Function]

Make a chart.

tick-lables is an alist of label-value pairs, where the value is given in chart height coordinates. The label can be #f.

This function makes the basic chart, setting up the basics like the title, axes, etc. You probably don't want to call this unless you are making a custom chart type.

This function returns a cairo context whose coordinate system has been flipped so that the origin of the chart is (0, 0), with positive in the northeast quadrant.

`make-page-map` [*title*] [*data*] [*write-to-png*] [*margin*] [*page-size*] [Function]
[*page-width*] [*page-height*] [*page-spacing*] [*title-text-height*] [*text-height*]
[*label-bar-spacing*] [*font-family*]

Make a page map.

A page map shows the components of a one-dimensional space. Each component has a label, a start, and a size. The result is a graphical representation of the space, divided in @@var@{page-size@} strips, along with a summary list of the different components.

The format of @@var@{data@} is as follows:

@@example
((@@var@{label@} . (@@var@{start@} . @@var@{size@})) ...)@}
@@end example

@@var@{label@} should be a string. @@var@{start@} and @@var@{size@} should be numbers.

The #:write-to-png option will write the chart out to the PNG file that you name.

An example invocation might look like:

@@example
(make-page-map
 "foo.so"
 '((".text" 1024 65535)
   (".data" 65536 20)
   (".rodata" 65556 200))
 #:write-to-png "foo.png")
@@end example

`make-performance-chart` [*title*] [*data*] [*write-to-png*] [*box-width*] [Function]
[*box-spacing*] [*test-spacing*] [*chart-height*] [*max-y*] [*min-y*] [*chart-params*]
[*legend-params*] [*ytick-label-formatter*] [*box-value-formatter*]

Make a performance chart.

A performance chart compares runtimes for some set of tests across some set of scenarios.

The format of *data* is defined as follows:

```
((scenario (test data-point ...) ...) ...)
```

scenario and *test* should be strings. *data-point* should be numbers.

The resulting plot will have time on the Y axis, and one X axis entry for each test. Each test/scenario data set will be represented as a box plot. In the future we should add more options (for example, a small vertical histogram on the plot).

This function returns the cairo surface. By default, make-chart will create an image surface, but you may override this by passing a `#:make-surface` function in the *chart-params*. In this way you can render charts to any surface supported by Cairo, e.g. PS, PDF, SVG, GDK, etc.

The #:write-to-png option will write the chart out to the PNG file that you name.

An example invocation might look like:

```
(make-performance-chart
 "Gabriel Benchmarks"
 '(("guile-1.8"
    ("tak" 0.12 0.13 0.17)
    ("fib" 1.13 1.24 1.05))
   ("guile-2.0"
    ("tak" 0.05 0.051 0.047)
    ("fib" 0.64 0.59 0.71)))
 #:write-to-png "/tmp/graph.png")
```

`make-performance-series` [*title*] [*data*] [*write-to-png*] [*box-width*] [*box-spacing*] [*test-spacing*] [*chart-height*] [*max-y*] [*min-y*] [*chart-params*] [*annotations*] [*ytick-label-formatter*] [*box-value-formatter*] [Function]

Make a performance chart.

A performance chart compares runtimes for some set of tests across some set of scenarios.

The format of *data* is defined as follows:

```
((x data-point ...) ...)
```

x and *data-point* should be numbers.

The resulting plot will have time on the Y axis, and one X axis entry for each test. Each data set will be represented as a box plot. In the future we should add more options (for example, a small vertical histogram on the plot).

This function returns the cairo surface. By default, make-chart will create an image surface, but you may override this by passing a `#:make-surface` function in the *chart-params*. In this way you can render charts to any surface supported by Cairo, e.g. PS, PDF, SVG, GDK, etc.

The #:write-to-png option will write the chart out to the PNG file that you name.

An example invocation might look like:

```
(make-performance-chart
 "Gabriel Benchmarks"
 '(("guile-1.8"
    ("tak" 0.12 0.13 0.17)
    ("fib" 1.13 1.24 1.05))
   ("guile-2.0"
    ("tak" 0.05 0.051 0.047)
    ("fib" 0.64 0.59 0.71)))
 #:write-to-png "/tmp/graph.png")
```

`make-scatter-plot` [*title*] [*data*] [*write-to-png*] [*test-spacing*] [Function]
[*chart-height*] [*min-x*] [*max-x*] [*min-y*] [*max-y*] [*log-x-base*] [*log-y-base*]
[*chart-params*] [*legend-params*] [*x-axis-label*] [*y-axis-label*] [*tick-label-formatter*]

Make a scatter plot.

A scatter plot shows a number of series as individual points.

The format of *data* is defined as follows:

```
((series (x . y) ...) ...)
```

series should be a string. *x* and *y* should be numbers.

This function returns the cairo surface. By default, make-chart will create an image surface, but you may override this by passing a `#:make-surface` function in the *chart-params*. In this way you can render charts to any surface supported by Cairo, e.g. PS, PDF, SVG, GDK, etc.

The #:write-to-png option will write the chart out to the PNG file that you name.

An example invocation might look like:

```
(make-scatter-plot
 "MPG for cars"
 '(("ford" (1 . 2) (2 . 3))
   ("opel" (1.2 . 3.5) (4.5 . 1)))
 #:write-to-png "/tmp/graph.png")
```

2 (charting draw)

2.1 Overview

2.2 Usage

`draw-annotations` [*cr*] [*annotations*] [*xticks*] [*width*] [*height*] [Function]

`draw-axis-label` [*cr*] [*text*] [*text-height*] [*axis-length*] [*vertical?*] [Function]
Draw an axis label.

The label will be drawn such that the current position of *cr* is the closest corner of the label's bounding box.

`draw-background` [*cr*] [Function]
Draw the background.

`draw-bar` [*cr*] [*height*] [*scale*] [*bar-width*] [*bar-value-formatter*] [*series*] [*decorator*] [Function]
Draw a single bar.

cr is expected to have been placed at the lower left corner of where the bar should be. *decorator* is a property list that can be passed to [charting draw draw-decorator], page 6.

`draw-bar-group` [*cr*] [*data*] [*bar-width*] [*scale*] [*bar-value-formatter*] [Function]
Draw a group of bars.

data is a property list suitable for passing to [charting draw draw-bar], page 5. *cr* is expected to have been positioned along the x axis in the center of where the bar group should be displayed.

`draw-bar-legend` [*cr*] [*data*] [*width*] [*text-height*] [*font-family*] [*horizontal-spacing*] [*vertical-spacing*] [Function]
Draw a "bar legend".

A bar legend is meant to show what categories exist, as well as indicating their contribution to a graph. Use a bar legend if it would be confusing to label some other chart in which the pixel count of a category is proportional to its magnitude, but you want to make sure to label all categories, even those with small magnitudes.

data is as in [charting draw draw-page-map], page 6. The legend will be written below the current position of *cr*.

`draw-chart-area` [*cr*] [*width*] [*height*] [Function]
Draw the actual box for the chart background.

cr is expected to have been positioned at the origin.

`draw-decorator` [*cr*] [*scale*] [*label*] [*y+-bracket*] [*y–bracket*] [*y-bracket*] [Function]
Draw a decorator.

A decorator is something drawn around a point, such as error bars. This function currently supports drawing error bars in the Y direction, which are specified individually as *y+-bracket* and *y–bracket*.

`draw-grid` [*cr*] [*ticks*] [*width*] [*vertical?*] [Function]
Draw grid lines.

ticks is a list of positions in the current cairo coordinate system. *width* is the that the grid lines should be: the chart width of *vertical?*, and the height otherwise.

`draw-legend` [*cr*] [*expand-right?*] [*expand-down?*] [*measure-only?*] [Function]
[*text-height*] [*draw-outlines?*] [*draw-background?*] [*text-measurer*] [*series-list*]
Draw a legend.

series-list is expected to be a list of series names. The *cr* is expected to be positioned at one of the corners of the legend; *expand-right?* and *expand-down?* control which way the legend will be rendered.

`draw-page-map` [*cr*] [*data*] [*chart-width*] [*chart-height*] [*page-size*] [Function]
[*page-height*] [*page-spacing*]
Draw a page map for the given data set.

data := `(label . (start . size)) ...)`

label is a string, and *start* and *size* are numbers. *cr* is expected to have been positioned at the lower-left corner of the chart area.

`draw-perf-series` [*cr*] [*data*] [*xticks*] [*box-width*] [*box-spacing*] [*min-y*] [Function]
[*y-scale*] [*box-value-formatter*]
Draw a group of boxes corresponding to runs of one benchmark in different scenarios.

data := `(x point ...) ...)`

, where the series is a string, and the points are numbers. *cr* is expected to have been positioned along the x axis in the center of where the data for the test should be displayed.

`draw-perf-test` [*cr*] [*data*] [*box-width*] [*box-spacing*] [*min-y*] [*y-scale*] [Function]
[*box-value-formatter*]
Draw a group of boxes corresponding to runs of one benchmark in different scenarios.

Each scenario corresponds to a series. The format of *data* is `((series point ...) ...)`, where the series is a string, and the points are numbers. *cr* is expected to have been positioned along the x axis in the center of where the data for the test should be displayed.

`draw-point` [*cr*] [*x*] [*y*] [*label*] [Function]
Draw a point at the current position.

`draw-tick-labels` [*cr*] [*tick-labels*] [*tick-size*] [*vertical?*] [*text-height*] [Function]
Draw tick labels on an axis.

tick-labels is an alist of label-position pairs, where position is in the current cairo coordinate system, along one axis.

`draw-ticks` [*cr*] [*ticks*] [*tick-size*] [*vertical?*] [Function]

Draw ticks on an axis.

ticks is a list of positions in the current cairo coordinate system.

`draw-title` [*cr*] [*text*] [*font-size*] [Function]

Draw a title.

cr is expected to have been positioned at the lower boundary of where the title should be written, in the center.

`reset-colors!` [Function]

Concept Index

(Index is nonexistent)

Function Index

D

M

R