Documentation for Perl Package **Chart**

Version 2.4.4

Chart Group[1]

Last change: 2012-01-06

[1]Bundesamt für Kartographie und Geodäsie, Geodätisches Observatorium Wettzell, Sackenrieder Strasse 25, D-93444 Bad Kötzting, E-mail: chart@fs.wettzell.de

Contents

List of Figures

1 Description

Synopsis

```
use Chart::type;     (type is one of: Bars, Composite,
Direction, ErrorBars, HorizontalBars, Lines, LinesPoints,
Mountain, Pareto, Pie, Points, Split or StackedBars)

$obj = Chart::type->new();
$obj = Chart::type->new(\$width, \$height);

$obj->set( $key_1,   $val_1, ... , $key_n,   $val_n);
$obj->set( $key_1 => $val_1, ... , $key_n => $val_n);
$obj->set( %hash );

# Graph.pm-style API to produce PNG formatted charts:
@data = ( \@x_tick_labels, \@dataset_1, ... , \@dataset_n);
$obj->png( "filename", \@data );
$obj->png( $filehandle, \@data );
$obj->png( FILEHANDLE, \@data );
$obj->cgi_png();

# Graph.pm-style API:
$obj->add_pt($label, $val_1, ..., $val_n);
$obj->add_dataset($val_1, ..., $val_n);
$obj->png("filename");
$obj->png($filehandle);
$obj->png(FILEHANDLE);
$obj->cgi_png();
# Similar functions are available for JPEG output.

# Retrieve imagemap information:
$obj->set('imagemap' => 'true');
$imagemap_ref = $obj->imagemap_dump();
```

The Perl module Chart creates PNG or JPEG output which can be written to a file or to stdout. Therefore, Chart can also create dynamic charts for web sites.

Many different chart types are available, viz., Bars, Composite, Direction, ErrorBars, HorizontalBars, Lines, LinesPoints, Mountain, Pareto, Pie, Points, Split, and StackedBars. Each specific type is implemented as a class by itself which is derived from the same abstract superclass, Base.

The hierarchy of Chart classes is shown in Figure 1.

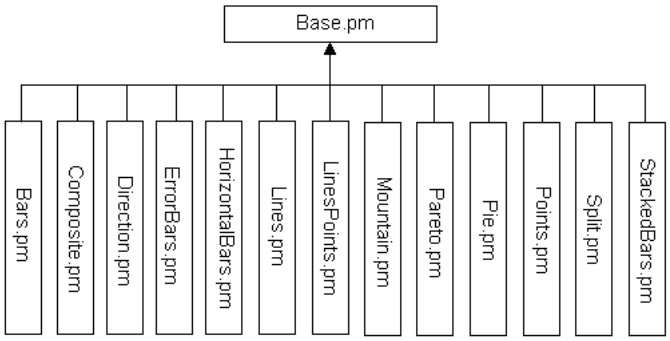

Figure 1: The hierarchy of Chart classes

You must create an *instance of one of the concrete subclasses* to get a Chart object. Take a look at the individual class descriptions to see how they work.

All the methods and most of the options Chart provides are implemented in the Chart::Base class. However, drawing of the graph itself happens in the appropriate subclass. Figure 2 shows the elements of a chart from a layout perspective.

The graph area in the middle is drawn by the subclass, all the other elements are drawn by Chart::Base. But some classes do not need all of those elements, or they may need additional elements. The Chart::Base methods producing these elements have then to be overwritten in the respective subclass. For example, class Chart::Pie needs no axes, so the methods for drawing these in file Base.pm are overwritten by methods in class Chart::Pie; in this case, no axes are drawn. Furthermore, the legend in a pie chart is slightly different. Therefore, Pie.pm has its own methods for drawing the legends. All these rules are managed by Chart, so you do not have to attend to it.

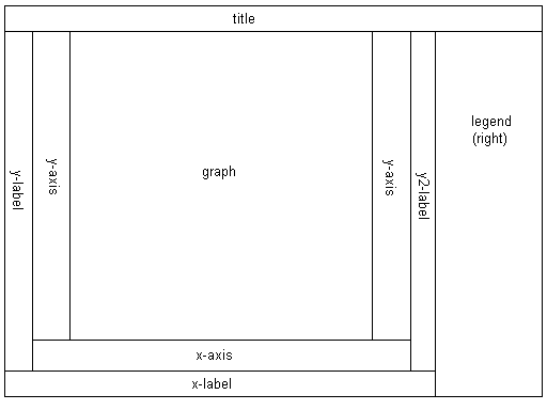

Figure 2: Layout Elements of a chart

Chart uses Lincoln Stein's GD module for all its graphics primitives calls. So you need an installed version of GD.pm to use Chart. This module is available in the CPAN online archive at `http://www.cpan.org/`, just like Chart itself.

The table lists all attributes that are currently used within the Chart package. It shows which of the concrete subclasses uses each attribute.

Attribute	Bars	Composite	Direction	ErrorBars	HorizontalBars	Lines	LinesPoints	Mountain	Pareto	Pie	Points	Split	StackedBars
angle_interval			X										
arrow			X										
brush_size			X	X		X	X						
brush_size1		X											
brush_size2		X											
brushStyle							X				X		
brushStyle1		X											
brushStyle2		X											
colors	X	X	X	X	X	X	X	X	X	X	X	X	X
composite_info		X											
custom_x_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
f_x_tick	X	X	X	X	X	X	X	X	X	X	X	X	X
f_y_tick	X	X	X	X	X	X	X	X	X	X	X	X	X
f_y_tick1		X											
f_y_tick2		X											
graph_border	X	X	X	X	X	X	X	X	X	X	X	X	X
grey_background	X	X	X	X	X	X	X	X	X	X	X	X	X
grid_lines	X	X	X	X	X	X	X	X	X	X	X	X	X
imagemap	X	X	X	X	X	X	X	X	X	X	X	X	X
include_zero	X	X	X	X	X	X	X	X	X	X	X	X	X
integer_ticks_only	X	X	X	X	X	X	X	X	X	X	X	X	X
interval												X	
interval_ticks												X	
label_font	X	X	X	X	X	X	X	X	X	X	X	X	X
label_values										X			
legend	X	X	X	X	X	X	X	X	X	X	X	X	X
legend_example_height		X											
legend_example_size	X	X	X	X	X	X	X	X	X	X	X	X	X
legend_font	X	X	X	X	X	X	X	X	X	X	X	X	X
legend_label_values										X			
legend_labels	X	X	X	X	X	X	X	X	X	X	X	X	X
legend_lines										X			
line			X										
max_circles			X										
max_val	X	X	X	X	X	X	X	X	X	X	X	X	X
max_val1		X											
max_val2		X											
max_x_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
max_y_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
min_circles			X										
min_val	X	X	X	X	X	X	X	X	X	X	X	X	X
min_val1		X											
min_val2		X											
min_x_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
min_y_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
no_cache	X	X	X	X	X	X	X	X	X	X	X	X	X
pairs			X										
png_border	X	X	X	X	X	X	X	X	X	X	X	X	X
point			X										
precision	X	X	X	X	X	X	X	X	X	X	X	X	X
pt_size			X	X			X				X		
ring										X			
same_error				X									
same_y_axes		X											
scale												X	
skip_int_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
skip_x_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
skip_y_ticks					X								
sort			X	X		X	X		X		X	X	
spaced_bars	X				X				X				X
start												X	
stepline						X	X						
stepline_mode						X	X						
sub_title	X	X	X	X	X	X	X	X	X	X	X	X	X
text_space	X	X	X	X	X	X	X	X	X	X	X	X	X
tick_label_font	X	X	X	X	X	X	X	X	X	X	X	X	X
tick_len	X	X	X	X	X	X	X	X	X	X	X	X	X
title	X	X	X	X	X	X	X	X	X	X	X	X	X
title_font	X	X	X	X	X	X	X	X	X	X	X	X	X
transparent	X	X	X	X	X	X	X	X	X	X	X	X	X
x_grid_lines	X	X	X	X	X	X	X	X	X	X	X	X	X
x_label	X	X	X	X	X	X	X	X	X	X	X	X	X
x_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
xlabels				X		X	X				X		
xrange				X		X	X				X		
xy_plot				X		X	X				X		
y_axes	X			X	X		X	X	X		X	X	X
y_grid_lines	X	X	X	X	X	X	X	X	X	X	X	X	X
y_label	X	X	X	X	X	X	X	X	X	X	X	X	X
y_label2	X	X	X	X	X	X	X	X	X	X	X	X	X
y_ticks	X	X	X	X	X	X	X	X	X	X	X	X	X
y_ticks1		X											
y_ticks2		X											
ylabel2	X	X	X	X	X	X	X	X	X	X	X	X	X

2 Chart::Base

Name: Chart::Base

File: Base.pm

Requires: GD, Carp, FileHandle

Description:

Chart::Base is the abstract superclass of classes Chart::Bars, Chart::Composite, Chart::Direction, Chart::ErrorBars, Chart::HorizontalBars, Chart::Lines, Chart::LinesPoints, Chart::Mountain, Chart::Pareto, Chart::Pie, Chart::Points, Chart::Split, and Chart::StackedBars.

Class Chart::Base provides all public methods and most of the attributes of Chart objects.

Constructor:

An object instance of class Chart can be created with the constructor `new()`:

```
$obj = Chart::Type→new();
$obj = Chart::Type→new(width, height);
```

Type here denotes the type of chart that is to be returned, e. g., `Chart::Bars→new()` returns a bar chart.

If `new()` is called without arguments, the constructor will return an object of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart object of the desired size.

Methods:

`$obj→add_dataset(`*@array*`)`

`$obj→add_dataset(\`*@array_ref*`)`

Adds a dataset to the object. The argument is an array or a reference to an array. Generally, the first array added is interpreted as being the x tick labels. The subsequent arrays contain the data points. E. g., after the calls

```
$obj→add_dataset('Harry', 'Sally');
$obj→add_dataset(5, 8);
```

Chart will draw a picture with two bars and label them 'Harry' and 'Sally'.

Some modules will operate slightly differently. Have a look at the description of the specific subclass to get more information. Such

differences will also come up if you want to use the `xy_plot` option in order to create a x–y graph.

`$obj→add_pt(`*@array*`)`
`$obj→add_pt(\`*@array_ref*`)`

This is a different method for adding data to a Chart object. The argument can be an array or a reference to an array. If you use this method, Chart wants the complete data of one data point, i. e., all the data that are associated with the same x value specified first in this call. E. g.,

```
$obj→add_pt('Harry', 5);
$obj→add_pt('Sally', 8);
```

would create the same graph as the example for `add_dataset()` above.

`$obj→add_datafile("`*filename*`", `*type*`)`
`$obj→add_datafile(`*$filehandle*`, `*type*`)`
`$obj→add_datafile()`

This method adds the contents of a complete data file to the chart object. *type* can be '`set`' or '`pt`'. In the former case, '`set`', each line in the data file must represent a complete data set (*data series*). The values of the set must be separated by whitespace. E. g., the file contents could look like this:

```
Harry Sally
3 8
2 1
```

If the argument is '`pt`', the lines of the file must look analogous to the parameter arrays used by method `add_pt()`: Each line includes all the values of one data point (i. e., all the y values associated with the same x value), also separated by whitespace. E. g.:

```
Harry 3 2
Sally 8 1
```

`$obj→get_data()`

If you want a copy of the data that have been added so far, make a call to this method like so:

```
$dataref = $obj→get_data();
```

This will return a reference to an array of references to datasets. For example, you can get the x tick labels by:

```
@x_labels = @{$dataref->[0]};
```

`$obj→clear_data()`

This is the method to remove all data that may have been entered until now.

`$obj→set(`*attribute*$_1$ `⇒`*value*$_1$`, ..., `*attribute*$_n$ `⇒`*value*$_n$`)`
`$obj→set(`*%hash*`)`
`$obj→set(`*attribute1*`, `*value*$_1$`, ..., `*attribute*$_n$`, `*value*$_n$`)`
`$obj→set(`*@array*`)`

Use this method to change the attributes of the chart object. `set()` looks for a hash of keys and values or an array of keys and values. E. g.,

```
$obj→set('title' ⇒'The title of the image');
```

would set the title. This would do the same job:

```
%hash = ('title' ⇒'The title of the image');
$obj→set(%hash);
```

`$obj→png("`*filename*`")`
`$obj→png($`*filehandle*`)`
`$obj→png(`*FILEHANDLE*`)`
`$obj→png("`*filename*`", \`*@data*`)`
`$obj→png()`

This method creates a PNG file. The file parameter can be a file name, a reference to a filehandle or a filehandle itself. If the file does not exist, Chart will create it for you. If there is already a file, Chart will overwrite it. In case of an error, the file is not created.

You can also add data to a Chart object through its `png()` method. The *@data* array should contain references to arrays of data, with the first array reference pointing to an array of x labels. *@data* might look like this:

```
@data = (['Harry', 'Sally'], [5, 8], [50, 80]);
```

This would set up a graph with two datasets and three data points in these sets.

`$obj→jpeg("`*filename*`")`
`$obj→jpeg(`*$filehandle*`)`
`$obj→jpeg(`*FILEHANDLE*`)`
`$obj→jpeg("`*filename*`", \@data)`
`$obj→jpeg()`

This is the method to create JPEG files. It works analogously to the `png()` method.

`$obj→cgi_png()`
`$obj→cgi_jpeg()`

With the CGI methods you can create dynamic images for your web

site. The CGI methods will print the chart along with the appropriate HTTP header to STDOUT, allowing you to call chart-generating scripts directly from your HTML pages (e. g., with a '⟨`img src="image.pl" /`⟩' HTML tag).

`$obj→imagemap_dump()`

Chart can also return pixel position information so that you can create image maps from the files generated by Chart. Simply set the '`imagemap`' option to '`true`' before you generate the file, then at the end call the `imagemap_dump()` method to retrieve the information. A structure will be returned almost identical to the *@data* array described above to pass the data into Chart.

```
$imagemap_data = $obj→imagemap_dump();
```

Instead of single data values, references to arrays of pixel information are passed. For the classes Chart::Bars, Chart::HorizontalBars, Chart::Pareto and Chart::StackedBars, the arrays will contain two x–y pairs (specifying the upper left and the lower right corner of the bar). Compare to:

```
($x1,$y1,$x2,$y2) = @{$imagemap_data→[$dataset][$datapoint]};
```

For the classes Chart::Lines, Chart::Points, Chart::LinesPoints and Chart::Split, the arrays will contain a single x–y pair (specifying the center of the point). Compare to:

```
($x, $y) = @{$imagemap_data→[$dataset][$datapoint]};
```

A few caveats apply here. First of all, Chart uses the GD module by Lincoln Stein to draw lines, circles, strings, and so on. GD treats the upper-left corner of the PNG/JPEG image as the reference point, therefore, positive y values are measured from the top of the image, not from the bottom. Second, these values will mostly contain long decimal values. GD, of course, has to truncate these to integer pixel coordinates. In a worst-case scenario, this will result in an error of one pixel on your imagemap. If this is really an issue, your only option is to experiment with it, or to contact Lincoln Stein and ask him. Third, please remember that the 0^{th} dataset will be empty, since that is the place for the data point labels on the x axis.

Attributes/Options:

These are the options which take effect on most Chart types. There are three different kinds of attributes:

- attributes expecting a number for value (e. g., the number of pixels),
- attributes expecting a textual value (e. g., the title of the chart),

- attributes expecting a Boolean value.

Before Version 2.5 of the module, the Boolean value '`true`' was represented by the string `'true'`, and the Boolean value '`false`' was represented by the string `'false'`. For all other values, the Boolean value was not well-defined. From version 2.5 onwards, the Boolean value '`true`' may be represented by any of `1`, `'t'` and `'true'`, where case does not matter. From version 2.5 onwards, the Boolean value '`false`' may be represented by any of `0`, `'f'`, `'false'`, and `undef`, where case does not matter. For all other values, the Boolean value is again not well-defined. Note that this behaviour is closer to the standard Perl way but is not identical, due to the need for backward compatibility in this module.

transparent
: Makes the background of the image transparent if set to '`true`'. Useful for making web page images. However, it does not seem to work for all browsers. Defaults to '`false`'.

png_border
: Sets the number of pixels used as a border between the graph and the edges of the image. Defaults to 10.

graph_border
: Sets the number of pixels used as a border between the title/labels and the actual graph within the image. Defaults to 10.

text_space
: Sets the amount of space left on the sides of text, to make it more readable. Defaults to 3.

title
: Tells Chart what to use for the title of the graph. If empty, no title is drawn. '\\' is treated as a newline. If you want to use normal quotation marks instead of single quotation marks, remember to quote ('\\\\') to get a linebreak. Default is empty.

sub_title
: Writes a subtitle under the title in smaller letters.

x_label
: Tells Chart what text to use as a label for the x axis. If empty, no label is drawn. Default is undef.

y_label
y_label2
: Tells Chart what kind of label should be used for the description of the y axis on the left or the right side accordingly. If empty, no label is drawn. Default is undef.

legend

Specifies the placement of the legend. Valid values are '`left`', '`right`', '`top`', '`bottom`', and '`none`'. Choosing '`none`' tells Chart not to draw a legend. Default is '`right`'.

legend_labels

Sets the values for the labels for the different datasets. Should be assigned a reference to an array of labels. E. g.,

```
@labels = ('foo', 'bar');
$obj->set ('legend_labels' =>\@labels);
```

Default is empty, in which case '`Dataset 1`', '`Dataset 2`', etc. are used as labels.

tick_len

Sets the length of the x and y ticks in pixels. Default is 4.

x_ticks

Specifies how to draw the x tick labels. Valid values are '`normal`', '`staggered`' (labels are drawn alternatingly close to the axis and further away from it), and '`vertical`' (label texts are rotated 90 degrees counter-clockwise). Default is '`normal`'.

y_ticks

The number of ticks to plot on the y scale, including the end points. E. g., for a y axis ranging from 0 to 50, with ticks every 10 units, **y_ticks** should have a value of 6.

min_y_ticks

Sets the minimum number of y ticks to draw when generating the y axis. Default is 6, minimum is 2.

max_y_ticks

Sets the maximum number of y ticks to draw when generating the y axis. Default is 100. This limit is used to avoid plotting an unreasonably large number of ticks if non-round values are used for **min_val** and **max_val**. The value for **max_y_ticks** should be at least 5 times as large as **min_y_ticks**.

min_x_ticks

max_x_ticks

These work similar to **max_y_ticks** and **min_y_ticks**, respectively. Of course, this applies only to x–y plots.

integer_ticks_only

Specifies how to draw the x and y ticks: as floating point ('`false`', '`0`') or as integer numbers ('`true`', '`1`'). If you want integer ticks, it may be better to set the attribute **precision** to zero. Default: '`false`'

skip_int_ticks

If **integer_ticks_only** was set to '`true`' the labels and ticks for the y axis will be drawn every n^{th} tick. (Note that in Chart::HorizontalBars the y axis runs horizontally.) Defaults to 1, i. e., no skipping.

precision

Sets the number of digits after the decimal point. Affects in most cases the y axis only. In x–y plots also affects the x axis, and in pie charts the labels. Defaults to 3.

max_val

Sets the maximum y value on the graph, overriding normal autoscaling. Does not work for Chart::Split charts. Default is undef.

min_val

Sets the minimum y value on the graph, overriding normal autoscaling. Does not work for Split charts. Default is undef. Caution should be used when setting **max_val** and **min_val** to floating point or non-round numbers: The range must start and end on a tick, ticks must have round-number intervals and must include round numbers.

Example: Suppose your dataset has a range of 35...114 units. If you specify these values as **min_val** and **max_val**, respectively, the y axis will be plotted with 80 ticks, so one at every unit. Without specification of **min_val** and **max_val**, the system would autoscale the range to 30...120 with 10 ticks every 10 units. If **min_val** and **max_val** are specified to excessive precision, they may be overridden by the system, plotting a maximum **max_y_ticks** ticks.

include_zero

If '`true`', forces the y axis to include zero even if it is not in the dataset range. Default is '`false`'. – Note: It is better to use this option than to set **min_val** if this is all you want to achieve.

skip_x_ticks

Sets the number of x ticks and x tick labels to skip. (I. e., if **skip_x_ticks** were set to 4, Chart would draw every 4^{th} x tick and x tick label). Default is undef.

custom_x_ticks

This option allows you to specify exactly which x ticks and x tick labels should be drawn. It should be assigned a reference to an array of desired ticks. Just remember that we are counting from the 0^{th} element of the array. (E. g., if **custom_x_ticks** is assigned [0,3,4], then the 0^{th}, 3^{rd}, and 4^{th} x ticks will be displayed) This does not apply to Chart::Split, Chart::HorizontalBars and Chart::Pie.

f_x_tick

Needs a reference to a function which accepts the x tick labels generated by *$data→[0]* as its argument. This function should return a reformatted version of the label as a string. E. g.
`$obj→set ('f_x_tick' ⇒\&formatter;)`
An example for the formatter function: Assume that x labels are seconds since some event. The referenced function could be designed to transform this number of seconds to hours, minutes and seconds.

f_y_tick

Similar to **f_x_tick**, but for y labels.

colors

This option lets you control the colors the chart will use. It takes a reference to a hash. The hash should contain keys mapped to references to arrays of RGB values. E.g.,
`$obj->set('colors' ⇒'background' ⇒[255,255,255]);`
sets the background color to white (which is the default). Valid keys for this hash are

- '`background`' (background color for the chart)
- '`title`' (color of the title)
- '`text`' (all the text in the chart)
- '`x_label`' (color of the x axis label)
- '`y_label`' (color of the primary y axis label)
- '`y_label2`' (color of the secondary y axis label)
- '`grid_lines`' (color of the grid lines)
- '`x_grid_lines`' (color of the x grid lines – on x axis ticks)
- '`y_grid_lines`' (color of the y grid lines – on primary y axis ticks)
- '`y2_grid_lines`' (color of the y2 grid lines – on secondary y axis ticks)
- '`dataset0`' ... '`dataset63`' (the different datasets)
- '`misc`' (everything else, e. g., ticks, box around the legend)

NB. For composite charts, there is a limit of eight datasets per component. The colors for '`dataset8`' through '`dataset15`' will be the same as those for '`dataset0`' through '`dataset7`' for the second component chart.

title_font

This option changes the font of the title line. The value must be a GD font, e. g., `GD::Font→Large`.

label_font

This option changes the font of the labels. The value must be a GD font.

legend_font

This option changes the font for the legend text. The value must be a GD font.

tick_label_font

This option changes the font of the ticks. The value must be a GD font.

grey_background

Puts a nice soft grey background on the actual data plot when set to '`true`'. Default is '`true`'.

x_grid_lines

Draws grid lines matching up to x ticks if set to '`true`'. Default is '`false`'.

y_grid_lines

Draws grid lines matching up to y ticks if set to '`true`'. Default is '`false`'.

grid_lines

Draws grid lines matching up to x and y ticks if set to '`true`'. Default is '`false`'.

imagemap

Lets Chart know that you are going to ask for information about the placement of the data for use in creating an image map from the chart. This information can be retrieved using the `imagemap_dump()` method. NB. The `imagemap_dump()` method cannot be called until after the chart has been generated (e. g., using the `png()` or `cgi_png()` methods).

ylabel2

The label for the secondary (right-hand side) y axis. (In a composite chart, this is the axis for the second component). Default is undef.

no_cache

Adds '`Pragma: no-cache`' to the HTTP header. Be careful with this one, since some older browsers (like Netscape 4.5) are unhappy about POST using this method.

legend_example_size

Sets the length of the example line in the legend. Defaults to 20.

3 Chart::Bars

Name: Chart::Bars

File: Bars.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Bars creates a chart made up of vertical bars. Chart::Bars is a subclass of Chart::Base.

Example:

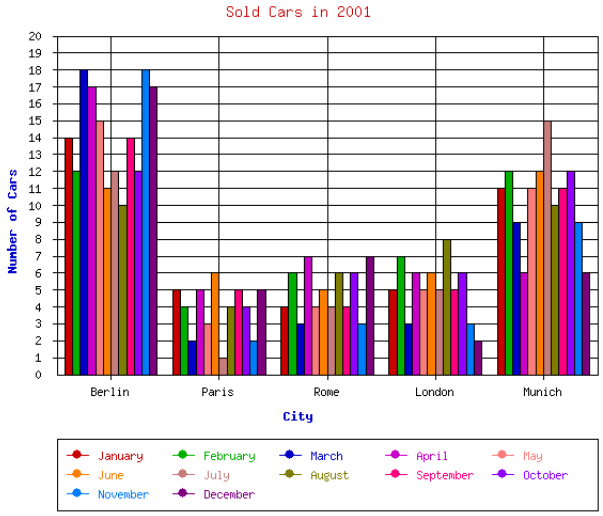

Figure 3: Bar chart

```
use Chart::Bars;

$g = Chart::Bars->new(600,500);

$g->add_dataset('Berlin', 'Paris', 'Rome', 'London', 'Munich');
$g->add_dataset(14, 5, 4, 5, 11);
$g->add_dataset(12, 4, 6, 7, 12);
$g->add_dataset(18, 2, 3, 3, 9);
$g->add_dataset(17, 5, 7, 6, 6);
$g->add_dataset(15, 3, 4, 5, 11);
$g->add_dataset(11, 6, 5, 6, 12);
```

```
$g->add_dataset(12, 1, 4, 5, 15);
$g->add_dataset(10, 4, 6, 8, 10);
$g->add_dataset(14, 5, 4, 5, 11);
$g->add_dataset(12, 4, 6, 6, 12);
$g->add_dataset(18, 2, 3, 3, 9);
$g->add_dataset(17, 5, 7, 2, 6);

%hash = ('title' => 'Sold Cars in 2001',
         'text_space'         => 5,
         'grey_background'    => 'false',
         'integer_ticks_only' => 'true',
         'x_label'            => 'City',
         'y_label'            => 'Number of Cars',
         'legend'             => 'bottom',
         'legend_labels'      => ['January',  'February',
                                  'March',    'April',
                                  'May',      'June',
                                  'July',     'August',
                                  'September','October',
                                  'November', 'December'
                                 ],
         'min_val'            => 0,
         'max_val'            => 20,
         'grid_lines'         =>'true',
         'colors'             => {'title'   => 'red',
                                  'x_label' => 'blue',
                                  'y_label' => 'blue'
                                 }
        );

$g->set(%hash);

$g->png("bars.png");
```

Constructor:

An object instance of Chart::Bars can be created with the constructor `new()`:

```
$obj = Chart::Bars→new();
$obj = Chart::Bars→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Bars object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

spaced_bars

Leaves some space between each group of bars when set to '`true`'. This usually make it easier to read a bar chart. Default is '`true`'.

y_axes

Tells Chart::Bars where to place the y axis. Valid values are '`left`', '`right`' and '`both`'. Defaults to '`left`'.

4 Chart::Composite

Name: Chart::Composite

File: Composite.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Composite creates a two component chart with two types of charts which are layered one above each other. Just set the option **composite_info**. For example, you can create a two component chart with bars and lines. A composite chart does not make sense with all combinations of chart types, but it works pretty good with Lines, Points, LinesPoints and Bars. Note that two similar chart types may come into visual conflict. Chart::Composite can do only composite charts made up of two components. Chart::Composite is a subclass of Chart::Base.

Example:

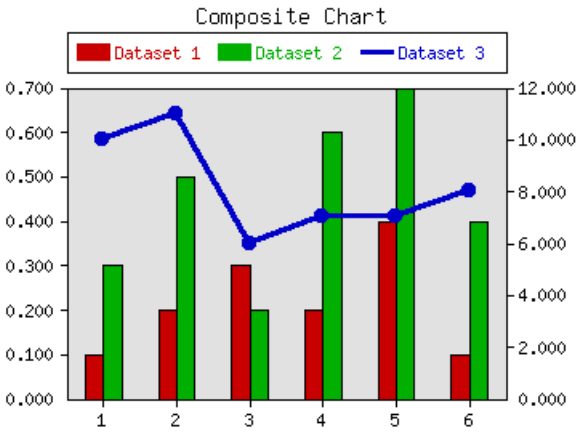

Figure 4: Composite chart

```
use Chart::Composite;

$g = Chart::Composite->new();
```

```
$g->add_dataset(1, 2, 3, 4, 5, 6);
$g->add_dataset(0.1, 0.2, 0.3, 0.2, 0.4, 0.1);
$g->add_dataset(0.3, 0.5, 0.2, 0.6, 0.7, 0.4);
$g->add_dataset(10,  11,  6,   7,   7,   8);

$g->set('composite_info' => [ ['Bars',        [1, 2]],
                              ['LinesPoints', [3]  ]
                            ],
        'title'                     => 'Composite Chart',
        'legend'                    => 'top',
        'legend_example_height'     => 'true',
        'legend_example_height0..1' => 10,
        'legend_example_height2'    =>  3,
       );
$g->set('include_zero'  => 'true');

$g->png("composite.png");
```

Constructor:

An object instance of Chart::Composite can be created with the constructor `new()`:

```
$obj = Chart::Composite→new();
$obj = Chart::Composite→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Composite object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

brush_size1

brush_size2

If using component charts having **brush_size** as one of their attributes, you can define the sizes of the brushes individually. Default is 6 (pixel).

composite_info

This option is only used for composite charts. It contains the information which types to use for the two component charts, and which datasets belong to which component chart. It should be a reference to an array of array references, containing information like the following:
$obj→set ('composite_info' ⇒[['Bars', [1,2]], ['Lines', [3,4]]]);

This example would set the two component charts to be a bar chart and a line chart. It would use the first two data sets for the bar chart and the second two data sets for the line chart. The default is undef. Note that the numbering starts at 1, not at 0 like most of the other numbered things in Chart, because index 0 refers to the x values which are shared by the two component charts. The ordering of the components may be important, since the first component is drawn first and then (partially) overdrawn with the second component. E. g., when composing a line graph and a bar graph, it is safer to have the bars in the first component since otherwise the line(s) might be hidden behind them.

f_y_tick1
f_y_tick2

Needs a reference to a function which uses the y tick labels for the primary and for the secondary y axis, respectively. These functions should return a reformatted version of the label as a string. E. g.

```
$obj→set ('f_y_tick1' ⇒\&formatter1);
$obj→set ('f_y_tick2' ⇒\&formatter2);
```

max_val1
max_val2

Only for composite charts. These options specify the maximum y value for the first and the second component, respectively. Both default to undef.

min_val1
min_val2

Only for composite charts. These options specify the minimum y value for the first and the second component, respectively. Both default to undef.

legend_example_height

Only for composite charts. This option changes the thickness of the lines in the legend. If 'legend_example_height' is set to 'true' the

thickness of each legend line can be changed individually. Default is false. E. g.

```
$obj→set ('legend_example_height' ⇒'true');
$obj→set ('legend_example_height0' ⇒'3');
$obj→set ('legend_example_height1..4' ⇒'10');
```

This example would set the thickness of the first line in the legend to 3, and the thicknesses of the following 4 lines to 10 (using the same indexing scheme as in 'composite_info'). The default value for each individual entry is 1, i. e. a 'normal' line is drawn. It is not possible to change a 'legend_example_height#'(where # denotes a dataset number) which was once defined. (The first setting will remain unchanged.)

same_y_axes

Forces both component charts in a composite chart to use the same maximum and minimum y values if set to 'true'. This helps to keep some composite charts from being too confusing. Default is undef.

y_ticks1

y_ticks2

The number of y ticks to use on the primary and on the secondary y axis on a composite chart, respectively. Please note that if you just set the 'y_ticks' option, both axes will use that number of y ticks. Both default to undef.

5 Chart::Direction

Name: Chart::Direction

File: Direction.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Direction creates a diagram based on polar coordinates. This type of diagram is occasionally referred to as a *radial* or as a *radar* chart. Chart::Direction plots data specified by angle (e. g., wind direction) and absolute value (e. g., wind strength). The first dataset to add is always the set of angles in degrees. The second set contains the absolute values. How additional datasets should be entered depends on the option **pairs** (cf. below). By default, Chart::Direction will draw a point chart. You can also get a lines chart by setting the option **point** to '`false`' and the option **line** to '`true`'. If you want a lines and point chart, then set both **point** and **line** to '`true`'. In addition, Chart::Direction plots arrows from the center to the point or to the end of the line if the option **arrow** is set to '`true`'. Chart::Direction is a subclass of Chart::Base.

Example:

```
use Chart::Direction;
$g = Chart::Direction->new(500,500);

$g->add_dataset( 0, 100, 50, 200, 280, 310);
$g->add_dataset(30,  40, 20,  35,  45,  20);

$g->add_dataset(10, 110, 60, 210, 290, 320);
$g->add_dataset(20,  30, 40,  20,  35,  45);

$g->add_dataset(20, 120, 70, 220, 300, 330);
$g->add_dataset(45,  20, 30,  40,  20,  35,);

%hash = ( 'title'           => 'Direction Demo',
          'angle_interval'  => 45,
          'precision'       => 0,
          'arrow'           => 'true',
          'point'           => 'false',
          'include_zero'    => 'true',
```

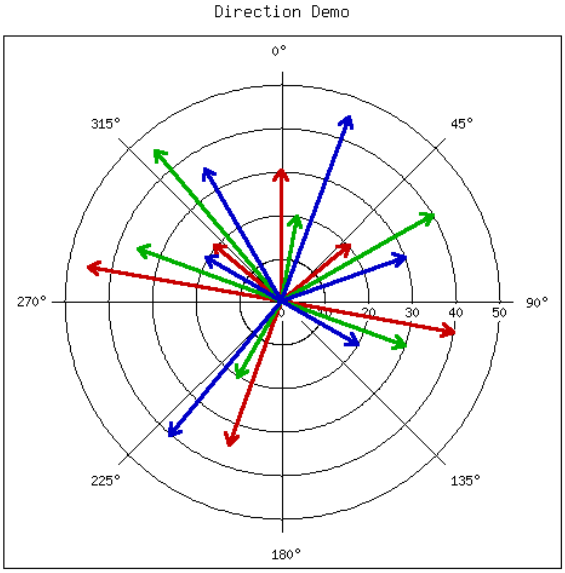

Figure 5: Direction chart

```
        'pairs'           => 'true',
        'legend'          => 'none',
        'grey_background' => 'false'
    );

$g->set(%hash);

$g->png("direction.png");
```

Constructor:

An object instance of Chart::Direction can be created with the constructor `new()`:

```
$obj = Chart::Direction→new();
$obj = Chart::Direction→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Direction object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

angle_interval

This option tells Chart::Direction how many angle lines should be drawn. It is the difference between two angle lines. The default value is 30, which means that one line will be drawn every 30 degrees. Not all values are permissible; the valid ones are: 0, 5, 10, 15, 20, 30, 45, and 90. If you choose 0, Chart::Direction will draw no lines.

arrow

Draws an arrow from the center of the chart to the point if set to '`true`'. By default '`false`'.

brush_size

Sets the width of the lines in pixels. Default is 6.

line

Connects the points with lines if set to '`true`'. Defaults to '`false`'.

max_circles

Sets the maximum number of circles to draw when generating the set of circles. Default is 100. This limit is used to avoid plotting an unreasonably large number of circles if non-round values are used for **min_val** and **max_val**. The value for **max_circles** should be at least 5 times that of **min_circles**.

min_circles

Sets the minimum number of circles to draw when generating a scale. Default is 4, minimum is 2.

pairs

This option tells Chart::Direction how to handle additional datasets. If **pairs** is set to '`true`', Chart::Direction uses the first dataset as a set of degrees and the second dataset as a set of values. Then, the third set is a set of degrees and the fourth a set of values, and so forth. If **pairs** is set to '`false`', Chart::Direction uses the first dataset as a set of angles and all following datasets as sets of values. Defaults to '`false`'.

point

Indicates to draw points for representing the data values. Possible values: '`true`' and '`false`', by default '`true`'.

pt_size

Sets the radius of the points in pixels. Default is 18.

sort

Sorts the data in ascending order if set to '`true`'. Should be set if the input data is not sorted and **line** is set to '`true`'. Defaults to '`false`'.

6 Chart::ErrorBars

Name: Chart::ErrorBars

File: ErrorBars.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::ErrorBars creates a point chart with error bars. This class expects the error values within the data array. By use of the `add_dataset()` method the error values are the next two sets after the y values. The first set after the y values has to be the set of values for the upper error bounds. The next set is the array of the lower error bounds. Note that the error values are not specified absolutely but rather as offsets from the y value: the upper error values will be added to the y values, the lower error values will be subtracted.

If you want to use the same value for the upper and lower error, you can set the **same_error** option to '`true`'. In this case only the set after the y values is interpreted as a set of errors.

Of course, it is also possible to use the `add_pt()` method in the appropriate way to achieve the same results. Chart::ErrorBars is a subclass of Chart::Base.

Example:

```
use Chart::ErrorBars;
$g = Chart::ErrorBars->new();

# the x values
$g->add_dataset(qw(1    1.1  1.2  1.3  1.4  1.5  1.6  1.7
                   1.8  1.9  2    2.1  2.2  2.3  2.4  2.5));
# the y values
$g->add_dataset(qw(1    1.1  1.2  1.1  1.14 1.15 1.26 1.2
                   1.1  1.19 1.2  1.4  1.6  2.0  2.5  3.1));
# the upper errors
$g->add_dataset(qw(0.4  0.1  0.2  0.1  0.14 0.15 0.26 0.27
                   0.1  0.19 0.2  0.1  0.1  0.2  0.1  0.3));
# the lower errors
$g->add_dataset(qw(0.2  0.11 0.12 0.11 0.2  0.3  0.12 0.27
                   0.11 0.3  0.2  0.2  0.2  0.1  0.1  0.2));
```

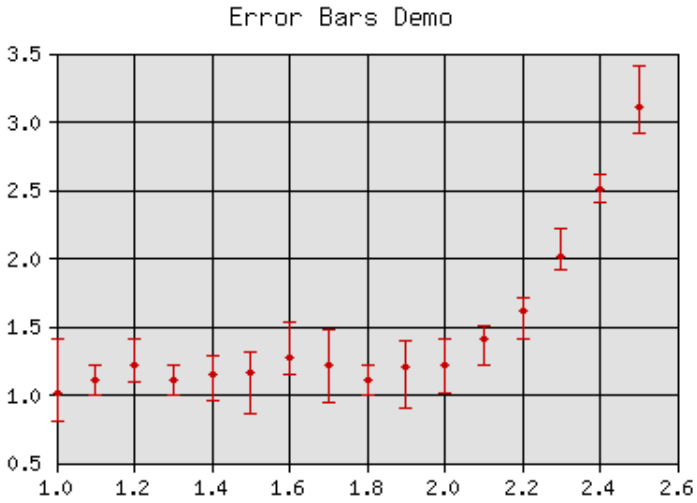

Figure 6: Error bars chart

```
$g->set( 'xy_plot'    => 'true',
         'precision'  => 1,
         'pt_size'    => 10,
         'brush_size' => 2,
         'legend'     => 'none',
         'title'      => 'Error Bars Demo',
         'grid_lines' => 'true'
       );

$g->png("errorbars.png");
```

Constructor:

An object instance of Chart::ErrorBars can be created with the constructor `new()`:

```
$obj = Chart::ErrorBars→new();
$obj = Chart::ErrorBars→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two argu-

ments, *width* and *height*, it will return a Chart::ErrorBars object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

brush_size

Sets the width of the lines in pixels. Default is 6.

pt_size

Sets the radius of the points in pixels. Default is 18.

same_error

Tells Chart::ErrorBars that you want to use the same values for upper and lower error bounds if set to '`true`'. Then you have to add just one set of error values. Defaults to '`false`'.

sort

Sorts the data in ascending order if set to '`true`'. Should be set if the input data is not sorted. Defaults to '`false`'.

xlabels

xrange

This pair of options allows arbitrary positioning of x axis labels. The two options must either both be specified or both be omitted. **xlabels** is a reference to 2-element array. The first of the elements is a nested (reference to an) array of strings that are the labels. The second element is a nested (reference to an) array of numbers that are the x values at which the labels should be placed. **xrange** is a 2-element array specifying the minimum and maximum x values on the axis. E. g.,

```
@labels = (['Jan', 'Feb', 'Mar'],
           [10,    40,    70   ]);
$chart->set(xlabels => \bs @labels,
            xrange  => [0, 100]
           );
```

xy_plot

Forces Chart::ErrorBars to plot a x–y graph if set to '`true`', i. e., to treat the x axis as numeric. Very useful for plots of mathematical functions. Defaults to '`false`'.

y_axes

Tells Chart::ErrorBars where to place the y axis. Valid values are '`left`', '`right`' and '`both`'. Defaults to '`left`'.

7 Chart::HorizontalBars

Name: Chart::HorizontalBars

File: HorizontalBars.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::HorizontalBars creates a chart of horizontally oriented bars. Chart::HorizontalBars is a subclass of Chart::Base.

Example:

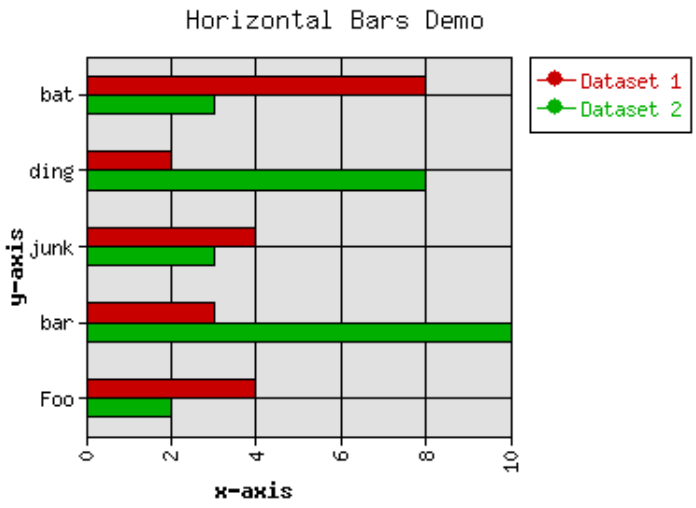

Figure 7: Chart with horizontal bars

```
use Chart::HorizontalBars;

$g = Chart::HorizontalBars->new();
$g->add_dataset('Foo', 'bar', 'junk', 'ding', 'bat');
$g->add_dataset(4, 3, 4, 2, 8);
$g->add_dataset(2, 10, 3, 8, 3);

%hash = ( 'title'        => 'Horizontal Bars Demo',
          'grid_lines'   => 'true',
```

```
          'x_label'      => 'x axis',
          'y_label'      => 'y axis',
          'include_zero' => 'true',
          'x_ticks'      => 'vertical',
         );
$g->set(%hash);

$g->png("hbars.png");
```

Constructor:

An object instance of Chart::HorizontalBars can be created with the constructor `new()`:

```
$obj = Chart::HorizontalBars→new();
$obj = Chart::HorizontalBars→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::HorizontalBars object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

skip_y_ticks

Does the same fo the y axis in a horizontal chart as **skip_x_ticks** does for other charts. Defaults to 1.

spaced_bars

Leaves some space between each group of bars when set to '`true`'. This usually make it easier to read a bar chart. Default is '`true`'.

y_axes

Tells Chart::HorizontalBars where to place the y axis. '`left`', '`right`' and '`both`'. Defaults to '`left`'.

8 Chart::Lines

Name: Chart::Lines

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Lines creates a lines chart. (If you want the data points marked with symbols, check Chart::LinesPoints on page 34.) Chart::Lines is a subclass of Chart::Base.

Example:

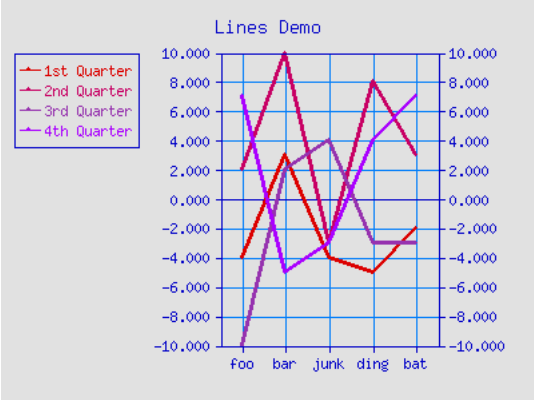

Figure 8: Lines chart

```
use Chart::Lines;

$g = Chart::Lines->new();
$g->add_dataset('foo', 'bar', 'junk', 'ding', 'bat');
$g->add_dataset( -4,  3, -4, -5, -2);
$g->add_dataset(  2, 10, -3,  8,  3);
$g->add_dataset(-10,  2,  4, -3, -3);
$g->add_dataset(  7, -5, -3,  4,  7);

%hash = ('legend_labels' => ['1st Quarter', '2nd Quarter',
                             '3rd Quarter', '4th Quarter'],
         'y_axes'          => 'both',
         'title'           => 'Lines Demo',
         'grid_lines'      => 'true',
```

```
        'legend'              => 'left',
        'legend_example_size' => 20,
        'colors' => {'text'       => 'blue',
                     'misc'       => 'blue',
                     'background' => 'grey',
                     'grid_lines' => 'light_blue',
                     'dataset0'   => [220,0,0],
                     'dataset1'   => [200,0,100],
                     'dataset2'   => [150,50,175],
                     'dataset3'   => [170,0,255]
                    }
      );

$g->set(%hash);

$g->png("lines.png");
```

Constructor:

An object instance of Chart::Lines can be created with the constructor `new()`:

```
$obj = Chart::Lines→new();
$obj = Chart::Lines→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Lines object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

brush_size

Sets the width of the lines in pixels. Default is 6.

sort

Sorts the data in ascending order if set to '`true`'. Should be set if the input data is not sorted. Defaults to '`false`'.

stepline

The points are connected by a stepping function,instead of by a direct line if set to '`true`'. Defaults to '`false`'.

stepline_mode

Determines whether to plot each stepping line at the level of the start of the interval (if set to '`begin`') or at its end if set to '`end`'. Defaults to '`begin`'.

xlabels

xrange

This pair of options allows arbitrary positioning of x axis labels. The two options must either both be specified or both be omitted. **xlabels** is a reference to 2-element array. The first of the elements is a nested (reference to an) array of strings that are the labels. The second element is a nested (reference to an) array of numbers that are the x values at which the labels should be placed. **xrange** is a 2-element array specifying the minimum and maximum x values on the axis. E. g.,

```
@labels = (['Jan', 'Feb', 'Mar'],
           [10,    40,    70   ]);
$chart->set(xlabels => \bs @labels,
            xrange  => [0, 100]
           );
```

xy_plot

Forces Chart::Lines to plot a x–y graph if set to '`true`', i. e., to treat the x axis as numeric. Very useful for plots of mathematical functions. Defaults to '`false`'.

9 Chart::LinesPoints

Name: Chart::LinesPoints

File: LinesPoints.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::LinesPoints creates a lines chart where additionally the individual data points are marked with a symbol. (If you want just lines without additional symbols, check Chart::Lines on page 31. If you want just symbols for the data points but no lines, check Chart::Points on page 46.) Chart::LinesPoints is a subclass of Chart::Base.

Example:

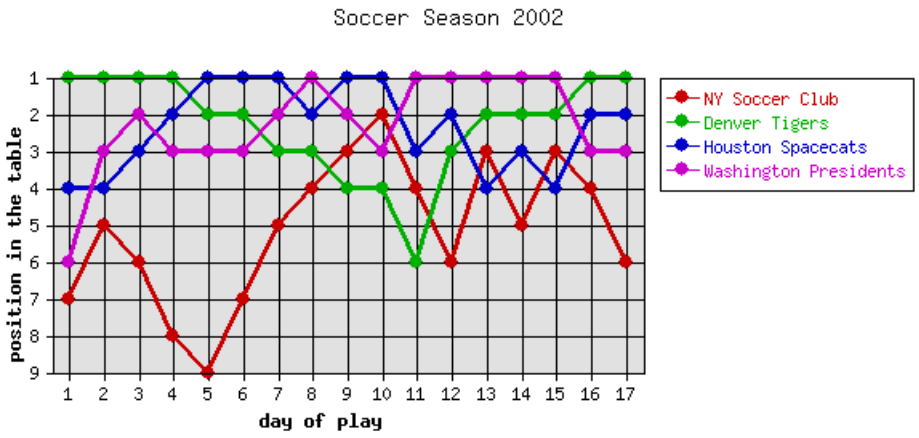

Figure 9: Linespoints chart

```
use Chart::LinesPoints;
use strict;

my (@data1, @data2, @data4, @data3, @labels, %hash, $g);

@labels = qw(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17);
@data1  = qw (-7 -5 -6 -8 -9 -7 -5 -4 -3 -2 -4 -6 -3 -5 -3 -4 -6);
```

```
@data2  = qw (-1 -1 -1 -1 -2 -2 -3 -3 -4 -4 -6 -3 -2 -2 -2 -1 -1);
@data3  = qw (-4 -4 -3 -2 -1 -1 -1 -2 -1 -1 -3 -2 -4 -3 -4 -2 -2);
@data4  = qw (-6 -3 -2 -3 -3 -3 -2 -1 -2 -3 -1 -1 -1 -1 -1 -3 -3);

$g = Chart::LinesPoints->new(600,300);
$g->add_dataset(@labels);
$g->add_dataset(@data1);
$g->add_dataset(@data2);
$g->add_dataset(@data3);
$g->add_dataset(@data4);

%hash = ('integer_ticks_only' => 'true',
         'title'         => 'Soccer Season 2002\n ',
         'legend_labels' => ['NY Soccer Club', 'Denver Tigers',
                             'Houston Spacecats',
                             'Washington Presidents'],
         'y_label'       => 'position in the table',
         'x_label'       => 'day of play',
         'grid_lines'    => 'true',
         'f_y_tick'      => \&formatter,
        );

$g->set( %hash);

$g->png("d_linesp2.png");

# Just a trick to have the y scale start at the biggest point:
# Initialise with negative values, remove the minus sign!
sub formatter {
  my $label = shift;
  $label    = substr($label, 1);
  return $label;
}
```

Constructor:

An object instance of Chart::LinesPoints can be created with the constructor `new()`:

```
$obj = Chart::LinesPoints→new();
$obj = Chart::LinesPoints→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::LinesPoints object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

brush_size

Sets the width of the lines in pixels. Default is 6.

brushStyle

Define the share of the points. The share may be specified to each dataset.

The possible shapes of the 'points' are

- FilledCircle (default),
- circle,
- donut,
- OpenCircle,
- triangle,
- upsidedownTriangle,
- square,
- hollowSquare,
- OpenRectangle,
- fatPlus,
- Star,
- OpenStar,
- FilledDiamond,
- OpenDiamond

To apply a different brush style to different data sets the following example of code can be used:

```
$g->set(brushStyles => { dataset0 => 'fatPlus', dataset1 => 'hollowSquare' });
```

pt_size

Sets the radius of the points in pixels. Default is 18.

sort

Sorts the data in ascending order if set to '`true`'. Should be set if the input data is not sorted. Defaults to '`false`'.

stepline

The points are connected by a stepping function,instead of by a direct line if set to '`true`'. Defaults to '`false`'.

stepline_mode

Determines whether to plot each stepping line at the level of the start of the interval (if set to '`begin`') or at its end if set to '`end`'. Defaults to '`begin`'.

xlabels

xrange

This pair of options allows arbitrary positioning of x axis labels. The two options must either both be specified or both be omitted. **xlabels** is a reference to 2-element array. The first of the elements is a nested (reference to an) array of strings that are the labels. The second element is a nested (reference to an) array of numbers that are the x values at which the labels should be placed. **xrange** is a 2-element array specifying the minimum and maximum x values on the axis. E. g.,

```
@labels = (['Jan', 'Feb', 'Mar'],
           [10,    40,    70   ]);
$chart->set(xlabels => \bs @labels,
            xrange  => [0, 100]
           );
```

xy_plot

Forces Chart::LinesPoints to plot a x–y graph if set to '`true`', i. e., to treat the x axis as numeric. Very useful for plots of mathematical functions. Defaults to '`false`'.

y_axes

Tells Chart::LinesPoints where to place the y axis. Valid values are '`left`', '`right`' and '`both`'. Defaults to '`left`'.

10 Chart::Mountain

Name: Chart::Mountain

File: Mountain.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Mountain creates a mountain chart, i. e., the individual data sets are stacked and the areas under the curves are colour filled. The first data set will be shown at the top of the stack, the last at the bottom. Chart::Mountain is a subclass of Chart::Base.

Example:

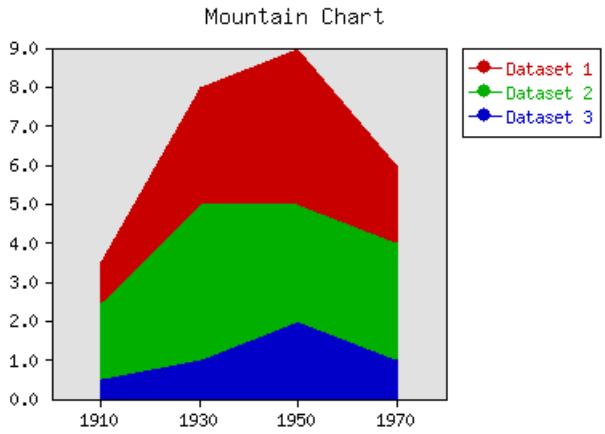

Figure 10: Mountain chart

```
use Chart::Mountain;

$g = Chart::Mountain->new();

@data = [ [1910, 1930, 1950, 1970],
          [1, 3, 4, 2],
          [2, 4, 3, 3],
          [0.5, 1, 2, 1]];
```

```
$g->set('title'      => 'Mountain Chart',
        'grid_lines' => 'false',
        'precision'  => 1);

$g->png("mountain.png", @data);
```

Constructor:

An object instance of Chart::Mountain can be created with the constructor `new()`:

```
$obj = Chart::Mountain→new();
$obj = Chart::Mountain→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Mountain object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

y_axes

Tells Chart::Mountain where to place the y axis. Valid values are '`left`', '`right`' and '`both`'. Defaults to '`left`'.

11 Chart::Pareto

Name: Chart::Pareto

File: Pareto.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Pareto creates a Pareto chart, i. e., a set of absolute values overlaid with a line chart of the accumulated values. (This latter curve is also known as an *empirical cumulative distribution function* or as a *Lorenz curve*.) This representation usually makes sense only if the values are sorted (either in ascending or in descending order). Chart::Pareto plots only one data set and its labels. Chart::Pareto is a subclass of Chart::Base.

Example:

```
use Chart::Pareto;

$g = Chart::Pareto->new(500,400);
$g->add_dataset('1st week', '2nd week', '3rd week', '4th week',
                '5th week', '6th week', '7th week', '8th week',
                '9th week', '10th week');
$g->add_dataset(37, 15, 9, 4, 3.5, 2.1, 1.2, 1.5, 6.2, 16);

%hash = ('colors' => { 'dataset0' => 'mauve',
                       'dataset1' => 'light_blue',
                       'title' => 'orange'
                     },
         'title'              => 'Visitors at the Picasso Exhibition',
         'integer_ticks_only' => 'true',
         'skip_int_ticks'     => 5,
         'grey_background'    => 'false',
         'max_val'            => 100,
         'y_label'            => 'Visitors in Thousands',
         'x_ticks'            => 'vertical',
         'spaced_bars'        => 'true',
         'legend'             => 'none'
        );

$g->set(%hash);
```

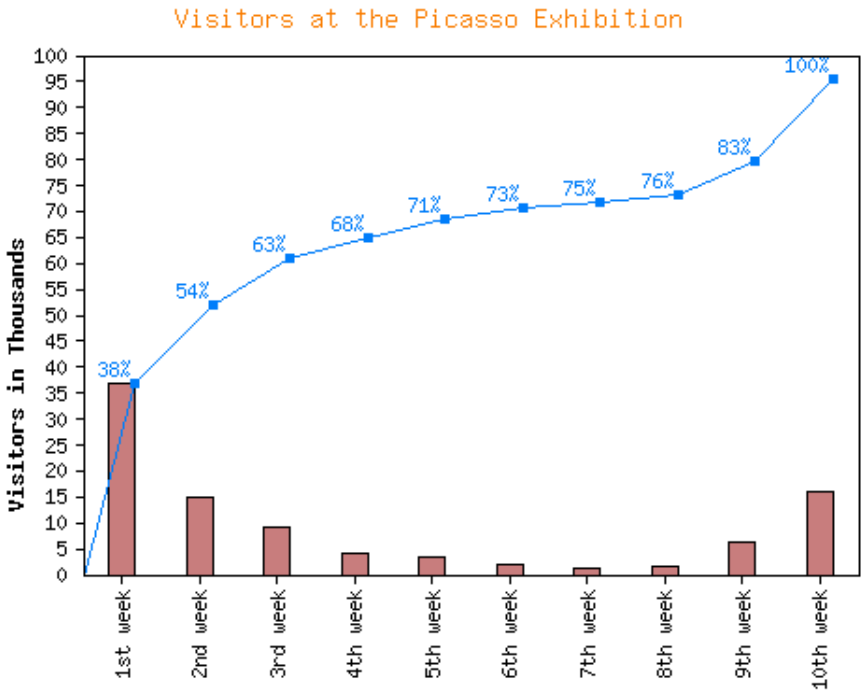

Figure 11: Pareto chart

```
$g->png("pareto.png");
```

Constructor:

An object instance of Chart::Pareto can be created with the constructor `new()`:

```
$obj = Chart::Pareto→new();
$obj = Chart::Pareto→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Pareto object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

sort

Sorts the data in ascending order if set to '`true`'. Should be set if the input data is not sorted. Defaults to '`false`'.

spaced_bars

Leaves some space between each group of bars when set to '`true`'. This usually make it easier to read a bar chart. Default is '`true`'.

y_axes

Tells Chart::Pareto where to place the y axis. Valid values are '`left`', '`right`' and '`both`'. Defaults to '`left`'.

12 Chart::Pie

Name: Chart::Pie

File: Pie.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Pie creates a pie chart. The first added set must contain the labels, the second set the values. Chart::Pie is a subclass of Chart::Base.

Example:

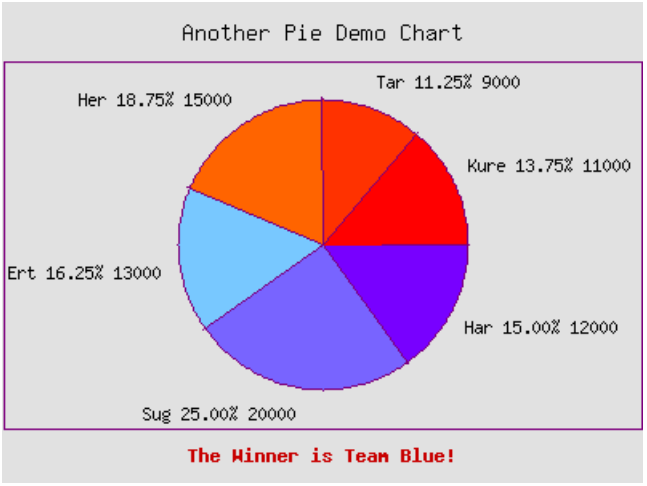

Figure 12: Pie chart

```
use Chart::Pie;

$g = Chart::Pie->new();

$g->add_dataset('Har', 'Sug', 'Ert', 'Her', 'Tar', 'Kure');
$g->add_dataset(12000, 20000 , 13000, 15000, 9000, 11000  );

%opt = ('title'        => 'Another Pie Demo Chart',
        'label_values' => 'both',
        'legend'       => 'none',
```

```
        'text_space'   => 10,
        'png_border'   => 1,
        'graph_border' => 0,
        'colors' => { 'x_label'    => 'red',
                      'misc'       => 'plum',
                      'background' => 'grey',
                      'dataset0'   => [120, 0, 255],
                      'dataset1'   => [120, 100, 255],
                      'dataset2'   => [120, 200, 255],
                      'dataset3'   => [255, 100, 0],
                      'dataset4'   => [255, 50, 0],
                      'dataset5'   => [255, 0, 0],
                    },
        'x_label'      => 'The Winner is Team Blue!',
       );

$g->set(%opt);

$g->png("pie.png");
```

Constructor:

An object instance of Chart::Pie can be created with the constructor `new()`:

```
$obj = Chart::Pie→new();
$obj = Chart::Pie→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Pie object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

label_values

Tells Chart::Pie what kind of value labels to show alongside the pie. Valid values are '`percent`', '`value`', '`both`' and '`none`'. Defaults to '`percent`'.

legend_label_values

Tells Chart::Pie what kind of labels to show in the legend. Valid values are '`percent`', '`value`', '`both`' and '`none`'. Defaults to '`value`'.

legend_lines

The labels drawn alongside the pie are connected with a line to the segment if this option is set to '`true`'.

ring

The pie can have a ring shape instead of the usual disc shape. This option determines the thickness of the ring as a fraction of the radius. Default is 1, i. e., a full pie.

13 Chart::Points

Name: Chart::Points

File: Points.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Points creates a point chart (also called *scattergram*) where the individual data points are marked with a symbol. (If you want lines in addition, check Chart::LinesPoints on page 34.) Chart::Points is a subclass of Chart::Base.

Example:

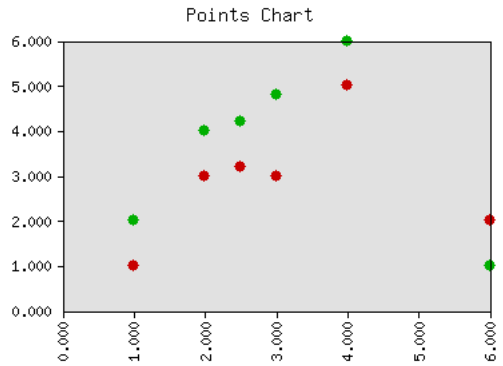

Figure 13: Points chart

```
use Chart::Points;

$g = Chart::Points->new();
$g->add_dataset(1, 4,   3, 6, 2, 2.5);  # x-coordinates
$g->add_dataset(1, 5,   3, 2, 3, 3.2);  # y-coordinates dataset 1
$g->add_dataset(2, 6, 4.8, 1, 4, 4.2);  # y-coordinates dataset 2

@hash = ('title'        => 'Points Chart',
         'xy_plot'      => 'true',
         'x_ticks'      => 'vertical',
         'legend'       => 'none',
```

```
        'sort'         => 'true',
        'precision'    => 3,
        'include_zero' => 'true',
       );

$g->set(@hash);

$g->png("Grafiken/points.png");
```

Constructor:

An object instance of Chart::Points can be created with the constructor `new()`:

```
$obj = Chart::Points→new();
$obj = Chart::Points→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Points object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

pt_size

Sets the radius of the points in pixels. Default is 18.

The points are extended by different brush styles.

brushStyle

Define the share of the points. The share may be specified to each dataset.

The possible shapes of the 'points' are

- FilledCircle (default),
- circle,
- donut,
- OpenCircle,
- triangle,
- upsidedownTriangle,

- square,
- hollowSquare,
- OpenRectangle,
- fatPlus,
- Star,
- OpenStar,
- FilledDiamond,
- OpenDiamond

To apply a different brush style to different data sets the following example of code can be used:

```
$g->set(brushStyles => { dataset0 => 'fatPlus', dataset1 => 'hollowSquare' });
```

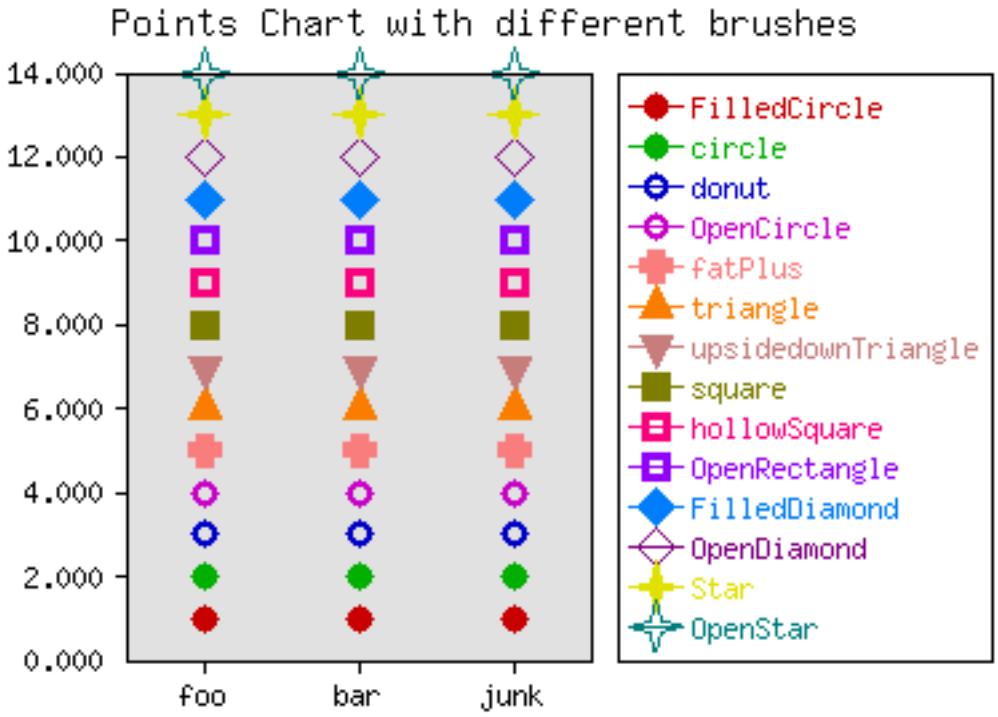

Figure 14: Points chart as an example for brush styles

sort

Sorts the data in ascending order if set to '`true`'. Should be set if the input data is not sorted. Defaults to '`false`'.

xlabels

xrange

This pair of options allows arbitrary positioning of x axis labels. The two options must either both be specified or both be omitted. **xlabels** is a reference to 2-element array. The first of the elements is a nested (reference to an) array of strings that are the labels. The second element is a nested (reference to an) array of numbers that are the x values at which the labels should be placed. **xrange** is a 2-element array specifying the minimum and maximum x values on the axis. E. g.,

```
@labels = (['Jan', 'Feb', 'Mar'],
           [10,    40,    70   ]);
$chart->set(xlabels => \bs @labels,
            xrange  => [0, 100]
           );
```

xy_plot

Forces Chart::Points to plot a x–y graph if set to '`true`', i. e., to treat the x axis as numeric. Very useful for plots of mathematical functions. Defaults to '`false`'.

y_axes

Tells Chart::Points where to place the y axis. Valid values are '`left`', '`right`' and '`both`'. Defaults to '`left`'.

14 Chart::Split

Name: Chart::Split

File: Split.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::Split creates a lines chart where both x and y axes are assumed to be numeric. Split charts are mainly intended for cases where many data points are spread over a wide x range while at the same time the y range is limited. Typical examples are weather or seismic data. The x axis will be split into several intervals of the same length (specified with the mandatory option **interval**). The intervals will be displayed in a stacked fashion. The start of the top interval is set with the mandatory option **start**. Chart::Split will draw only positive x coordinates. The y axis will not be labelled with the y values. Rather, the axis will show only the sequence numbers of the intervals. Chart::Split is a subclass of Chart::Base.

Example:

```
use Chart::Split;

$g = Chart::Split->new(650, 900);

# Get the data from a file and push them into arrays
open(FILE, "data.dat") or die "Can't open the data file!\n";
while (<FILE>) {
  ($x, $y) = split;
   push (@x, $x);
   push (@y, $y);
}
close(FILE);

# Add the data
$g->add_dataset(@x);
$g->add_dataset(@y);

# Set the options
$g->set('xy_plot'        => 'true');
$g->set('legend'         => 'none');
```

```
$g->set('title'          => 'Split Demo');
$g->set('interval'       => 1/288);
$g->set('interval_ticks' => 10);
$g->set('start'          => 260.5);
$g->set('brush_size'     => 1);
$g->set('precision'      => 4);
$g->set('y_label'        => '5 minutes interval');

# Give me a nice picture
$g->png("split.png");
```

Constructor:

An object instance of Chart::Split can be created with the constructor `new()`:

```
$obj = Chart::Split→new();
$obj = Chart::Split→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::Split object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

start

Sets the start value of the first interval. If the x coordinate of the first data point is 0, **start** should also be set to 0. *Required* value for a Chart::Split chart. Defaults to undef.

interval

Sets the interval of one segment to plot. *Required* value for a split chart. Defaults to undef.

interval_ticks

Sets the number of ticks for the x axis. Defaults to 5.

scale

Every y value of a Chart::Split chart will be multiplied by this value, without however changing the sclaing of the y axis. (This might result in some segments being overdrawn by others.) Only useful

if you want to give prominence to the maximal amplitudes of data. Defaults to 1.

sort

Sorts the data in ascending order if set to '`true`'. Should be set if the input data is not sorted. Defaults to '`false`'.

y_axes

Tells Chart::Split where to place the y axis. Valid values are '`left`', '`right`' and '`both`'. Defaults to '`left`'.

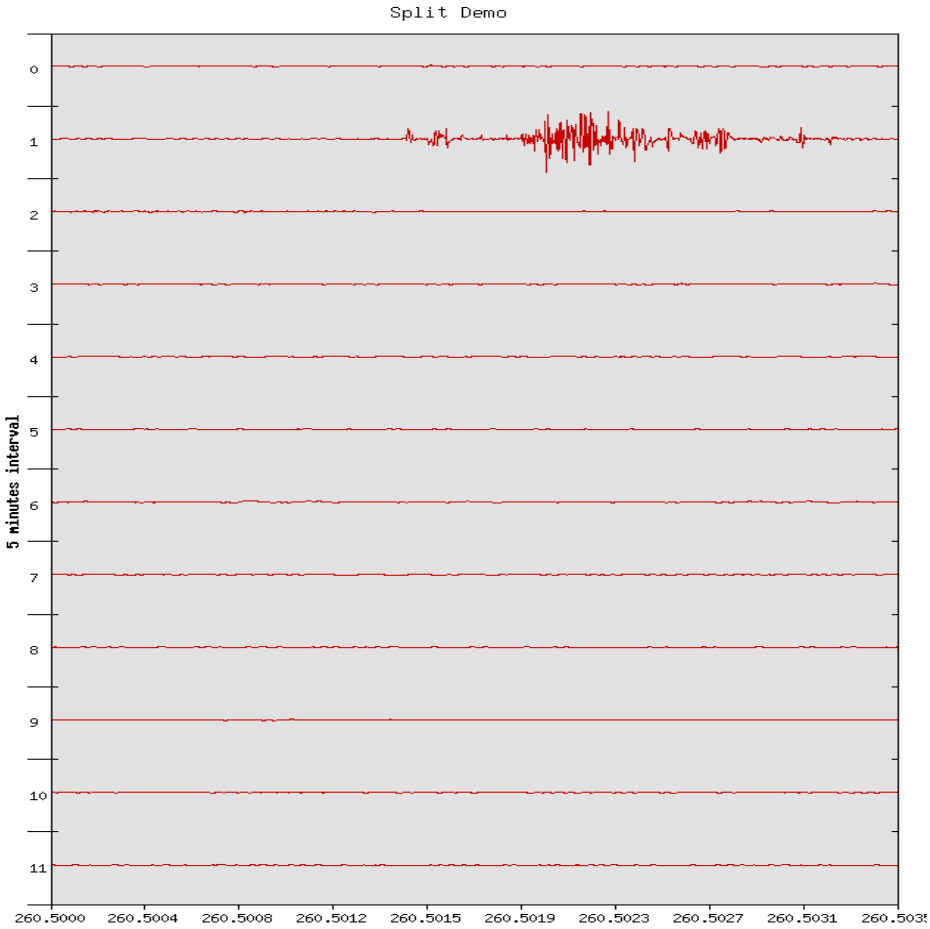

Figure 15: Split chart

15 Chart::StackedBars

Name: Chart::StackedBars

File: StackedBars.pm

Requires: Chart::Base, GD, Carp, FileHandle

Description:

The class Chart::StackedBars creates a chart made up of stacked vertical bars. The first data set will be shown at the bottom of the stack, the last at the top. Chart::StackedBars is a subclass of Chart::Base.

Example:

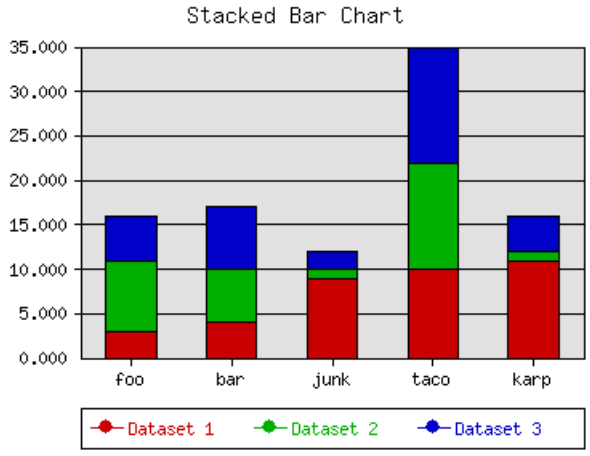

Figure 16: Chart with stacked bars

```
use Chart::StackedBars;

$g = Chart::StackedBars->new();

$g->add_dataset(qw(foo bar junk taco karp));
$g->add_dataset(3, 4, 9, 10, 11);
$g->add_dataset(8, 6, 1, 12, 1);
$g->add_dataset(5, 7, 2, 13, 4);

$g->set('title'        => 'Stacked Bar Chart');
```

```
$g->set('y_grid_lines' => 'true');
$g->set('legend'       => 'bottom');

$g->png("stackedbars.png");
```

Constructor:

An object instance of Chart::StackedBars can be created with the constructor `new()`:

```
$obj = Chart::StackedBars→new();
$obj = Chart::StackedBars→new(width, height);
```

If `new()` is called without arguments, the constructor will return an image of size 300×400 pixels. If `new()` is called with two arguments, *width* and *height*, it will return a Chart::StackedBars object of the desired size.

Methods:

All universally valid methods, see page 5 of class Chart::Base.

Attributes/Options:

All universally valid options, see page 8 of class Chart::Base. In addition, the following options are defined for this class:

spaced_bars

Leaves some space between the individual bars when set to '`true`'. This usually make it easier to read a bar chart, with stacked bars, however, it is not as important as with groups of bars. Default is '`true`'.

y_axes

Tells Chart::StackedBars where to place the y axis. Valid values are '`left`', '`right`' and '`both`'. Defaults to '`left`'.

Index