Contents

1 Description

SYNOPSIS

```
use Chart::type;     (type is one of: Bars, Composite,
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 );

#GifGraph.pm-style API to produce png formatted charts
@data = ( \@x_tick_labels, \@dataset_1, ... \@datase_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();
The similar functions are available for jpeg

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

The Perl module **Chart** creates png or jpeg Files with charts. Chart can also create dynamic charts for web sites.
It is possible to create a lot of different chart types with Chart: Bars, Composite, ErrorBars, HorizontalBars, Lines, LinesPoints, Mountain, Pareto, Pie, Points, Split, StackedBars.
Take a look at their descriptions to see how they work. All of the special types are classes by them self. All this classes have the same abstract superclass: Base.pm. The hierarchy of Chart is shown in Figure 1. Therefore you have to

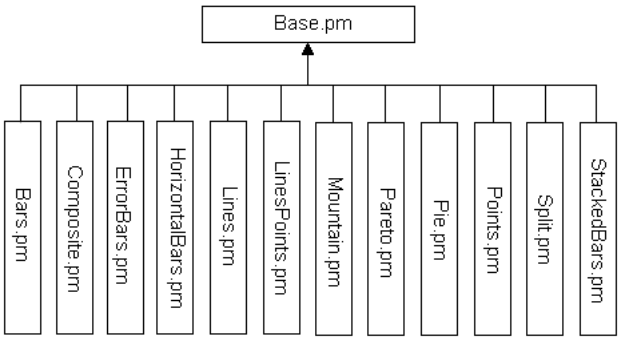

Figure 1: The hierarchy of chart

create an **instance of one of the subclasses** , to get a chart object.
All of the methods and most of the options chart provides, are implemented in Base. But the drawing of the graph itself happens in the respective subclass. Figure 2 shows the Elements of an chart object.
The graph area in the middle is draw by the subclass, all other Elements are

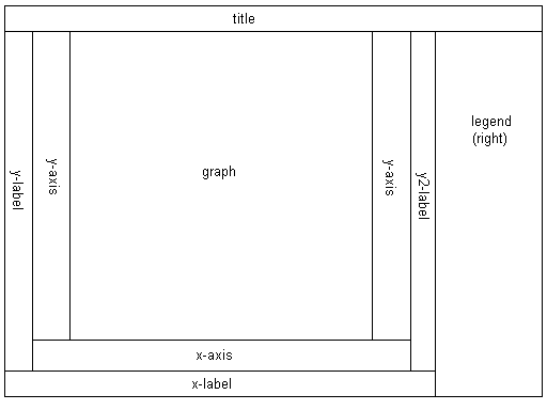

Figure 2: Elements of a chart

drawn by Base. But some classes don‘t need all of this elements or need special elements. Then those elements had to be over written in the respective class. For Example the class Pie doesn’t need axes, so the methods for drawing the axes in Base.pm are over written by methods in Pie.pm, which draw nothing.

Furthermore the legend in a pie chart are a little bit different. Therefore Pie.pm has own methods for drawing the legends. But these things are managed by Chart. You don't have to attend to it.

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

2 Chart::Base

Name: Chart::Base

File: Base.pm

Requires: GD, Carp, FileHandle

Description: **Base** is the **abstract superclass** of the modules: Bars, Composite, ErrorBars, HorizontalBars, Lines, LinesPoints, Mountain, Pareto, Pie, Points, Split, StackedBars
The class Base provides all public methods and most of the attributs of a chart object.

Constructor: An instance of a chart object can be created with the constructor new():
$obj = Chart:: *Type* **->new();**
$obj = Chart:: *Type* **->new(** *width* **,** *height* **);**

Type means the type of chart it returns, i.e. Chart::Bars returns a chart with bars.
If **new** has no arguments, the constructor returns an object with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns a chart object with the desired size.

Methods:

$obj->add_dataset(@array);
$obj->add_dataset(\@array_ref);
Adds a dataset to the object. The parameter is an array or a reference to an array. Generally the first added array are interpreted by chart as the x-tick labels. The following arrays should include the data points. For example if the first call with an bars object is

$obj->add_dataset('Harry', 'Sally'); and the second call is
$obj->add_dataset(5, 8);

then chart will draw a picture with two bars and label them with Harry and Sally.

Some modules handle it a little bit different. Look at the respective description of the module to get more information.
There are also differences if you want to use the **xy_plot** option, to create a xy-graph.

$obj->add_pt(@array);

$obj->add_pt(\@array_ref);
This is another method to add data to a chart object. An 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.
For example

$obj->add_pt('Harry', 5);
$obj->add_pt('Sally', 8);

would create the same graph as the example for add_dataset.

$obj->add_datafile("file", *type* **);**
$obj->add_datafile($filehandle, *type* **);**
This method adds a complete data file to the chart object.
Type can be 'set' or 'pt'. If the parameter is 'set' then one line in the data file has to be a complete data set. The values of the set has to be separated by whitespaces. For Example the file looks like this:

Harry Sally
3 8
2 1

If the parameter is 'pt' the lines of the file have to look like the parameter arrays of the add_pt method. Which means the line includes all values of one data point, also separated by whitespaces. For Example:

Harry 3 2
Sally 8 1

$obj->get_data();
If you want a copy of the data that has been added so far, make a call to the get_data method like so:

$dataref = $obj->get_data();

It returns a reference to an array of references to datasets. For Example, you can get the x-tick labels this way:
@x_labels = @{$dataref->[0]};

$obj->clear_data();
This is the method to remove all data that may have been entered before.

$obj->set(*attribut 1* **=>** *value 1* **, ... ,** *attribute n* **=>** *value n* **);**

$obj->set(%hash);
$obj->set(*attribut 1* **,** *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.
For Example

$obj->set('title' => 'The title of the image');

would set the title. The same job would do:

%hash = ('title' => 'The title of the image');
$obj->set(%hash);

$obj->png("file");
$obj->png($filehandle);
$obj->png(FILEHANDLE);
$obj->png("file", \@data);
This method creates the png file. The file parameter can be, the file name, a reference to a filehandle or a filehandle itself. If the file doesn't exist, chart will create a file for you. If there is already a file, chart will overwrite this file.
You can also add the data to the chart object in the png method. The @data array should contain references to arrays of data, with the first array reference pointing to an array with x-tick labels. @data could look like this:

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

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

$obj->jpeg("file");
$obj->jpeg($filehandle);
$obj->jpeg(FILEHANDLE);
$obj->jpeg("file", \@data);
These are the methods to create jpeg files. They work similar like 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 (ie. with a <img scr=image.pl>HTML tag).

$obj->imagemap_dump();
Chart can also return the pixel positioning information so that you can create image maps from the files Chart generates. Simply set the 'imagemap' option to 'true' before you generate the file, then 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, you will be passed references to arrays of pixel information. For Bars, HorizontalBars, Pareto and StackedBars charts, the arrays will contain two x-y pairs (specifying the upper left and the lower right corner of the bar), like so

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

For Lines, Points, LinesPoints and Split, the arrays will contain a single xy-pair (specifying the center of the point), like so

($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 as the (0,0) point, so positives y values are measured from the top of the png/jpeg, not the bottom. Second, these values will mostly contain long decimal values. GD, of course, has to truncate these to single pixel values. In a worst-case scenario, this will result 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 0th dataset will be empty, since that's the place in the @data array for the data point labels.

Attributes/Options: These are the options which have effects on all types of chart:

'transparent' Makes the background of the image transparent if set to 'true'. Useful for making web page images. It doesn't 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 png/jpeg. Defaults to 10.

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

'text_space' Sets the amount of space left on the sides of text, to make it more readable. Defaults to 2.

'title' Tells Chart what to use for the title of the graph. If empty, no title is drawn. It recognizes '\n' as a newline, and acts accordingly. Remember, if you want to use normal quotation marks instead of single quotation marks then you have to quote "'\\n"'. Default is empty.

'sub_title' Writes a sub-title under the title in smaller letters.

'x_label' Tells Chart what to use for the x-axis label. If empty, no label is drawn. Default is empty.

'y_label', 'y_label2' Tells Chart what to use for the y-axis labels. If empty, no label is drawn. Default is empty.

'legend' Specifies the placement of the legend. Valid values are 'left', 'right', 'top', 'bottom'. Setting this to '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. For example,

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

Default is empty, in which case 'Dataset 1', 'Dataset 2', etc. are used as the 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' (staggers the labels vertically), and 'vertical' (the labels are draw upwards). Default is 'normal'.

'min_y_ticks' Sets the minimum number of y_ticks to draw when generating a scale. Default is 6, The minimum is 2.

'max_y_ticks' Sets the maximum number of y_ticks to draw when generating a scale. Default is 100. This limit is used to avoid plotting an unreasonably large number of ticks if non-round values are used for the min_val and max_val.

The value for 'max_y_ticks' should be at least 5 times larger than 'min_y_ticks'.

'max_x_ticks', 'min_x_ticks' Works similar as 'max_y_ticks' and 'min_y_ticks'. Of course, it works only for xy-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 is maybe better to set the option 'precision' at zero. Default: 'false'

'skip_int_ticks' If 'integer_ticks_only' was set to 'true' the labels and ticks at the y-axis will be drawn every nth tick. Of course in HorizontalBars it affects the x-axis. Default to 1, no skipping.

'precision' Sets the number of numerals after the decimal point. Affects in most cases the y-axis. But also the x-axis if 'xy_plot' is set and the labels in a pie chart. Defaults to 3.

'max_val' Sets the maximum y-value on the graph, overriding the normal autoscaling. Does not work for a Split chart. Default is undef.

'min_val' Sets the minimum y-value on the graph, overriding the normal autoscaling. Does not work for a Split chart. Default is undef.

Caution should be used when setting 'max_val' and 'min_val' to floating point or non-round numbers. This is because the scale must start & end on a tick, ticks must have round-number intervals, and include round numbers.

Example: Suppose your dataset has a range of 35-114 units, If you specify them as the 'min_val' & 'max_val', The y_axis will be plot with 80 ticks every 1 unit.. If no 'min_val' & 'max_val', the system will autoscale the range to 30-120 with 10 ticks every 10 units.

If the 'min_val' & 'max_val' are specified to exesive precision, they may be overiden by the system, plotting a maximum 'max_y_ticks' ticks.

'include_zero' If 'true', forces the y-axis to include zero if it is not in the dataset range. Default is 'false'.

In general, it is better to use this, than to set the 'min_val' if that is all you want to achieve.

'skip_x_ticks' Sets the number of x-ticks and x-tick labels to skip. (ie. if 'skip_x_ticks' was set to 4, Chart would draw every 4th 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 I'm counting from the 0th element of the array. (e.g., if 'custom_x_ticks' is assigned [0,3,4], then the 0th, 3rd, and 4th x-ticks will be displayed) It doesn't work for Split, HorizontalBars and Pie.

'f_x_tick' Needs a reference to a function which uses the x-tick labels generated by the @data->[0] as the argument. The result of this function can reformat the labels. For instance

```
$obj -> set ('f_x_tick' => \&formatter );
```

An example for the function formatter: x labels are seconds since an event. The referenced function can transform this seconds to hour, minutes and seconds.

'f_y_tick' The same situation as for 'f_x_tick' but now used 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. For instance,

$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 png)
'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 first y axis label)
'y_label2' (color of the second y axis label)
'grid_lines' (color of the grid lines)
'x_grid_lines' (color of the x grid lines - for x axis ticks)
'y_grid_lines' (color of the y grid lines - for to left y axis ticks)
'y2_grid_lines' (color of the y2 grid lines - for right 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 8 datasets per component. The colors for 'dataset8' through 'dataset15' become the colors for 'dataset0' through 'dataset7' for the second component chart.

'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 you're going to ask for information about the placement of the data for use in creating an image map from the png. This information can be retrieved using the imagemap_dump() method. NB. that 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 right y-axis (the second component chart). Default is undef.

'no_cache' Adds Pragma: no-cache to the http header. Be careful with this one, as Netscape 4.5 is unfriendly with 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: **Bars** is a **subclass** of Chart::Base.
The class Bars creates a chart with bars.

Example:

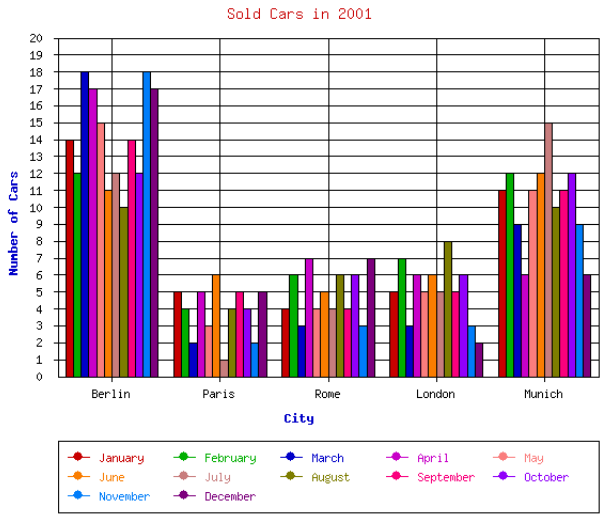

Figure 3: Bars chart

```
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 instance of a bars chart object can be created with the constructor new():
$obj = Chart::Bars->new();
$obj = Chart::Bars->new(*width* , *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available, these special options:

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

'spaced_bars' Leaves space between the groups of bars at each data point when set to 'true'. This just makes it easier to read a bar chart. Default is 'true'.

4 Chart::Composite

Name: Chart::Composite

File: Composite.pm

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

Description: **Composite** is a **subclass** of Chart::Base.
The class Composite creates a two component chart with two types of chart. For example you can create a two component chart with bars and lines. Just set the option 'composite_info'! A composite chart doesn't make sense with all types of chart. But it works pretty good with Lines, Points, LinesPoints and Bars.

Example:

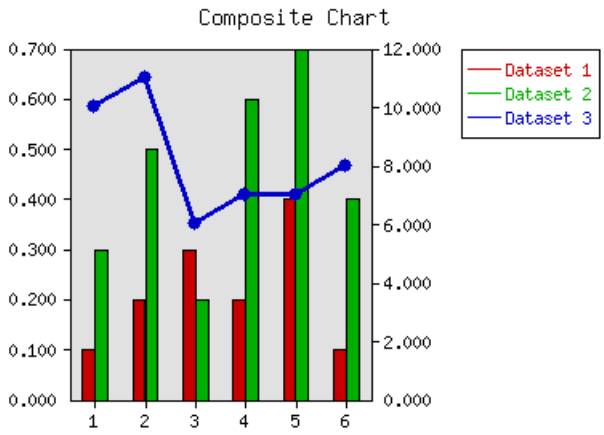

Figure 4: Composite chart

```
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('title' => 'Composite Chart',
        'composite_info' => [ ['Bars', [1,2]],
                              ['LinesPoints', [3]] ] );
$g->set( 'include_zero' => 'true');
```

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

Constructor: An instance of a Composite object can be created with the constructor new():
$obj = Chart::Composite->new();
$obj = Chart::Composite->new(*width* **,** *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available, these special options:

'composite_info' This option is only used for composite charts. It contains the information about 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 (note that the numbering starts at 1, not zero like most of the other numbered things in Chart), and the second two data sets for the line chart. The default is undef.

NB. Chart::Composite can only do two component charts.

'min_val1', 'min_val2' Only for composite charts, these options specify the minimum y-value for the first and second components respectively. Both default to undef.

'max_val1', 'max_val2' Only for composite charts, these options specify the maximum y-value for the first and second components respectively. Both default to undef.

'y_ticks1', 'y_ticks2' The number of y ticks to use on the first and second y-axis on a composite chart. Please note that if you just set the 'y_ticks' option, both axes will use that number of y ticks. Both default to undef.

'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 the composite charts from being too confusing. Default is undef.

5 Chart::ErrorBars

Name: Chart::ErrorBars

File: ErrorBars.pm

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

Description: **ErrorBars** is a **subclass** of Chart::Base.
The class ErrorBars creates a point chart with error bars.
Chart expects the error values within the data array. By use of the add_dataset method the error values are the two next sets after the y values. The first set after the y values has to be a set values for the upper error bars. The next set is an array of the down errors.
If you want to use the same value for the up and down error, then you have to set the 'same_error' option to 'true'. In this case only one set after the y values is interpreted as a set of errors.
Of course, it's also possible to use the add_pt method in a respective way.

Example:

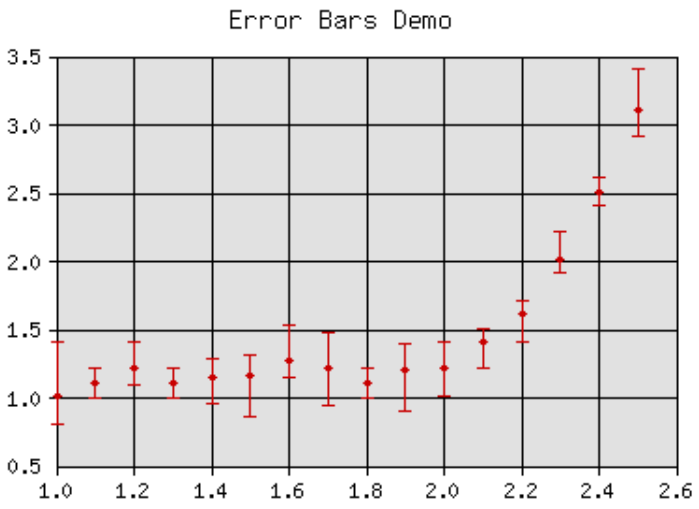

Figure 5: Error bars chart

```
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 up 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 down 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));

$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 instance of a error bars chart object can be created with the constructor new():
$obj = Chart::ErrorBars->new();
$obj = Chart::ErrorBars->new(*width* , *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available these special options:

'same_error' Tells chart that you want to use the same error value of a data point if set to true. Then you have to add just one set of error values. Defaults to 'false'.

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

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

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

'xy_plot' Forces Chart to plot a x-y-graph, which means that the x-axis is also numeric if set to 'true'. Very useful for plots of mathematical functions. Defaults to 'false'.

'sort' Sorts the data of a x-y-graph ascending if set to 'true'. Should be set if the added data isn't sorted. Defaults to 'false'.

6 Chart::HorizontalBars

Name: Chart::HorizontalBars

File: HorizontalBars.pm

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

Description: **HorizontalBars** is a **subclass** of Chart::Base.
The class HorizontalBars creates a chart with bars, that run horizontal.

Example:

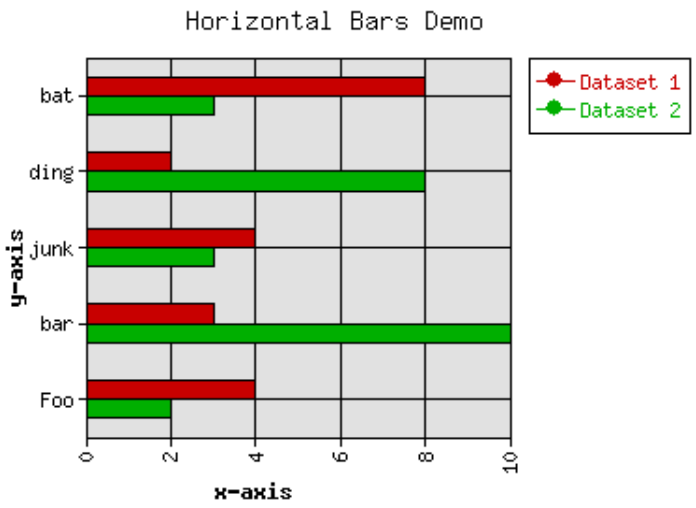

Figure 6: Chart with horizontal bars

```
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 instance of a HorizontalBars object can be created with the constructor new():
$obj = Chart::HorizontalBars->new();
$obj = Chart::HorizontalBars->new(*width* **,** *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available, these special options:

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

'spaced_bars' Leaves space between the groups of bars at each data point when set to 'true'. This just makes it easier to read a bar chart. Default is 'true'.

'skip_y_ticks' Does the same for the y-axis at a HorizontalBars chart as 'skip_x_ticks' does for other charts. Defaults to 1.

7 Chart::Lines

Name: Chart::Lines

File: Lines.pm

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

Description: **Lines** is a **subclass** of Chart::Base.
The class Lines creates a lines chart.

Example:

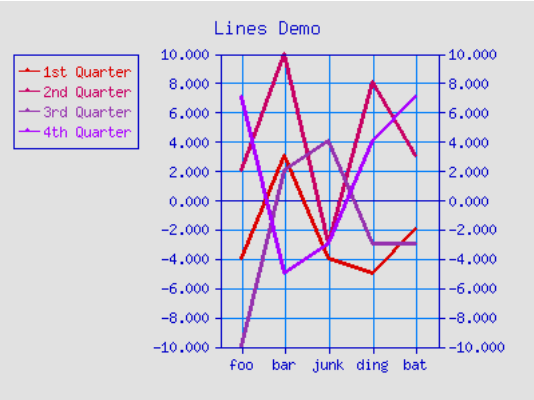

Figure 7: Lines chart

```
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 instance of a lines chart object can be created with the constructor new():
$obj = Chart::Lines->new();
$obj = Chart::Lines->new(*width* **,** *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Special options for this type of chart are:

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

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

'xy_plot' Forces Chart to plot a x-y-graph, which means that the x-axis is also numeric if set to 'true'. Very useful for plots of mathematical functions. Defaults to 'false'.

'sort' Sorts the data of a x-y-graph ascending if set to 'true'. Should be set if the added data isn't sorted. Defaults to 'false'.

8 Chart::LinesPoints

Name: Chart::LinesPoints

File: LinesPoints.pm

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

Description: **LinesPoints** is a **subclass** of Chart::Base.
The class LinesPoints creates a lines chart with points at the ends of the lines.

Example:

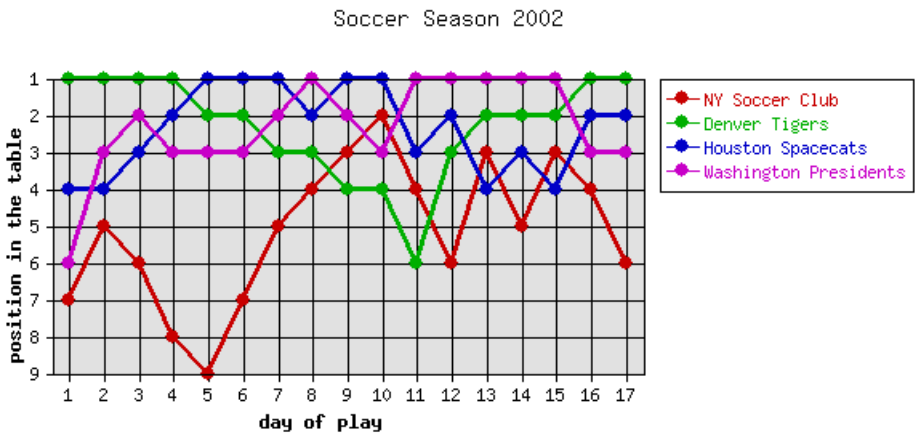

Figure 8: Linespoints chart

```
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 ("Grafiken/d_linesp2.png");

#just a trick, to let the y scale start at the biggest point:
#initiate with negative values, remove the minus sign!
sub formatter {
  my $label = shift;
  $label = substr($label, 1,2);
  return $label;
}
```

Constructor: An instance of a linespoints chart object can be created with the constructor new():
$obj = Chart::LinesPoints->new();
$obj = Chart::LinesPoints->new(*width* **,** *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available these special options:

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

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

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

'xy_plot' Forces Chart to plot a x-y-graph, which means that the x-axis is also numeric if set to 'true'. Very useful for plots of mathematical functions. Defaults to 'false'.

'sort' Sorts the data of a x-y-graph ascending if set to 'true'. Should be set if the added data isn't sorted. Defaults to 'false'.

9 Chart::Mountain

Name: Chart::Mountain

File: Mountain.pm

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

Description: **Mountain** is a **subclass** of Chart::Base.
The class Mountain creates a mountain chart.

Example:

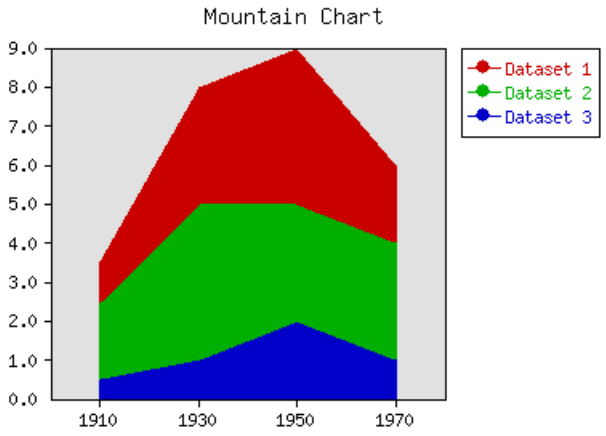

Figure 9: Mountain chart

```
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 instance of a mountain chart object can be created with the constructor new():

$obj = Chart::Mountain->new();
$obj = Chart::Mountain->new(*width* **,** *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available, these special options:

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

10 Chart::Pareto

Name: Chart::Pareto

File: Pareto.pm

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

Description: **Pareto** is a **subclass** of Chart::Base.
The class Pareto creates a pare-to chart. Pareto plots only one dataset and its labels.

Example:

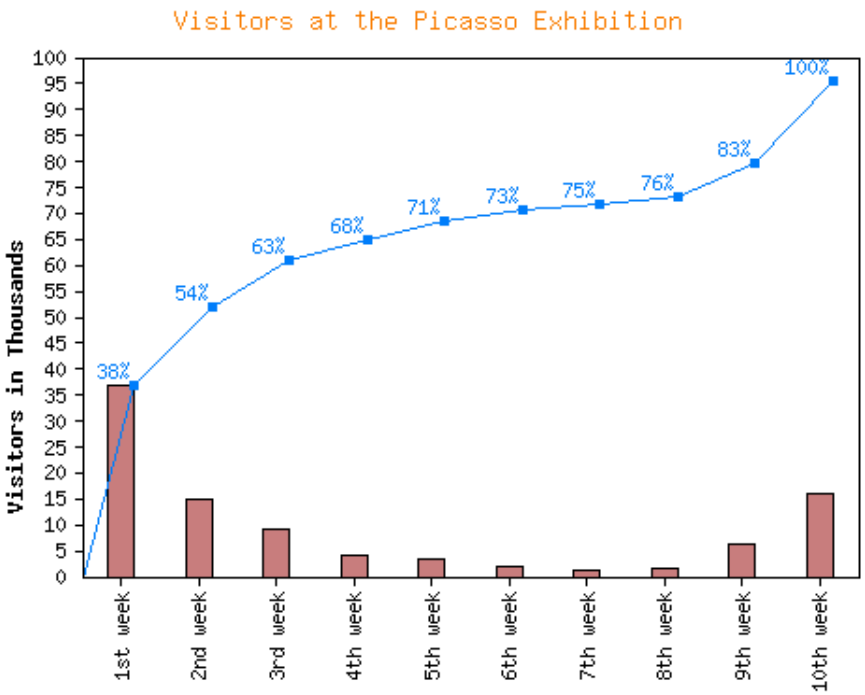

Figure 10: Pare-to chart

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

Constructor: An instance of a pare-to chart object can be created with the constructor new():
$obj = Chart::Pareto->new();
$obj = Chart::Pareto->new(*width* **,** *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available, these special options:

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

'spaced_bars' Leaves space between bars at each data point when set to 'true'. This just makes it easier to read a bar chart. Default is 'true'.

'sort' Sorts the data descending if set to 'true'. Defaults to 'false'.

11 Chart::Pie

Name: Chart::Pie

File: Pie.pm

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

Description: **Pie** is a **subclass** of Chart::Base.
The class Pie creates a pie chart. The first added set are the labels. The second set are the values.

Example:

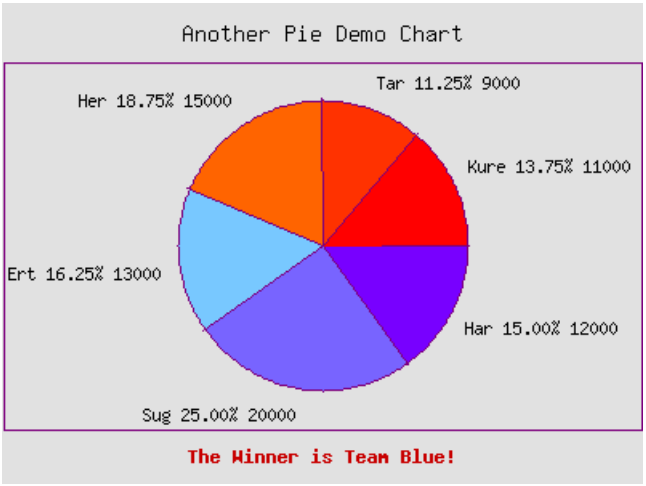

Figure 11: Pie chart

```
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 instance of a pie chart object can be created with the constructor new():
$obj = Chart::Pie->new();
$obj = Chart::Pie->new(*width* , *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available, these special options:

'label_values' Tells the pie chart what labels to draw beside the pie. Valid values are 'percent', 'value', 'both' and 'none'. Defaults to 'percent'.

'legend_label_values' Tells the pie chart what labels to draw in the legend. Valid values are 'percent', 'value', 'both' and 'none'. Defaults to 'value'.

12 Chart::Points

Name: Chart::Points

File: Points.pm

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

Description: **Points** is a **subclass** of Chart::Base.
The class Points creates a point chart.

Example:

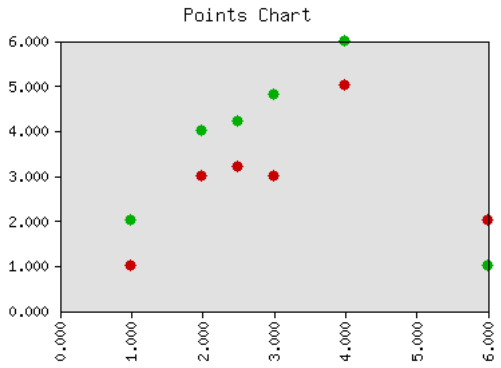

Figure 12: Points chart

```
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 instance of a points chart object can be created with the constructor new():
$obj = Chart::Points->new();
$obj = Chart::Points->new(*width* **,** *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available these special options:

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

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

'sort' Sorts the data of a x-y-graph ascending if set to 'true'. Should be set if the added data isn't sorted. Defaults to 'false'.

'xy_plot' Forces Chart to plot a x-y-graph, which means that the x-axis is also numeric if set to 'true'. Very useful for plots of mathematical functions. Defaults to 'false'.

13 Chart::Split

Name: Chart::Split

File: Split.pm

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

Description: **Split** is a **subclass** of Chart::Base.
The class Split creates a lines chart. Split makes always an xy-plot, which means that both axes are numeric. The x-axis will be split in several parts of a same interval (option 'interval' has to be set!). These intervals will be drawn one upon the other. The top interval starts at the start point, which has to be set by the programmer (option 'start').
The first passed dataset are the x coordinates. The following added sets are the y coordinates of the sets.
Split draws only positive x-coordinates.
The y-axis is a numbering of the intervals.
The Split module is useful if you have a lot of data points to plot.

Example:

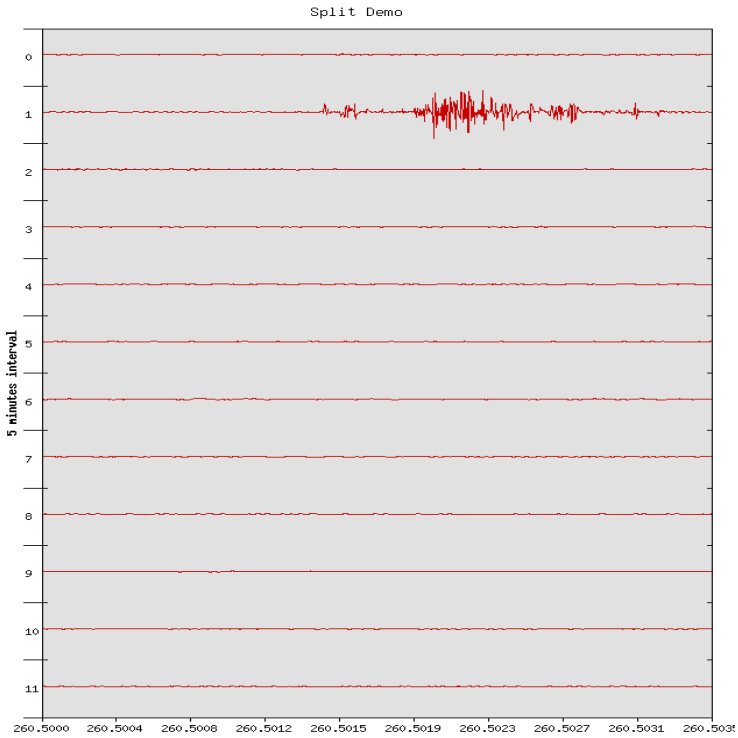

Figure 13: Split chart

```
use Chart::Split;

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

#get the data that are in a file and push them in arrays
open( FILE , "data.dat") or die 'Can't open the data file!\n';
while (defined ($line = <FILE>) ) {
  ($x, $y,) = unpack("a11 x1 a8" , $line);
   push (@y, $y);
   push (@x, $x);
}
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 instance of a split chart object can be created with the constructor new():
$obj = Chart::Split->new();
$obj = Chart::Split->new(*width* , *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available, these special options:

'start' Required value for a split chart. If the x coordinate of the first data point is zero, you should set start to zero. Sets the start value of the first interval. Defaults to undef.

'interval' Required value of a split chart. Sets the interval of one line to plot. Defaults to undef.

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

'scale' Every y-value of a split chart will be multiplied with that value, but the scale won't change. Which means that split allows to overdraw certain rows! Only useful if you want to give prominence to the maximal amplitudes of the data. Defaults to 1.

'sort' Sorts the data ascending if set to 'true'. Should be set if the added data isn't sorted. Defaults to 'false'.

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

14 Chart::StackedBars

Name: Chart::StackedBars

File: StackedBars.pm

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

Description: **StackedBars** is a **subclass** of Chart::Base. The class StackedBars creates a chart with stacked bars.

Example:

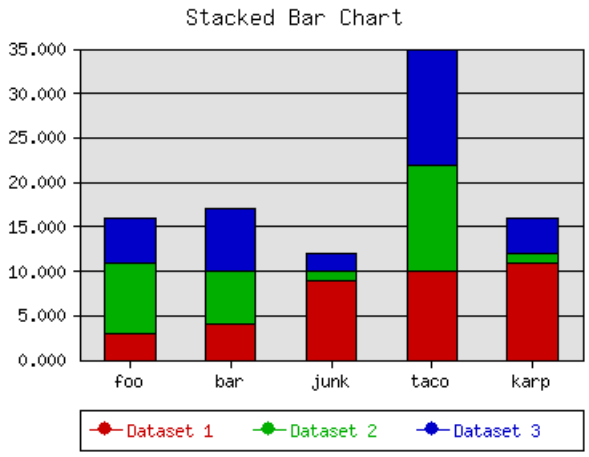

Figure 14: Chart with stacked bars

```
use Chart::StackedBars;

$g = Chart::StackedBars->new;

$g->add_dataset ('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 ("Grafiken/stackedbars.png");
```

Constructor: An instance of a stacked bars object can be created with the constructor new():

$obj = Chart::StackedBars->new();
$obj = Chart::StackedBars->new(*width* **,** *height* **);**

If **new** has no arguments, the constructor returns an image with the size 300x400 pixels. If new has two arguments *width* and *height* , it returns an image with the desired size.

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

Attributes/Options: All universally valid options, see page 8. Also available, these special options:

'y_axes' Tells chart where to place the y-axis. Valid values are 'left', 'right' and 'both'. Defaults to 'left'.

'spaced_bars' Leaves space between the groups of bars at each data point when set to 'true'. This just makes it easier to read a bar chart. Default is 'true'.

List of Figures