1 Introduction

1.1 Version

This manual covers version 1.1 of JpGraph. A 2D graph plotting library for PHP 4.02 and above.
Note that this library will not work with versions prior to PHP 4.02 due to extension in the object model that is used in this library.

1.2 Features

This is a truly OO graph library which makes it easy to both draw a "quick and dirty" graph with a minimum of code and quite complex graphs which requires a very fine grain of control. The library tries to assign sensible default values for most parameters hence making the learning curve quite flat since for most of the time very few commands is required to draw graphs with a pleasing esthetical look.

Some highlights of available features are

- Flexible scales, supports text-lin, text-log, lin-lin, lin-log, log-lin and log-log
- Supports both PNG, GIF and JPG graphic formats. Note that the available formats are dependent on the specific PHP installation where the library is used.
- Supports caching of generated graphs to lessen burden of a HTTP server.
- Intelligent autoscaling which gravitates towards esthetical values, i.e. multiples of 2:s and 5:s
- Fully supports manual scaling, with fine grain control of position of ticks
- User specified grace for autoscaling
- Supports up to two different y-scale, it is possible to have different left and right y-scale and add plots to both
- Supports, line-plots, filled line-plots, accumulated line-plots, bar plots, accumulated bar plots, grouped bar plots, error plots, line error plots, scatter plots, impuls plots, spider (a.k.a. Web) plots and pie charts.
- Supports unlimited number of plots in each graph, makes it easy to compose complex graph which consists of several plot types
- User specified position of axis
- Supports color gradient fill in seven styles
- Designed as a flexible OO framework which makes it easy to add new types of plots
- Supports automatic legend generation
- Supports both vertical and horizontal grids
- Supports anti-alising of lines
- Supports rotation of linear graphs
- More then 400 named colors
- Designed modularly – you don't have to include code which isn't used

In addition to these high level features the library has been designed to be orthogonal and very coherent in its' naming convention. For example, to specify color each object (i.e. axis, grids, texts, titles etc) within the graph implements the method SetColor() with the same signature.

1.3 Planned future addition

All the following features, which have not been marked as tentatively, will be added. The timeframe for these versions are:

?? Version 1.3 Q2 2001
?? Version 2.0 Q4 2001

No time frames have been determined for version 2.x and above. If you like these timeframes to move forward get involved in the development. Changes, bugfixes and additions are always welcome.

For the latest upate on planned future version see the web-site for JpGraph at www.aditus.nu/jpgraph/

1.4 Known bugs and omissions

?? Rounding errors. Some combination of image size and scale span might on some points display a one-pixel difference between scale labels and plot points. This is for all practical purposes not visually detectable. A walkthrough of all computation routines within the library will be necessary to assure that they are all rounded/truncated the exact same way.

1.5 Acknowledgements

The idea for writing this library grew out of my own needs for a high quality graph drawing library for PHP4. Before reinventing the wheel I searched the net to see if there where anything already available that would meet my needs. When searching I found three other PHP graph plotting libraries:

1. "chart 0.3" http://quimby.gnus.org/circus/chart/chart-0.3.tar.gz, by Lars Magne Ingebrigtsen
2. "ykcee.php", http://ykcee.sourceforge.net
3. "phplot.php", http://www.phplot.com

All these libraries implements some fine graphic features but unfortunately none of those completely fulfilled my needs either for available functionality (for example none of these supported both two Y-scales, auto-scaling, and logarithmic scales), or general flexibility, I especially needed the option of two Y-scales, which none of the above packages supported. My own preference for design was closest to "chart 0.3" so I started by fixing some small bugs in that package and adding some new features. However I soon realized that to add all the features and flexibility I wanted to "chart 0.3" it would require a complete rewrite since the original design wasn't flexible enough, especially adding a second Y-scale would require a more flexible OO architecture.

Since at the time I was also interested in giving PHP a try to see how well it would support a fully object oriented design so I started designing this library. The library is completely written from scratch but I have taken some ideas, especially the caching feature from "chart 0.3" and implemented this.

I'm therefore thankful for those ideas. Any bugs and faults within the code are completely my own.

1.6 Implementing an OO library in PHP4

In terms of OO support PHP is still at loss to Java, Eiffel or C++ but since it always been my view that OO design is more a state of mind then of implementation details it is still possible to arrive with a decent OO design even in PHP. One of the big obstacles is probably the very different assigning semantics used by PHP as compared to other OO languages since it is always copies of the object that is passed around by default and not references. This took some time for me personally to get used to (giving my own background in OO design in Java, Eiffel and C++).

There is also a bug (present in all versions <=4.04pl1) that makes it impossible to use a reference to the '$this' pointer in the constructor. This has an easy workaround by adding an extra method, say Init(), which is called immediately after the constructor and may safely use a reference to '$this' pointer.

As an example of JpGraph's OO features this is, to my knowledge, the only piece of PHP code to fully implement the Observer pattern.

1.7 Getting the latest version

The latest version of jpgraph can always be found on http://www.aditus.nu/jpgraph/
The current version as of this writing is 1.2

Note. I keep a very simple version scheme to avoid the version number inflation that seems to be going on, this means
1.x -> 1.x.1 Bug fix release for version 1.x
1.x -> 1.(x+1) Added functionality. Keeping backwards compatibility
1.x -> 2.0 Substantially new functionality which might break backward compatibility

1.8 Reporting bugs and suggested improvements

Bug reports and suggestions are always welcome. I only ask you to make sure that you read the requirements before submitting bugs and making sure you have an up to date version of PHP4, the necessary graphic libraries etc. I will unfortunately not be able to answer standard OO or PHP4 questions.

Please note that this library will **not** work with versions prior to PHP4.02.

Bug reports and suggestions should be sent to Jpgraph@aditus.nu

1.9 Software License

JpGraph 1.2 is released under GPL (GNU Public License 2.0)

1 Quick start on how to use JpGraph

1.1 Generating images with PHP

As a general rule each PHP which generates an image must be specified in a separate file which is then called on in an <IMG> tag reference. For example, the following HTML excerpt includes the image generated by the PHP script in "fig1.php"

```
. . .
<img src="fig1.php" border=0 align=center width=300 height=200>
. . .
```

The library will automatically generate the necessary headers to be sent to the browser to correctly recognize the data stream as an image of either PNG/GIF/JPEG format.

To get access to the library you will need to include at least two files, the base library and one or more of the plot extensions. So for example if you want to do line plots the top of your PHP file must have the lines:

```
<?php
include ("jpgraph.php");
include ("jpgraph_line.php");
. . .
// Code that uses the jpgraph library
. . .
?>
```

Note: You might also use the PHP directive requires(). The difference is subtle in that include will only include the code if the include statement is actually excuted. While require() will always be replaced by the file specified. See PHP documentation for further explanation. For most practical purposes they are identical.

1.2 A first example

The following simple example draws a line graph consisting of 10 Y-values

```
<?php
include ("jpgraph.php");
include ("jpgraph_line.php");

$ydata = array(11,3,8,12,5,1,9,13,5,7);

// Create the graph. These two calls are always required
$graph = new Graph(300,200);
$graph->SetScale("textlin");

// Create the linear plot
$lineplot=new LinePlot($ydata);

// Add the plot to the graph
$graph->Add($lineplot);

// Display the graph
$graph->Stroke();
?>
```

Figure 1. PHP script for simple graph. (example1.php)

This script will generate the following graph

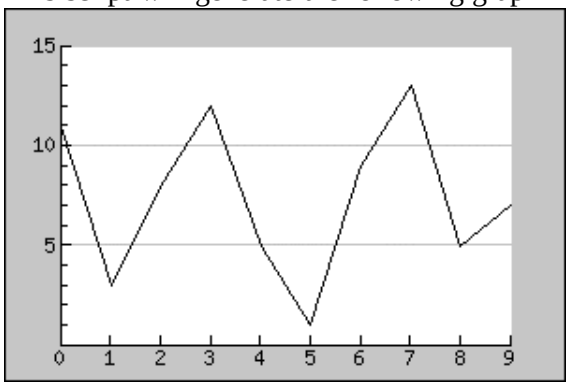


Figure 2. The simplest possible JpGraph

You might note a few things:

?? Both the X and Y axis have been automatically scaled. We will later on show how you might control the autoscaling how it determines the number of ticks which is displayed.
?? By default the Y-grid is displayed in a “soft” color
?? By default the image is bordered and the margins are slightly gray.
?? By default the 0 label on the Y-axis is not displayed

This is a perfect fine graph but we want might to add a few things like

?? A title for the graph
?? Title for the axis
?? Increase the margins to account for the title of the axis

To handle this we just need to add a few more lines. (We only show the part of example 1 we changed)

```
. . .

// Increase the margins (left,right,top,bottom)
$graph->img->SetMargin(40,20,20,40);

// Add graph and axis title
$graph->title->Set("Example 2");
$graph->xaxis->title->Set("X-title");
$graph->yaxis->title->Set("Y-title");

. . .
```

Figure 3. Example 2. Adding a graph title and axis title

The graph will now look like this

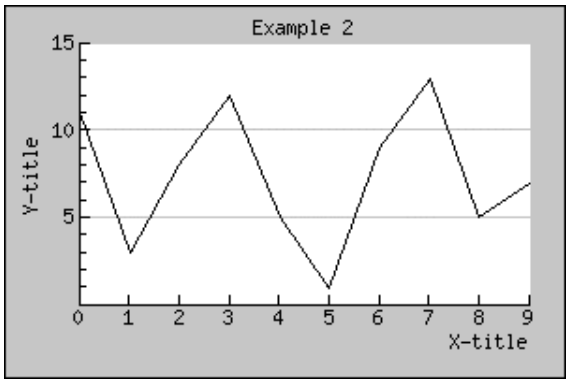


Figure 4. A graph with added titles.

Again a couple of things should be noted

?? A default font and size is used for the text
?? The default position for the title of the graph is to be centered at the top
?? The default position for the title of the x-axis is the far right and for the y-axis centered and rotated in a 90^0 angle.

A nice change would now be to have all the titles in a bold font and the line plot a little bit thicker and in blue color. Let's do that

```
. . .
$graph->title->SetFont(FONT1_BOLD);
$graph->yaxis->title->SetFont(FONT1_BOLD);
$graph->xaxis->title->SetFont(FONT1_BOLD);

$lineplot->SetColor("blue");
$lineplot->SetWeight(2);  // Two pixel wide
. . .
```

You might by now have noticed that you apply the same methods to different objects within the graph. This is something that is a red line in this OO library. So far you have seen that a graph has a title, an x-axis, an y-axis etc. In the reference section a complete list of available objects and methods are listed. In almost most cases you will learn about a new method, like SetColor(), which you could then apply to other objects. As an example, let's make the Y-axis red. As you might guess this is accomplished by the line:

```
. . .
$graph->yaxis->SetColor("red");
. . .
```

Or perhaps making the Y-axis a little bit thicker (note that this will also affect the Y-grid thickness)

```
. . .
$graph->yaxis->SetWeight(2);
. . .
```

As a final touch on this first example lets add a shadow to the frame surrounding the image. By default this is switched off. This is done by adding the line

```
. . .
$graph->SetShadow();
. . .
```

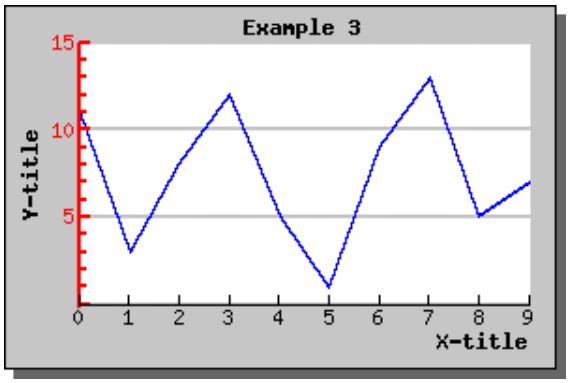


Figure 5. The final appearance of the graph after changing the properties of the Y-axis and adding a shadow to the frame.

1.3 Adding plot marks to line graphs

It might sometimes be desirable to highlight the data-points with marks in the intersection between the given x and Y-coordinates. This is accomplished by specifying the wanted plot mark type for the "mark" property of the line graph.

As of JpGraph 1.0 the following marks are available:

?? MARK_SQUARE, A filled square
?? MARK_UTRIANGLE, A upward pointing triangle
?? MARK_DTRIANGLE, A downward pointing triangle
?? MARK_DIAMOND, A diamond shape
?? MARK_CIRCLE, A **non-filled** circle.

Let's add diamond marks to the graph in example 3 above. This is accomplished by adding the single line

```
$lineplot->mark->SetType(MARK_DIAMOND);
```

The resulting graph is shown below

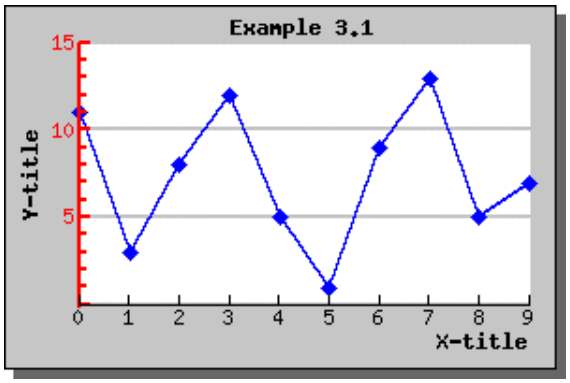


Figure 6. Line graph with plot marks in the intersections.

The colors of the marks will, if you don't specify them explicitly, follow the line color. Please note that if you want different colors for the marks and the line the call to SetColor() for the marks must be done after the call to the line since the marks color will always be reset to the lines color when you set the line.

1.4 Adding several plots to the same graph

What if we want to add a second plot to the graph we just produced? Well, this is quite straightforward and just requires two step:

1. Create the second plot
2. Add it to the graph

To create the second plot we need some data (we could of course use the same data as in example 1 but then we wouldn't be able to see the new plot!)

The following lines show how to create the new plot and add it to the graph (we only show the new lines – not the full script)

```
. . .
$ydata2 = array(1,19,15,7,22,14,5,9,21,13);
$lineplot2=new LinePlot($ydata2);
$lineplot2->SetColor("orange");
$lineplot2->SetWeight(2);
. . .
$graph->Add($lineplot2);
. . .
```

The graph resulting from these changes will look like

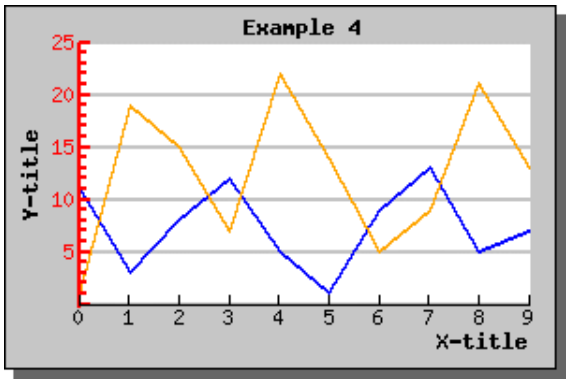


Figure 7. Adding several plots to the same graph.

You should now note that

?? The Y-scale has changed to accommodate the larger range of Y-values for the second graph.

?? If you add several plots to the same graph they should contain the same number of data points. This is not a requirement the graph will be automatically scaled to accommodate the plot with the largest number of points but it will not look very good since one of the plot end in the middle of the graph.

1.5 Adding a second Y-scale

As you saw in the preceding example you could add multiple plots to the same graph and Y-axis. However what if the two plots you want to display in the graph has very different ranges. One might for example have Y-values like above but the other might have Y-values in the 100:s. Even though it is perfectly possible to add them as above the graph with the smallest values will have a very low dynamic range since the scale must accomplish the bigger dynamic range of the second plot.

The solution to this is to use a second Y-axis with a different scale and add the second plot to this Y-axis instead. Let's take a look at how that is accomplished.

First we need a nice data array with large values

```
. . .
$y2data = array(354,200,265,99,111,91,198,225,293,251);
. . .
```

Then we need to add a second linear Y axis to the graph

```
. . .
$graph->SetY2Scale("lin");
. . .
```

and finally we create a new line plot and add that to the second Y-axis. Note that we here use a new method AddY2() since we want this plot to be added to the second Y-axis (version 1.0 of jpgraph only supports two different Y-axis and Y-scales.)

```
. . .
$lineplot2=new LinePlot($y2data);
$graph->AddY2($lineplot2);
. . .
```

To make the graph a little bit more esthetical pleasing we use different colors for the different plots and let the two different Y-axis get the same colors as the plots.

```
. . .
$lineplot2->SetColor("orange");
$lineplot2->SetWeight(2);
$graph->SetColor("orange");
. . .
```

The final graph will now look like this:

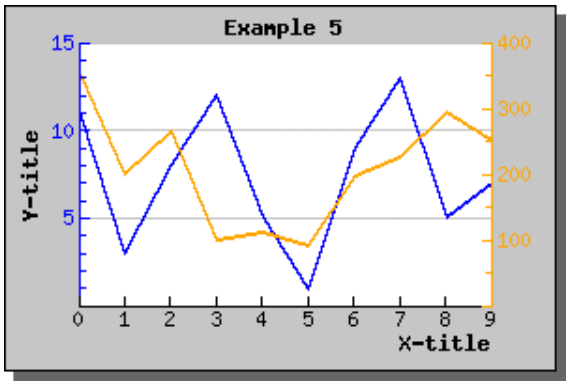


Figure 8. A graph with two different Y-axis and a plot for each Y-axis

1.6 Adding a legend to the plot

To know what different plots stand for it is custom to add a legend to the graph that explains what each plot represents. This is very easy to do. We only need to specify the legend text for each plot and most likely where we want the legend to be displayed. Let's first see what we get with the default settings so we just add the text we want associated with each plot, let's say "Plot 1" and "Plot 2". This is done by the following two added lines

```
. . .
$lineplot->Legend->Set("Plot 1");
$lineplot2->Legend->Set("Plot 2");
. . .
```

If we do this we get a resulting graph as

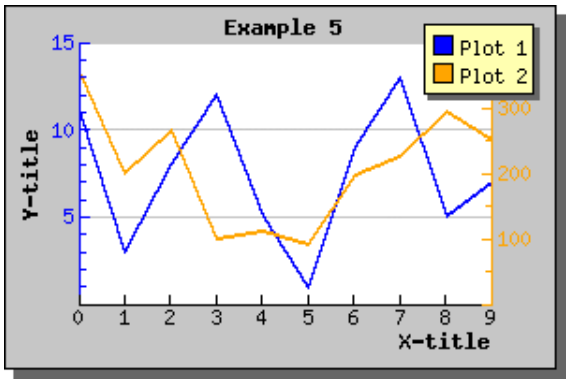


Figure 9. Graph with a legend

As you can see the legend gets automatically sized depending on how many plots there are that have legend texts to display. By default it is placed with it's top right corner close to the upper right edge of the image. Depending on the image you might want to adjust this or you might want to add a larger margin which is big enough to accompany the legend. Let's do both. First we increase the right margin and then we place the legend so that it is roughly centred. We will also enlarge the overall image so the plot area doesn't get to squeezed. (We don't show the new values for the margins just the new method to position the legend.)

```
. . .
$graph->legend->Pos(0.05,0.5,"right","center");
. . .
```

This will then generate the following graph

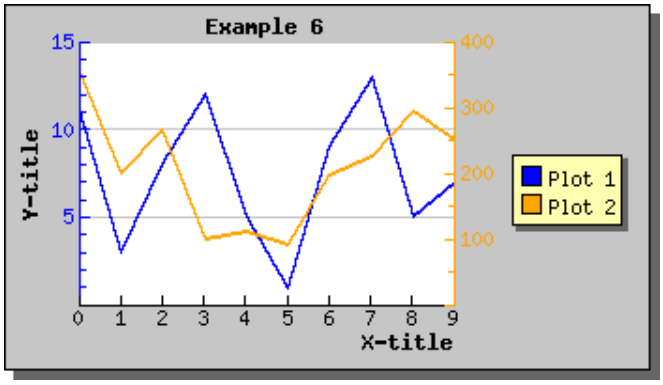


Figure 10. Graph with adjusted legend

The above method deserves some explanation since it might not be obvious. You specify the position as percentage of the overall width and height of the entire image. This makes it possible for you to resize the image within disturbing the relative position of the legend. We will later see that the same method is just to place arbitrary text in the image.

To give added flexibility one must also specify to what edge of the legend the position given should be relative to. In the example above we have specified "right" edge on

the legend for the for the X-axis meaning that the distance between the right edge of the legend and the right edge of the image is 5% of the images entire width.

Allowed values for the X-position are [“left” ,”center”, “right”], and for the Y-position [“top”,”center”].

By default the text in the legend are stacked on top of each other. The other possibility to layout the legend is horizontally, i.e. the text is place horizontally after each other. You decide which way you want to have the legend by a call to the method “SetLayout($layout)” allowed values for $layout are

?? LEGEND_HOR
?? LEGEND_VERT

Lets illustrate this by changing the legend in the preceding example to use horizontal layout instead and place the legend at the bottom of the image. This is accomplished by the lines

```
. . .
$graph->img->SetMargin(40,40,20,70);
. . .
$graph->legend->SetLayout(LEGEND_HOR);
$graph->legend->Pos(0.5,0.85,"center","center");
. . .
```

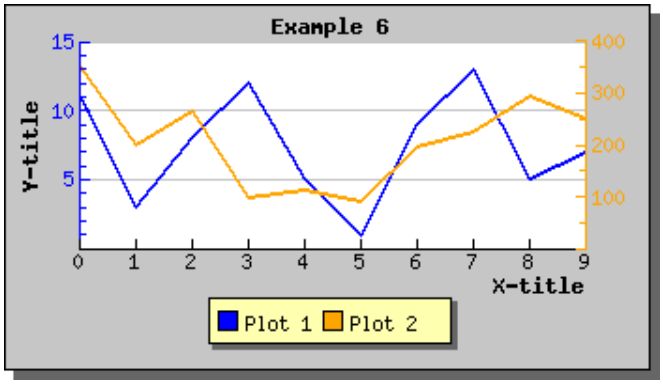


Figure 11. Plot with alternative layout of legend.

1.7 Using the “Step style” to render line plots

Step style refers to an alternate way of rendering line plots by not drawin a direct line between two adjacent points but rather draw two segements. The first segment being a horizontal line to the next X-value and then a vertical line from that point to the crrect Y-value. This is perhaps easier demonstrated by an example.

You specify that you want the plot to ber rendered with this style by calling the method SetStepStyle() as

```
. . .
$lineplot1->SetStepStyle()
. . .
```

For example, the following graph can be generated:

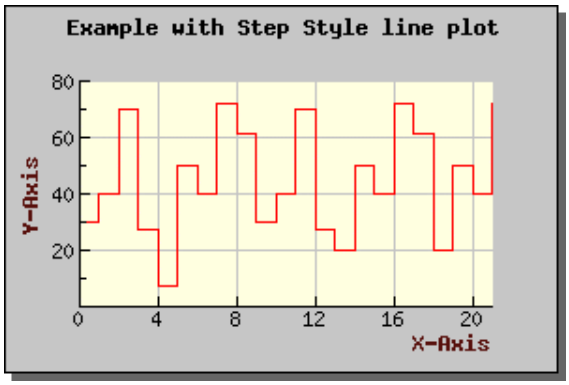


Figure 12. Example of lineplot with the "step style"

1.8 Using a logarithmic scale

Using a logarithmic scale requires you to include the logarithmic add on module in "jpgraph_log.php". So you must have the line include("jpgraph_log.php") on the top of your code. To Illustrate how to use a logarithmic scale let's make the right Y scale in the previous example a logarithmic scale. This is done by the line

```
. . .
$graph->SetY2Scale("log");
. . .
```

The resulting graph will then be as illustrated below

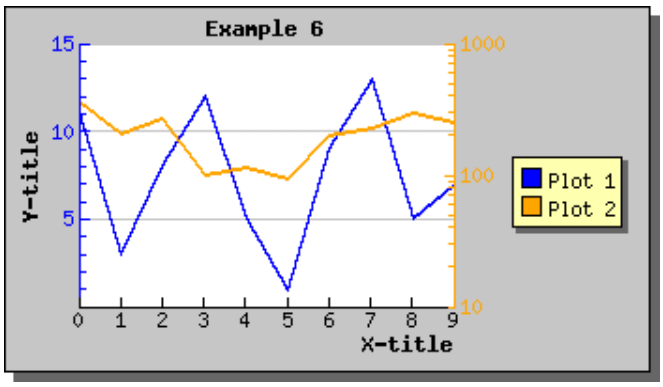


Figure 13. Graph with a logarithmic Y2 scale.

If you also wanted the normal (left) Y scale to be logarithmic you would have had to change the SetScale() method call to

```
. . .
$graph->SetScale("textlog");
. . .
```

1.9 Using different combination of scales

As you saw in the previous example it is possible to use different types of scales. The supported types are

?? Linear scale. Both X and Y axis
?? Logarithmic scale. Both X and Y axis
?? Text scale. Only on X axis

Any combination of these may be used. Linear and logarithmic scales are pretty straightforward. The text scale might deserve some explanation. The easiest way to think of the text scale is as a linear scale consisting of only natural numbers, i.e. 0,1,2,3,4,… . This scale is used when you just have a number of Y-values you want to plot in a consecutive order and don't care about the X-values. For the above example it will also work fine to use a linear X-scale (try it!). However, the scale is now treated as consisting or real numbers so the autoscaling, depending on the size of the image an the number of data points, might decide to display other labels then the natural numbers., i.e. a label might be 2.5 say. This is not going to happen if you use a text scale.

If no X-scale is given the whole numbers in consecutive order will be used as X-coordinates of the supplied Y-points as displayed in all the previous examples.

To specify which combination of X and Y scales you want to use a parameter is passed in the SetScale() method of the graph. The following values are allowed

?? "linlin" Linear X, Linear Y
?? "linlog" Linear X, Log Y
?? "textlin" Text X, Linear Y
?? "textog" Text X, Log Y
?? "loglin" Log X, Linear Y
?? "loglog" Log x, Log Y

So for example to specify a Text X scale and Log Y scale you will call

```
$graph->SetScale("textlog");
```

To specify the Y2 axis you used use "half" of the parameter string, i.e to specify a linear Y2 scale you call

```
$graph->SetY2Scale("lin")
```

Note. The behaviour of specifying "Text" for a Y-scale is undefined and might even blow up your server…

Specifying a log scale for the normal Y-axis will then generate the following image

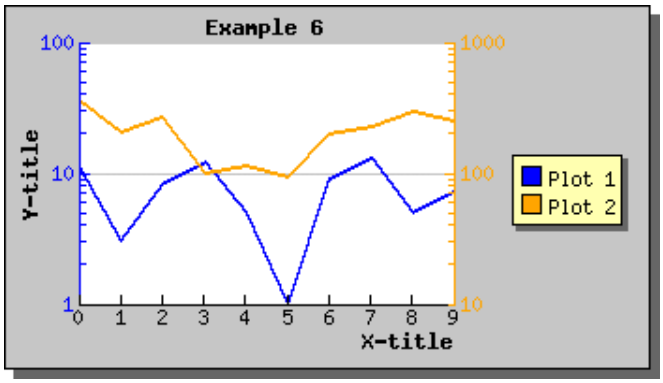


Figure 14. Using different Y and Y2 scales.

1.10 Adding more gridlines to the plot

By default only the Y-axis have a grid and then only on major ticks, i.e. ticks which have a label. It is of course possible to change this. Both the X , Y and Y2 can have a grid. It is also possible to let the gridlines also be drawn on the minor tick marks, i.e. ticks without a label. Lets see how we can apply this to the graph above.

The grid is modified by accessing the xgrid (or ygrid) component of the graph. So to disply minor grid lines for the Y graph we make the call

```
$graph->ygrid->Show(true,true);
```

The first parameter determines if the grid should be displayed at all and the second parameter determines whether or not the minor grid lines should be displayed.

If you instead wanted the gridlines to be displayed for the Y2 axis instead you would call

```
$graph->y2grid->Show(true,true);
```

Note. In general it is not a good idea to display both the Y and Y2 gridlines since the resulting image becomes difficult to read for a viewer.

We can also enable the X-gridlines with the call

```
$graph->xgrid->Show(true,false);
```

The resulting image will now look like

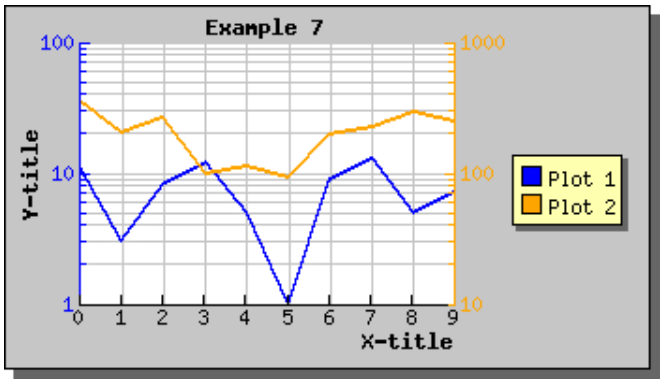


Figure 15. Graph with both X and Y gridlines.

Here we might show a nice feature of jpgraph. Since the Y (and Y2) scales first label (1 and 10) is quite close to the X-labels we might want to not display the first tick label. This can be done with a call
To the method SupressFirst() on the Tick object in the scale for each axis as

```
$graph->yaxis->scale->ticks->SupressFirst();
$graph->y2axis->scale->ticks->SupressFirst();
```

The graph will now look as

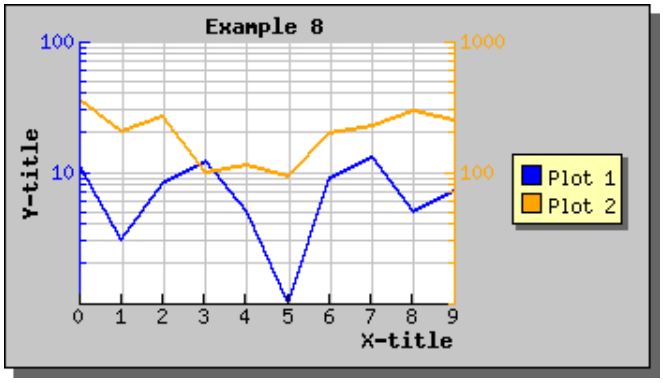


Figure 16. Graph with the first tick marks on the Y-axis suppressed.

1.11 Specifying the labels for X-axis

You might want to have specific labels you want to use for the X-axis when this has been specified as a "text" scale. In the previous example each Y-point might represent a specific measurement for each of the first 10 month. We might then want to display the name of the months as X-scale. This can be done as follows.

```
. . .
$datax=array("Jan","Feb","Mar","Apr","Maj","Jun","July",”aug”,"Sep","Oct");
$graph->xaxis->SetTickLabels($datax);
. . .
```

This will then result in the following graph

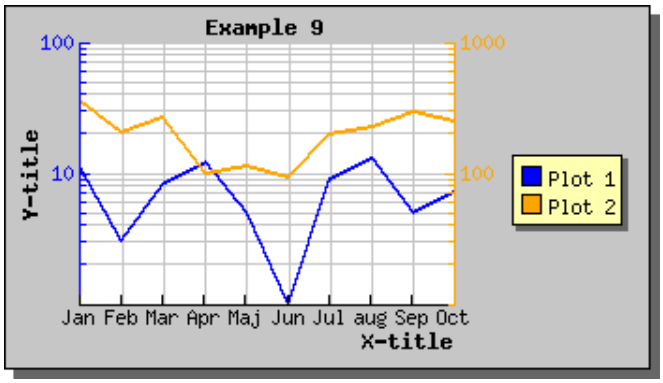


Figure 17. Graph with specified labels for each tick of the X-axis.

It is also perfectly legal to override the default labels for the Y (and Y2) axis in the same way, however there is seldom need for that. Please note that the supplied labels will be applied to each major tick label. If there are insufficient number of supplied labels the non-existent positions will have empty labels.

1.12 Adjusting the ticks on a text scale

As can be seen in the previous example (9) the X-axis is slightly cluttered with the labels very close to each other. We might rectify this by either enlarging the image or just displaying every second tick label on the x-axis.

Specifying that we only want to print every second label on the axis is done by a call to the method
SetTextTicks() as

```
$graph->xaxis->SetTextTicks(2);
```

There is one important thing to remember with this. **The $datax array must be adapted to only contain every second value as well!** My reasoning behind this design decision is that when you have many Y-values, perhaps a couple of hundred, and only wants to have an X label on every 100 you shouldn't have to specify all the labels you don't use.

So now we also change $datax to

```
$datax=array("Jan","Mar","Maj","July","Sep");
```

The resulting graph will now look more esthetical pleasing as

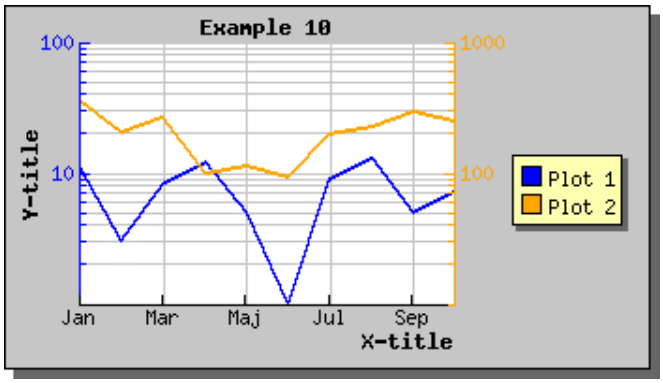


Figure 18. A graph with the X-ticks adjusted to only display every second Major tick.

1.13 Using filled line graphs

Using a filled line plot is not much different from using a normal line plot, in fact the only difference is that you must call the method SetFillColor() on a normal line plot. This will then fill the area under the line graph with the chosen color. So for example plotting a filled "orange" line plot you would add the line

```
. . .
$lineplot->SetFillColor("orange");
. . .
```

If you look closely at a line-plot you will see that the normal line is still there with the color you specified with a previous call to SetColor(). IF you don't wont this bounding line to bee visible just set it to the same color as the fill.

Note 1. If you add multiple filled line plots to one graph make sure you add the one with the highest Y-values first since it will otherwise overwrite the other plots and they will not be visible. Plots are stroked in the order they are added to the graph, so the graph you want front-most must be added last.

Note 2. When using legends with filled line plot the legend will show the fill color and not the bounding line color.

1.14 Using accumulated line graphs

Accumulated line graphs are line graphs that are "stacked" on top of each other. That is the values in the supplied data for the Y-axis is not the absolute value but rather the relative value from graph below. For example if you have two line graphs with three points each, say [3,7,5] and [6,9,7]. The first graph will be plotted on the absolute Y-values [3,7,5] nut the second plot will be plotted at [3+6, 7+9, 5+7], hence the values of the previous graphs will be used as offsets.

You may add any number of graphs together. If you want to use three line plots in an accumulated line plot graph you write the following code

```
. . .
 .// First create the individual plots
$p1 = new LinePlot($datay_1);
$p2 = new LinePlot($datay_2);
$p3 = new LinePlot($datay_3);

// Then add them together to form a accumulated plot
$ap = new AccLinePlot(array($p1,$p2,$p3));

// Add the accumulated line plot to the graph
$graph->Add($ap);
. . .
```

You might of course also fill each line plot by adding the lines

```
. . .
$p1->SetFillColor("red");
$p2->SetFillColor("blue");
$p3->SetFillColor("green");
. . .
```

Using some appropriate data this might then give a graph perhaps like the one showed in the figure below

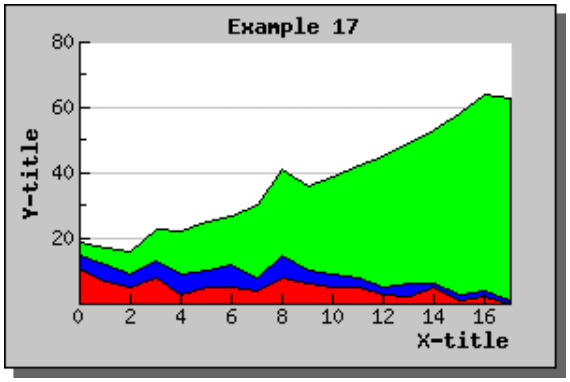


Figure 19. Example of Accumulated filled line plot.

1.15 Using elementary bar graphs

Version 1.0 of jpgraph only supports 2D vertical bar plots. Before you can use any bar plots you must make sure that you included the file "jpgraph_bar.php" in your script.

Using bar plots is quite straightforward and works in much the same way as line plots which you are already familiar with from the previous examples. Assuming you have a data array consisting of the values [12,8,19,3,10,5] and you want to display them as a bar plot. This is the simplest way to do this:

```
. . .
$datay=array(12,8,19,3,10,5);
$bplot = new BarPlot($datay);
$graph->Add($bplot);
. . .
```

This will then display a graph as

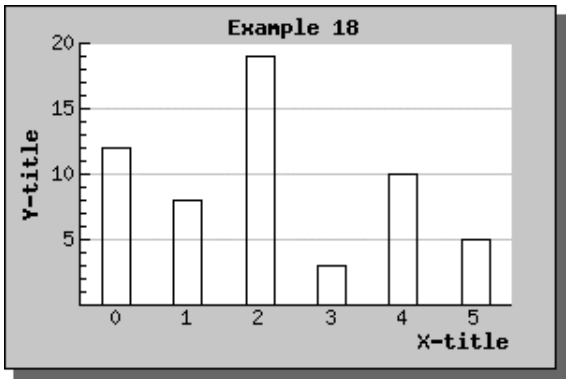


Figure 20. The simplest form of bar graphs

To have the bars filled with a solid color you must invoke the SetFillColor() method on the plot. So adding the line

```
$bplot->SetFillColor("orange");
```

will generate the following graph (no big surprise here..)

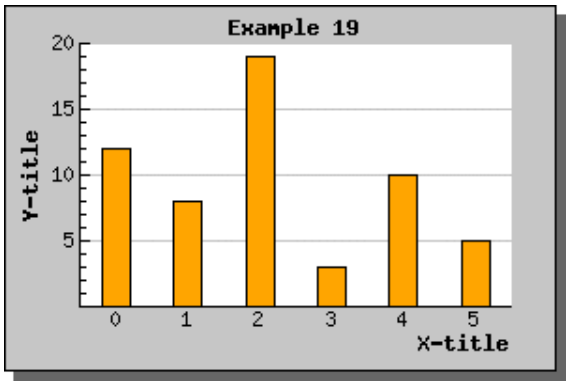


Figure 21. Filled bar graphs with default width.

You should note from the previous two graphs that bar graph gets automatically centred when using as text x-scale. If you were to use a linear scale they would instead start at the left edge of the X-axis.

1.16 Adjusting the width of the bars

By default the width of the bars are 40% of the major tick marks, i.e. the distance between two labels on the x-axis. To change this you will have to invoke the method SetWidth() with the percentage you would like to use instead, so for example having the bar graphs fill out the complete graph we specify a width of 100% (i.e. 1.0)

```
$bplot->SetWidth(1.0)
```

This would then generate the graph

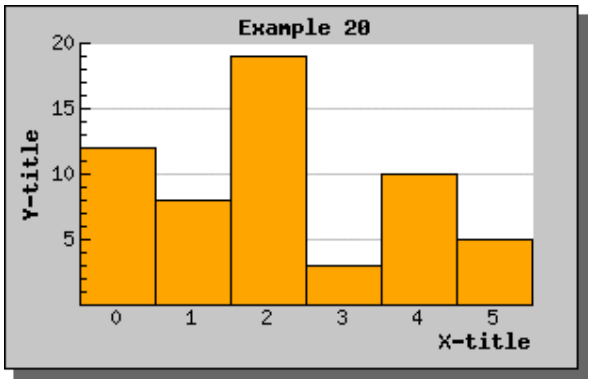


Figure 22. Bar graphs with a width of 100%

1.17 Using grouped bar graphs

These types of bars make is easy to group two or more bars together around each tick. The bars will be placed immediately beside each other and as a group centred on each tick mark. An example will make this clear.

```
. . .
// Some data
$data1y=array(12,8,19,3,10,5);
$data2y=array(8,2,11,7,14,4);

// Create the bar plots
$b1plot = new BarPlot($data1y);
$b1plot->SetFillColor("orange");
$b2plot = new BarPlot($data2y);
$b2plot->SetFillColor("blue");

// Create the grouped bar plot
$gbplot = new GroupBarPlot(array($b1plot,$b2plot));

// ...and add it to the graPH
$graph->Add($gbplot);
. . .
```

The above script will now generate the following image

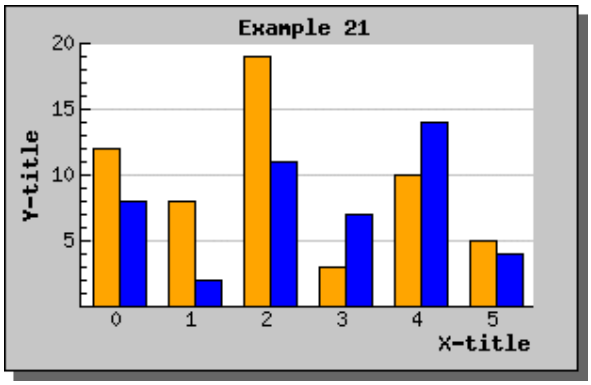


Figure 23. Example of grouped bars

There is no limit on the number of plots you may group other then purely visually, it might be hard to see a couple of thousand plots gr ouped together…

If you use the SetWidth() method on the GroupBarPlot() this will affect the total width used by all the added plots. Each individual bar width will be the same for all added bars. The default width for grouped bar is 70%.

So calling

```
$gbplot ->SetWidth(0.9);
```

would have the affect of generating the following image

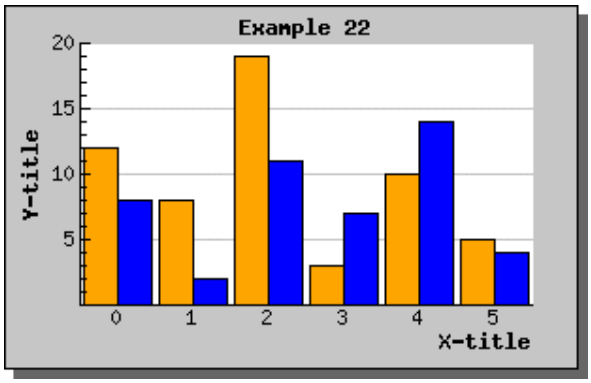


Figure 24. Grouped bar when the width has been specified as 90%

1.18 Using accumulated bar graphs

The final varieties of group bars you can have are accumulated bars. They work in much the same way as accumulated line plots described above. Each plot is stacked on top of each other. An example makes this clear. Let's use the same data as in the two examples above but instead of grouping the bars we accumulate (or stack) them. The code would be very similar (actually only one line has to change)

```
. . .
$abplot = new AccBarPlot(array($b1plot,$b2plot));
. . .
```

This would then generate the following graph.

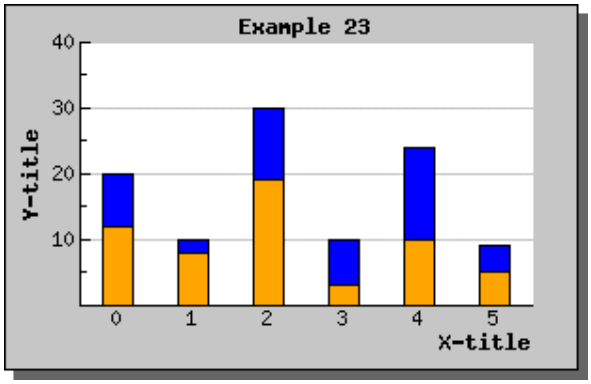


Figure 25. Accumulated bar plot.

As you can see each plot is stacked on top of each other.

1.19 Using grouped accumulated bar graphs

It is perfectly possible to combine the previous bar types to have grouped accumulated bar plots. This is done by just adding the different accumulated plots to a group bar plot, for example the following code would do that

```
// Create all the 4 bar plots
$b1plot = new BarPlot($data1y);
$b1plot->SetFillColor("orange");
$b2plot = new BarPlot($data2y);
$b2plot->SetFillColor("blue");
$b3plot = new BarPlot($data3y);
$b3plot->SetFillColor("green");
$b4plot = new BarPlot($data4y);
$b4plot->SetFillColor("red");

// Create the accumulated bar plots
$ab1plot = new AccBarPlot(array($b1plot,$b2plot));
$ab2plot = new AccBarPlot(array($b3plot,$b4plot));

// Create the grouped bar plot
$gbplot = new GroupBarPlot(array($ab1plot,$ab2plot));

// ...and add it to the graPH
$graph->Add($gbplot);
```

The resulting plot would now look like

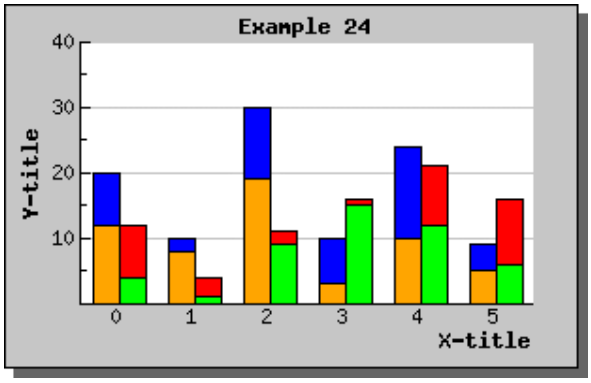


Figure 26. Combining both accumulated and grouped bar plots.

1.20 Using error plots

Error plots are used to visually indicate uncertain data points. This is done by for each X value give both a minimum and a maximum Y-value.

The following example illustrates a simple error bar. We will have 5 points, so we need 10 , so we need 10 Y-values. We also would like the error bars to be red and 2 pixels wide. All this is accomplished with (assuming the same basic graph as we used in previous examples)

```
. . .
$errdatay = array(11,9,2,4,19,26,13,19,7,12);
$errplot=new ErrorPlot($errdatay);
$errplot->SetColor("red");
$errplot->SetWeight(2);

$graph->Add($errplot);
. . .
```

The resulting graph would now look like

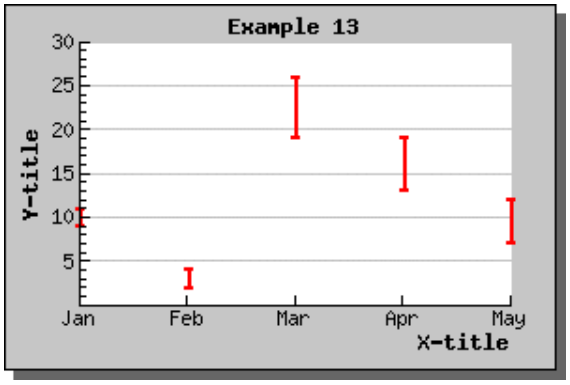


Figure 27. A simple example of error plot.

You might notice that there is one displeasing esthetical quality of this graph. The X-scale is just wide enough to just accompany the number of error bars and hence the first bar is drawn on the Y-axis and the and last bar just at the edge of the plot area. To adjust this you might call the SetCenter() method which will adjust the graph so that each X-point is centred in the middle of each major scale tick. The following example illustrates this

```
. . .
$errplot->SetCenter();
. . .
```

The resulting plot will now look more esthetic pleasing as

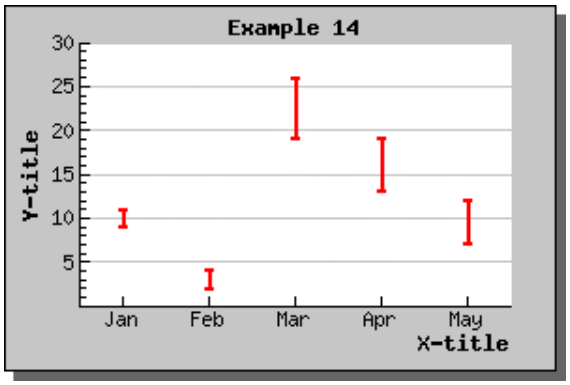


Figure 28. Centring an error graph with centred X-points within the major tick marks.

You might also note that the X-labels have also adjusted to this changed positioning, as you probably would expect.

1.21 Using line error plots

A line error plot is an error plot with the addition that a line is drawn between the average value of each error pair. You use this type of plot the exact same way you would use an error plot. The only change is that you must instantiated an ErrorLinePlot() instead and make sure you have included the “jpgraph_line.php” since the line error plot makes use of the line plot class to stroke the line, hence

```
. . .
$elplot=new ErrorLinePlot($errdatay);
. . .
```

To control the various properties of the line drawn the “line” property of the error line plot may be accessed. So, for example, if you want the line to be 2 pixels wide and blue you would have to add the following two lines

```
. . .
$elplot->line->SetWeight(2);
$elplot->line->SetColor(“blue”);
. . .
```

If we add that line to the previous example we will get the following graph

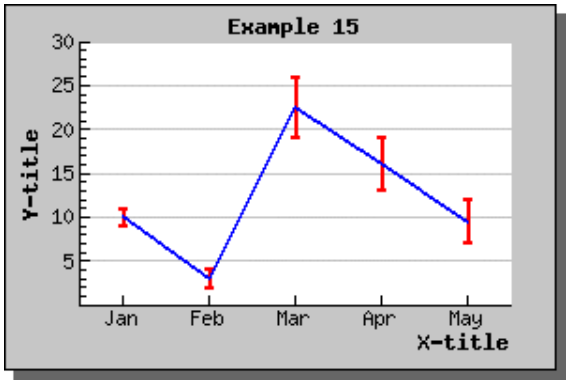


Figure 29. Example of a line error plot.

You may of course add legends to none, one or both of the line types in the above graph. So for example if we wanted the legend “Min/Max” for the red error bars and a legend “Average” for the blue line you would have to add the lines

```
$errplot->SetLegend("Min/Max");
$errplot->line->SetLegend("Average");
```

The resulting graph will now look like (note we are using the default placement of the legend box)

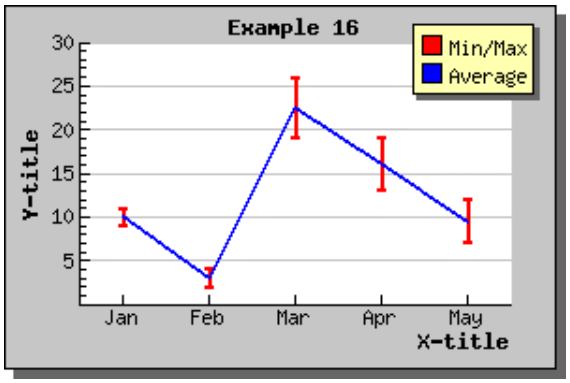


Figure 30. Line error plot with legends.

1.22 Combining different types of plots

It is perfectly legal to add several different plot types to the same graph. It is therefore possible to mix line plots with (for example) filled bar graphs. What you should keep in mind doing this is the order in which these plots are stroked to the image since a later stroke will overwrite a previous one. All plots are stroked in the order you add them, i.e. the first plot added will be stroked first. You can therefore control which plot is placed in the background and which one is placed in the foreground.

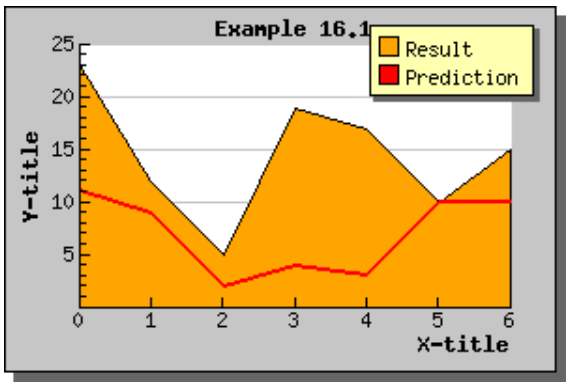
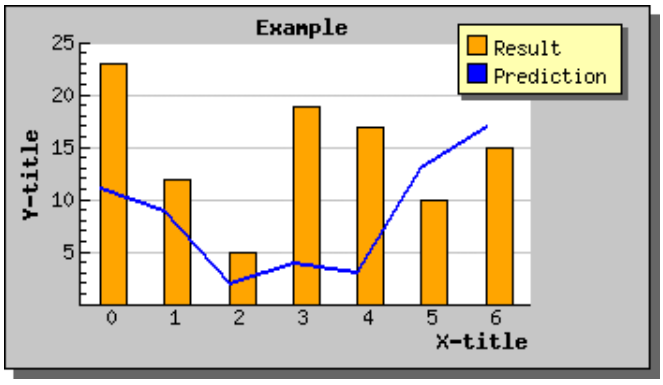


Figure 31. Example of plot containing both line plot and filled line plot.



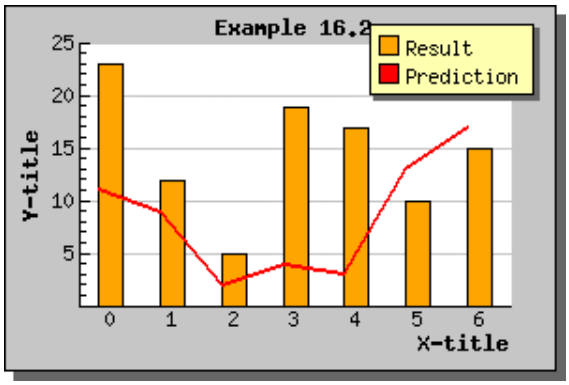


Figure 32. Example of graph with both line plots and bars.

Note the alignment of line plot together with bar plots. Line plots are aligned with the left edge of the bar. This is a deliberate design decision since It looks (to me) less esthetical to have the line centred in the middle of the bars.

Tip: If you want the graph with bars and line start at the very left edge just change the x-axis to use a linear scale instead of a text scale.

1.23 Adding text to the graph

It is possible to add any number of text strings freely positioned within the image. Each text string you want to add to the graph must be added as an instance of the Text class. The positions of the strings are given as percentage of the width/height of the image. A small example will demonstrate this. Lets add a red text “This is a text” to the middle by centring it horizontal in the graph, .

```
. . .
$txt=new Text(“This is a text”);
$txt->Pos(0.5,0.5,”centered”);
$txt->SetColor(“red”);
$graph->AddText($txt);
. . .
```

That’s it! You can also adjust the size and font of the text by using the “SetFont()” method. All available text methods are described in the reference section of the manual.

Note. The alignment you give tells how you want the layout algorithm to treat the positions you supply.

It is also possible to have the text surrounded by a, possible, filled box. This is accomplished by the SetBox() method.

```
. . .
$txt->SetBox(“white”,”black”,true);
. . .
```

The above line will add a textbox with a white background, black frame and a drop shadow. This is illustrated in the figure below

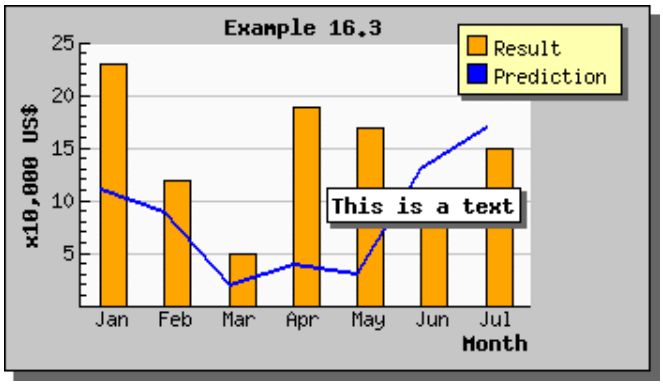


Figure 33. Example of added text box “This is a text”.

1.24 Using scatter plots

Scatter plots are very simple; they plot a number of points specified by their X- and Y-coordinate. Each point is stroked on the image with a mark as with lineplots.

Even though you would normally supply X-coordinates it is still perfectly possible to use a text-scale for X-coordinates to just enumerate the points. This is especially usefull when using the “Impuls” type of scatter plot as is shown below.

Scatter pots are created by including the jpgraph extension “jpgraph_scatter.php” and then creating an instance of plot type of ScatterPlot(). To specify the mark you want to use you access the mark with the instance variable “mark” in the scatter plot. The default is to use an unfilled small circle. An example clarifies this.

```
include( “jpgraph_scatter.php”);
. . .
$sp1 = new ScatterPlot( $datay, $datax);
. . .
$graph->Add($sp1);
. . .
```

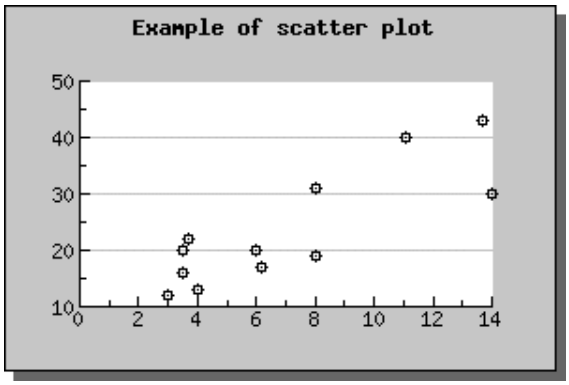


Figure 34. Example of scatter plot with default marks.

To change the apperance of the marks you can both fill tem with a specified color and you may also change their sze. Lets make the circle 10 pixels wide and filled with a red color. This is done by the lines

```
$sp1->mark->SetType(MARK_FILLEDCIRCLE);
$sp1->mark->SetFillColor("red");
$sp1->mark->SetWidth(10);
```

The resulting plot will now become

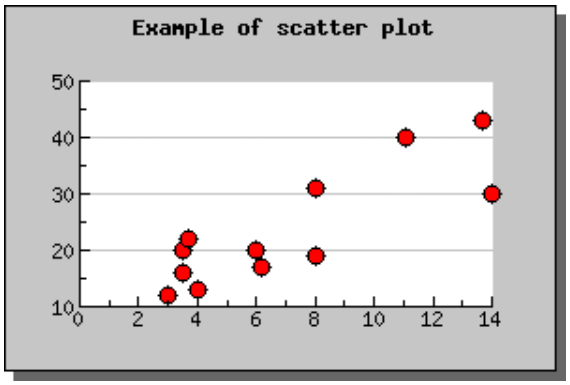


Figure 35. Example of scatter plot with modified marks.

For a complete list of available methods for "marks" see the reference section "Class PlotMark".

1.25 Using impuls scatter plots

A final modification we can do to scatter plot is to change it to a "impuls" type plot. This is simple a scatter plot with lines from the x-axis up to the mark. This type of plot is often used in conjunction with illustration of digital signal analysis (hence the name I've choosen).

This change is accomplished by calling the SetImpuls() method as in

```
. . .
$spl->SetImpuls();
```

An example plot (where we use a text X-scale) will now look like

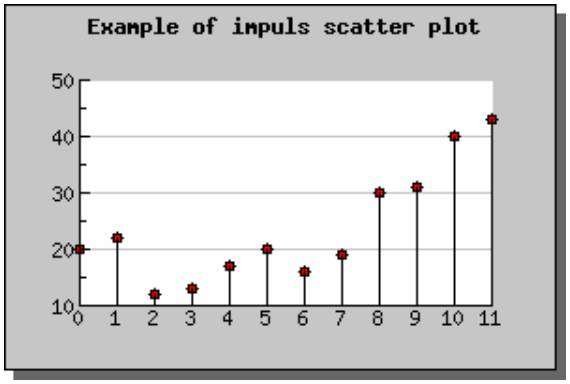


Figure 36. Example of scatter plot with Impuls style.

You may specify the thickness and color for the impuls line with the methods SetColor() and SetWeight() as in

```
. . .
$sp1->SetColor("blue");
$sp1->SetWeight(2);
```

The modified plot will then look like

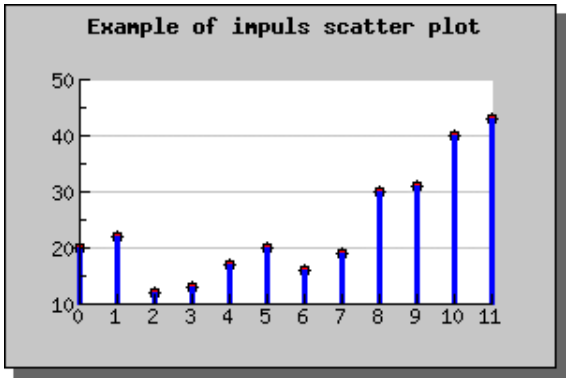


Figure 37. Example of impuls scatter plot with blue impuls lines.

You may draw impuls graphs without any mark by specifying the mark type as (-1) . That way only the impuls lines will be drawn. Applying this to the previous graph will then give the result

```
. . .
$sp1->mark->SetType(-1);
```

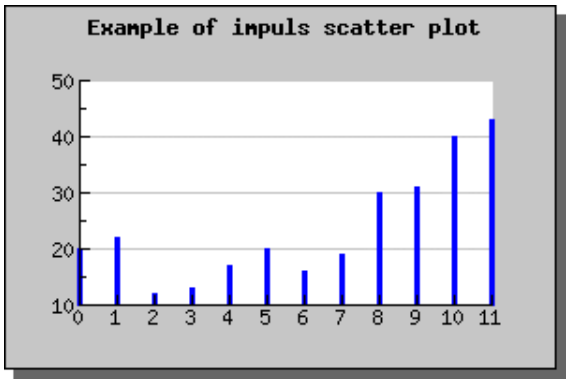


Figure 38. Impuls scatter plot with no marks.

1.26 Using Pie Plots

Ince by now you would have a fairly good understanding on the principles you will be pleased to find that Pie plots fit quite nicely in the previous framework.

To Use Pie plots you must include (as usual) the base library and the pile plot extension “jpgraph_pie.php” . Let’s show the simplest possible complete code for a pie plot

```
<?php
include ("jpgraph.php");
include ("jpgraph_pie.php");

// Some data
$data = array(40,21,17,14,23);

// Create the Pie Graph. Note you may cach this by adding the
// ache file name as PieGraph(300,300,"SomCacheFileName")
$graph = new PieGraph(300,200);
$graph->SetShadow();

// Set A title for the plot
$graph->title->Set("Example 1 Pie plot");
$graph->title->SetFont(FONT1_BOLD);

// Create graph
$p1 = new PiePlot($data);
$graph->Add($p1);

// .. and finally stroke it
$graph->Stroke();
?>
```

The generated graph will the be

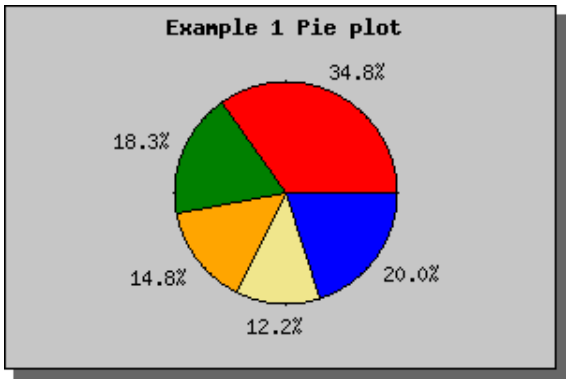


Figure 39. The simplest possible pie chart.

You may note a few thing

- ?? By default a set of standard color is used
- ?? By default the percentage for each slice is printed as a legend
- ?? By default the precision of the percent figures is to use one-decimal
- ?? By default the first slice always start at the horizontal axis (at 0 degree angle)
- ?? By default “Black” is used for lines.

The simplest addition we can do is now to add some explaining legends to what the different pie-slices stand for. This is accomplished by the method Setlegends(), lets name the legends after the months as an example by adding the line:

```
$p1->SetLegends(array("Jan","Feb","Mar","Apr","May"));
```

which will the generate the graph

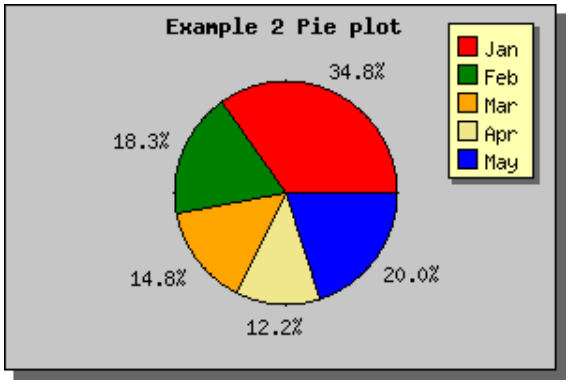


Figure 40. Pie chart with a legend.

1.27 Changing size and position for the pie chart

Changing size and position for the pie plot is accomplished by specifyin the size and position as percentage values. The size is changed by SetSize() which specifies the radius of the plot in percentage of whatever is the smallest of width and height of the image. The center of the pie is set by SetCenter(). An example of how to use these

methods are given in the next section when we show how we can add several pie charts to the same graph.

1.28 Adding several pie chart to the same graph

This is done completely analogues as with adding plots as we have seen before. Just create some more Pie plots and use the Add() method to add them to the image. One thing worth keeping in mind in regards to Legends. Since the pie graph only maintain one legend all the legend texts you add will be added to that legend. It is therefore most practical to use the same colors to mean the same things in each pie plot.

As an example lets take the previous image and just make four copies of the same pie plot just smaller so they fit within the image and place them evenly in a square, not much real use but it's getting late and I run out of imagination for new data….

I have also take the opportunity to set the size if the legend to the smallest font (with a call to SetFont()) so I don't have to make the image to large to fit all the plots.

However, we create the four plots with the lines

```
. . .
// Create plots
$size=0.13;
$p1 = new PiePlot($data);
$p1->SetLegends(array("Jan","Feb","Mar","Apr","May"));
$p1->SetSize($size);
$p1->SetCenter(0.25,0.32);
$p1->SetFont(FONT0);
$p1->title->Set("2001");

$p2 = new PiePlot($data);
$p2->SetSize($size);
$p2->SetCenter(0.65,0.32);
$p2->SetFont(FONT0);
$p2->title->Set("2002");

$p3 = new PiePlot($data);
$p3->SetSize($size);
$p3->SetCenter(0.25,0.75);
$p3->SetFont(FONT0);
$p3->title->Set("2003");

$p4 = new PiePlot($data);
$p4->SetSize($size);
$p4->SetCenter(0.65,0.75);
$p4->SetFont(FONT0);
$p4->title->Set("2004");

$graph->Add($p1);
$graph->Add($p2);
$graph->Add($p3);
$graph->Add($p4);

$graph->Stroke();
```

Note: We only set the legend for the first pie plot since we assume that the other plot have the same meaning.

You may note that I also used the “title” property for each plot to assign each plot an individual title. (You may also add other text to the graph by creating instances of Class Text() and add them to the graph via the AddText() method in the PieGraph class.)

The plot will now become

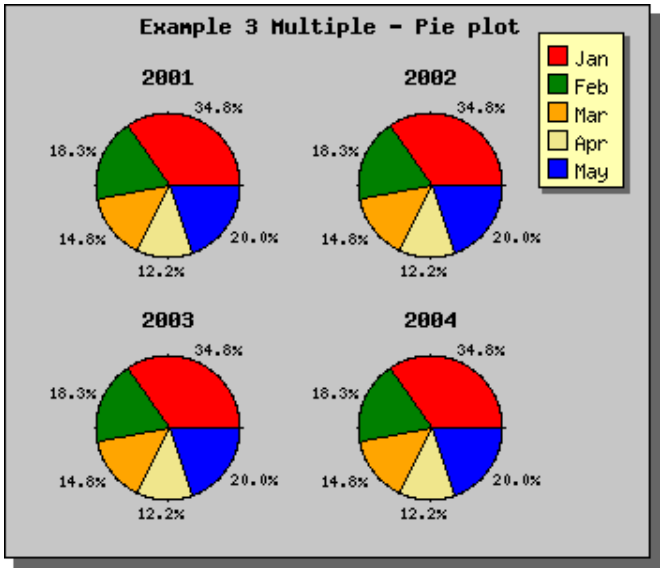


Figure 41. Example of adding several pie plots to the same graph.

1.29 Additional modifications to pie plots

Just a quick note on some additional modifications you might do to pie plots.

- Hiding labels. You may hide the percentage labels for a plot by a call to the method `HideLabels()`
- Changing the colors of the labels by a call to `SetFontColor()`
- Set precision of percentage figure with a call to method `SetPrecision()`
- Setting different colors to pie then default by calling `SetSliceColors()`

The above are all methods in the PiePlot class. For a complete overview of all the methods see the reference section.

1 Jpgraph reference

1.1 Conventions

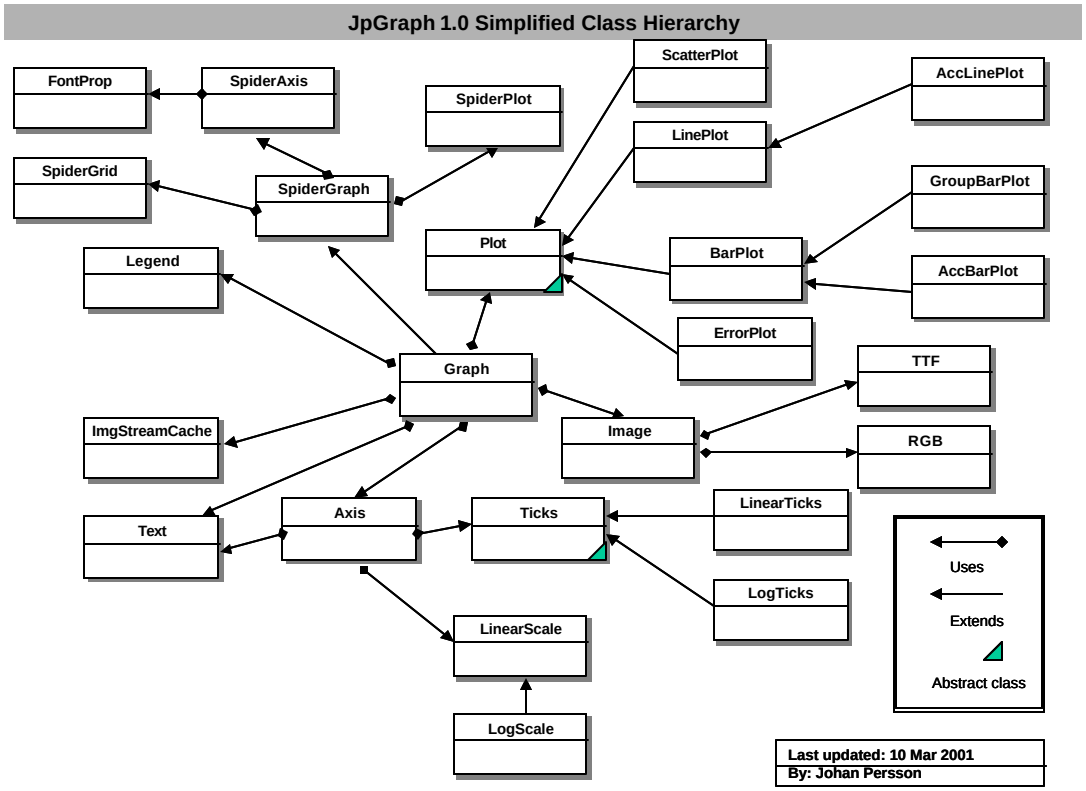
All classes are documented with their private and public methods as well as their parents to clearly show which methods have been overridden. Since PHP is not a strict typed language I have taken the liberty of specifying each method with the type you would call it with to make it clearer what is intended.

When properties are listed their Class (if any) are specified as a normal type. When arguments for methods have default arguments they are specified.

An overview of available methods are given for each class in a table format. For each class that extends another class the immediately parent class is also given for reference. Methods which are overridden in a child class is indicated as strike-through in the parent class to emphasise that the method in the child is used.

1.2 Class overview

The class diagram below shows a simplified version of the overall (simplified) class hierarchy used in JpGraph.



1.3 Public Class references

The following section describes all the classes used in the library. For each class the file where it is defined is specified, and it’s class hierarchy.

6.3.1 Class Graph

Defined in file: jpgraph.php

Public properties
Class Axis xaxis,yaxis,y2axis; Class Grid sgrid,ygrid,y2grid; Class Image img; Class Text title;
Public methods
Graph() Add() AddY2() AddText() Box() SetColor() SetMarginColor() SetFrame() SetShadow() SetScale() SetY2Scale() SetTickDensity() Stroke()
Private properties & methods
Class LinearScale xscale, yscale, y2scale; GetPlotsYMinMax() StrokeFrame()

General description

The Graph class is the main container class which controls the creation of the entire graph. You must always instantiate one instance to create a graph. Through this class one controls many overall settings of the image displayed.

Graph(int $width, int $height, String $cacheName="")

Parameters:

```
Int width           Width in pixel of the overall image generated
Int height          Height in pixel of the overall image generated
String cacheName    Name for picture in cache.
```

Description:

Creates a new graph. This is often the first call made to set-up a new graph..
If the cache name is specified then the method will first try to locate the named file in the "./cache/" directory rather then generating the graph on the fly. If the file is not there the graph will be generated and saved as the specified file. This file is the passed through to the browser.
If no cache name is specified then the graph will always be generated and the cache bypassed.

Before any other operation is performed on the graph a call to SetScale() should be made to finish the initialisation of the graph.

Returns:

NA

See also:

Class ImgStreamCache

Example:

$graph = new Graph(300,200); // Create a 300x200 big image to work with

Add(&Class Plot)

Parameters:
```
Class Plot          Plot to be added to the graph
```

Description:
Each plot that should be displayed within the graph has to be added to the graph. This method will add the plot so it will use the "Left" Y-scale, (the normal Y scale).

Note that since the plot is added as a reference any changes you make to the original plot will also happen to the plot you have added to the graph.

Returns:
NA

See also:
AddY2(), SetScale()

Example:
```
$lineplot = New LinePlot($datay);
$graph->Add($lineplot);
$lineplot->SetColor("red");      // Will affect the graph
```

AddY2(&Class Plot)

Parameters:
```
Class Plot          Plot to be added to the graph
```

Description:
Works the same way as Add() but the plot is added for use with the Y2 scale (the right Y scale) instead.

Returns:
NA

See also:
Add(), SetY2Scale()

Example:
```
$graph->new Graph(300,200);
$graph->SetScale("linlin");
$graph->SetY2Scale("linlog");
$lineplot = New LinePlot($datay);
$graph->AddY2($lineplot);
```

AddText(&Class Text)

Parameters:
```
Class Text          Text object to be added to the graph
```

Description:
Adds an instance of the Text class to the graph, allowing arbitrary text to be placed anywhere in the graph.

Returns:
NA

See also:
Class Text

Example:
```
$caption=new Text("Figure 1. Temperature over time",0.1,0.8);
$caption->SetFont(FONT1_BOLD);
$graph->AddText($caption);
```

SetBox(Boolean $box=true, Int $weight=1, Color $color="black")

Parameters:

```
Boolean $b          Flag to set plotarea box on or off
Int weight          Line weight for box
Color color         Line color
```

Description:
This is used to specify whether the plot-area should have a rectangle around it and the specifics of that rectangle.

Note: As of version 1.0 the weight parameter is not honoured and hence the box will always be one pixel wide.

Returns:
NA

See also:
SetFrame()

Example:

```
$graph->new Graph(300,200);
$graph->SetScale("linlin");
$graph->Box();
```

SetColor(Color $c)

Parameters:

```
Color $c            Set the background color for the plot are
```

Description:
Sets the background color for the plot-area.

Returns:
NA

See also:
SetMarginColor()

Example:
$graph->SetColor("wheat");

SetMarginColor(Color $c)

Parameters:

```
Color $c            Set the background color for the margins
```

Description:
Specifies the color of the area between the plot area and the edge of the image.

Returns:
NA

See also:
SetColor(), SetMargins()

Example:

SetFrame(Boolean $frame=true, Color $color="black", Int $weight=1)

Parameters:

Boolean frame	Flag if the fram around the image should be drawn or not
Color color	Color of the frame
Int weight	Line weight for the frame

Description:
Sets a frame (rectangle) of the chosen color around the edges of the image.

Returns:
NA

See also:
SetBox()

Example:
$graph->SetFrame();

SetShadow(Boolean $shadow=true,int shadowWidth=4,Color shadowColor=grey40)

Parameters:

Boolean shadow	Flag if the shadow should be displayed or not
Int shadowWidth	Shadow width
Color shadowColor	Shadow color

Description:
Sets a frame with a shadow around the entire image

Returns:
NA

See also:
SetFrame()

Example:
$graph->SetShadow()

SetScale(String $axtype,int $ymin=1,int $ymax=1,int $xmin=1,int $xmax=1)

Parameters:

String $axtype	Type of scale
Int $ymin	Min Y scale value
Int $ymax	Max Y scale value
Int $xmin	Min X scale value
Int $xmax	Max X scale value

Description:
Specifies what kind of scales should be used in the graph. The following combinations are allowed

?? Linear scale. Both X and Y axis
?? Logarithmic scale. Both X and Y axis
?? Text scale. Only on X axis

Any combination of these may be used. Linear and logarithmic scales are pretty straightforward. The text scale might deserve some explanation. The easiest way to think of the text scale is as a linear scale consisting of only natural numbers, i.e. 0,1,2,3,4,… . This scale is used when you just have a number of Y-values you want to plot in a consecutive order and don't care about the X-values

To specify which combination of X and Y scales you want to use the $axtype parameter is specified. The following values are allowed

?? "linlin" Linear X, Linear Y

?? “linlog” Linear X, Log Y
?? “textlin” Text X, Linear Y
?? “textog” Text X, Log Y
?? “loglin” Log X, Linear Y
?? “loglog” Log x, Log Y

It is normally recommended to use the auto-scaling feature since for most practical purposes it is good enough. However on rare occasions you might want to specify the limits yourself. This is then done by the rest of the parameters to the method.

Note: If you want to use a logarithmic scale you must make sure that the “jpgraph_log.php” is included.

Returns:
NA

See also:
SetY2Scale()

Example:
$graph->SetScale(“textlin”);

SetY2Scale(String $axtype, int $ym in, $ymax)

Parameters:

```
String $axtype      Type of scale
Int $ymin           Min Y scale value
Int $ymax           Max Y scale value
```

Description:
The graph allows two different Y-scales to be used and you can choose which one you want to use for a specific plot by the way you are adding the plot to the graph, either by Add() or by AddY2() method.

This method works in the exact same way for the Y2 axis as the SetScale() method previously described.

Allowed values for the $axtype are
?? “lin” Linear scale
?? “log” Logarithmic scale

Note: If you want to use a logarithmic scale you must make sure that the “jpgraph_log.php” is included.

Returns:
NA

See also:
SetScale()

Example:
$graph = new Graph(300,200);
$graph->SetScale(“textlin”);
$graph->SetY2Scale(“log”);

SetTickDensity(int $densy=TICKD_NORMAL, int $densx=TICKD_NORMAL)

Parameters:

```
Int $densy          Density hint for Y axis autoscaling
```

```
Int $densy          Density hint for X axis autoscaling
```

Description:
This method is used to hint how many ticks the auto-scaling should try to fit on each of the axis.
The following defines may be used to hint to the auto-scaling how many ticks should be allocated

??	TICKD_DENSE	Small distance between ticks
??	TICKD_NORMAL	Default value
??	TICKD_SPARSE	Longer distance between ticks
??	TICKD_VERYSPARSE	Very few ticks

Returns:
NA

See also:
NA

Example:
$graph->SetTickDensity(TICKD_DENSE); // Many Y-ticks

Stroke()

Parameters:
NA

Description:
Should be the final method called in the script that generates a graph. This will generate the image and send it back to the browser

Returns:
NA

See also:
NA

Example:
$graph->Stroke()

6.3.2 Class Axis

Defined in file: jpgraph.php

Public properties
`Class LinearScale scale;`
`Class Text title;`
Public methods
`Hide()`
`HideFirstTickLabel()`
`SetColor()`
`SetWeight()`
`SetTitle()`
`SetTickLabels()`
`SetTextTicks()`
`SetLabelPos()`
`SetFont()`
Private properties & methods
`Axis()`
`Stroke()`

General description

The Axis class is used to represent both the X and Y axis in the graph. It is possible to control the individual properties of the axis such as color, weight, font used for labels, title etc through the method defined in this class.

Instances

```
$graph->xaxis
$graph->yaxis
$graph->y2axis
```

Axis(Class Image &$img, &$aScale, $color="black")

Parameters:

```
Class Image &$img                Type of scale
Class LinearScale &$aScale       Min Y scale value
Color $ymax                      Max Y scale value
```

Description:

Creates a new axis. A new axis can be either a X-axis or an Y-axis. To create a new axis one supplies an Image and a scale. It is also possible to specify the color: Colors may also be specified through the SetColor() method.

Returns:

NA

See also:

Class LinearScale, Class LogScale, Class Image

Example:

Hide(Boolean $h=true)

Parameters:

```
Boolean $h
```

Description:

Hides the axis if $=true

Returns:

NA

See also:

Example:
```
$graph->yaxis->Hide()
```

HideFirstTickLabel(Boolean $flag=false)

Parameters:
```
Boolean $flag
```

Description:
If you (for esthetical reason) does not want to display the first tick label you call this method.

Returns:
NA

See also:

Example:

SetWeight(int $weight)

Parameters:
```
Int weight
```

Description:
Specify line weight of the axis.

Returns:
NA

See also:
SetColor()

Example:
```
$graph->yaxis->SetWeight(2)
```

SetColor(Color $color)

Parameters:
```
Color Color
```

Description:
Specified color of sxis.

Returns:
NA

See also:
SetWeight()

Example:
```
$graph->yaxis->SetColor()
```

SetTitle(String $t, String $adj="high")

Parameters:
```
String $t
String $adj
```

Description:

Specify title for the axis. The title may also be accessed as the “title” property of the axis. The title may be adjust to either in the middle, at the high end or at the low end of the axis.

This method is actually a shortcut for $axis->title->Set($t). To change the specifics of the title (like color or font) you apply the suitable method on the title property.

Returns:
NA

See also:

Example:
$graph->xaxis->SetColor(“red”);
$graph->xaxis->SetFont(FONT1_BOLD);

SetTickLabels(String Array $l)

Parameters:
`String Array $l`

Description:
Normally ticks are given numeric values corresponding to it’s position on the scale. However it is also possible to specify alternative labels, for example you might want to have the name of the months on the x-axis.

When using this method you should supply a value for each major tick mark.

Returns:
NA

See also:

Example:
$months = array(“Jan”,“Feb”,“Mar”,“Apr”,“May”,“June”);
$graph->xaxis->SetTickLabels($month);

SetTextTicks(int $step, int $start=0)

Parameters:
`Int $step`
`Int $start`

Description:
When you have specified a text scale for the X-axis by default every whole number is used as a major tick, i.e if you have 10 data-points the x-axis will have the labels (0,1,2,3,4,5,6,7,8,9). If you have many data-points you might not want to display all these labels. This method let’s you control which labels will be displayed.

The first parameter $step specifies that every $step ticks should be displayed. For example SetTextTicks(2) will cause every second label to be displayed so given the 10 data-points before the scale will now display (0,2,4,6,8).

The other parameter $start specifies which offset should the scale should start on, For example SetTextTicks(2,1), will generate the scale (1,3,5,7,9).

If you combine both SetTickLabels() and SetTextTicks() you can fully control which data-points have your specified text label.

Returns:
NA

See also:
`SetTickLabels()`

Example:
```
$month = array(“Feb”,”Apr”,”Jun”,”Aug”,”Oct”,”Dec”);
$graph->xaxis->SetTextTicks(2,1);
$graph->xaxis->SetTickLables($month)
```

SetLabelPos(int $pos)

Parameters:
`Int $pos`

Description:
Specify which side of the axis you want the text labels on. Valid values for $pos are (1, -1)

Returns:
NA

See also:

Example:
```
$graph->y2axis->SetLabelPos(-1); // Set labels to the left of the Y2 axis.
```

SetFont(int $size, String $font="internal")

Parameters:
```
Int $size
String $font
```

Description:
Specify font for labels in the axis.

Returns:
NA

See also:

Example:

Stroke(Class LinearScale $otherAxisScale)

Parameters:
`Class LinearScale $otherAxisScale`

Description:
Draws the axis. Since the position of the axis is specified in relation to the other axis it is also necessary to supply the other scale as a parameter to draw the axis. By default the axis is place at the lower end at the other axis if not otherwise specified with a call to Pos().

Returns:
NA

See also:

Example:

6.3.3 Class Ticks

Defined in file: jpgraph.php

Public properties
`Class LinearScale scale;`
`Class Text title;`
Public methods
`SetColor()`
`SetWeight()`
SupressZeroLabel()
SupressMinorTickMarks()
SupressFirst()
SetPrecision()
SetDirection()
Private properties & methods
`Class Image img;`
`Stroke()`
`Ticks()`
`GetMinTickAbsSize()`
`GetMajTickAbsSize()`

General description

Abstract base class for the linear and logarithmic ticks. Internal class which does never have to be instantiated. Responsible for the overall layout and format for tick lines. Note that the actual tick labels are drawn by the Axis class based on tick position calculations computed by actual subclasses to this class.

Instances
$axis->scale->ticks

Ticks(&$scale)

Parameters:
`$scale` `Scale to fit ticks on`

Description:
Construct ticks for the specified scale.

Returns:
NA

See also:
LinerTicks(), LogTicks()

Example:

GetMinTickAbsSize()

Parameters:
NA

Description:
Get distance in pixels between minor tick marks.

Returns:
NA

See also:

Example:

SupressZeroLabel(Boolean $z=true)

Parameters:
$z TRUE/FALSE

Description:
Specify whether a label with numeric value 0 should be displayed

Returns:
NA

See also:

Example:

SupressMinorTickMarks(Boolean $tm=true)

Parameters:
$tm TRUE/FALSE

Description:
Specify whether minor tick marks should be displayed or not.

Returns:
NA

See also:

Example:

SupressFirst(Boolean $ft=true)

Parameters:
$ft TRUE/FALSE

Description:
Determine if the first tick mark should be displayed or not. It is sometimes useful to suppress the first tick mar if the labels from both scales gets very close to each other.

Returns:
NA

See also:

Example:

GetMajTickAbsSize()

Parameters:
NA

Description:
Get distance, in pixels, between Major tick marks.

Returns:
NA

See also:

Example:

Set(real $maj, real $min)

Parameters:

`$maj` Specify, in world coordinates, the distance between major tick marks
`$min` Specify, in world coordinates, the distance between minor tick marks

Description:
Specify where the major and minor tick marks should be.

Returns:
NA

See also:

Example:

SetPrecision(int $p)

Parameters:

$p Number of decimal points

Description:
Specify how many decimals should be displayed in the automatic labels

Returns:
NA

See also:

Example:

SetDirection(int $dir=1)

Parameters:

$dir -1 for left (or up), +1 for right (or down)

Description:
Specify if the tick marks should be to the left or right side for an Y-axis or on the up or down side for an X-axis.

Returns:
NA

See also:
SetDirection for class Axis which specifies which side the labels should go on.

Example:

6.3.4 Class Text

Defined in file: jpgraph.php

Public properties
Public methods
Text() Set() Hide() Center() SetColor() SetFont() SetBox() SetOrientation() GetWidth() GetFontHeight()
Private properties & methods
Stroke()

General description

Represents a text string which may be added to the graph area in an auxiliary position.

Text(String $txt="",real $x=0, real $y=0)

Parameters:

$txt	Text string to display
$x	X-position in percent of image width. 0 percent is the left edge
$y	Y-position in percent of image width. 0 percent is the top edge

Description:

Creates a new text object which may be displayed anywhere within the image. The text object is then added to a specific graph through the AddText() method in the Graph class.

Returns:

NA

See also:

Example:

```
$t1 = new Text("Overview",100,180);
$graph->AddText($t1);
```

Set(String $t)

Parameters:

$t	Text string

Description:

Set the text for a previous created Text object.

Returns:

NA

See also:

Example:

```
$t1->Set("New title");
```

SetBox(Mix $fcolor=array(255,255,255), Color $bcolor=array(0,0,0), $shadow=false)

Parameters:

`$fcolor`	Box fill color, or FALSE if no box should be displayed
`$bcolor`	Box frame color
`$shadow`	Specifies if the box should have a drop shadow

Description:
Specifies that the text should be in a frame. If fill color is specified as “nofill” then the text will be framed but will not have a filled background.

Returns:
NA

See also:
Class Image :: StrokeBoxedText()

Example:

Pos(real $x=0, real $y=0,String $halign="left")

Parameters:

$x	X-Coord in percent of image width
$y	Y-Coord in percent of image width
$haling	Horizontal alignment

Description:
Set position and specify alignment.

Returns:
NA

See also:
Class Image :: StrokeText()

Example:

Hide(Boolean $f=true)

Parameters:
$f TRUE/FALSE

Description:
Hide the text. The test will not be drawn.

Returns:
NA

See also:

Example:

SetFont(int $size, String $name="internal")

Parameters:
$size
$name

Description:
Specifies text font. See Image::SetFont() for a detailed description.

Returns:

NA

See also:

Example:
```
$t1->SetFont(FONT1_BOLD);
```

Center(int $left, int $right, Boolean Mixed $y=false)

Parameters:

$left	Left x-coordinate
$right	Left x-coordinate
$y	If specified, the Y-coordinate

Description:
Center the text between the two X-coordinates using possible a previous specified Y-coordinate.

Returns:
NA

See also:
Pos()

Example:

SetColor(Color $color)

Parameters:

$color	Color of text

Description:
Specify text color to be used.

Returns:
NA

See also:

Example:
```
$t1->SetColor(“navy”);
```

SetOrientation(String $d="horizontal")

Parameters:

$d	Specify if the text should be drawn vertical or horizontal.

Description:
Set the orientation of the text, either vertical or horizontal.

Returns:
NA

See also:
Pos()

Example:

GetWidth(Class Image &$img)

Parameters:

$img	The image we are drawing to

Description:
Returns the width, in pixels, of the text

Returns:
NA

See also:

Example:

GetFontHeight(Class Image &$img)

Parameters:
`$img` The image we are drawing to

Description:
Returns the height, in pixels, of the text

Returns:
NA

See also:

Example:

Stroke(Class Image &$img)

Parameters:
`$img` The image we are drawing to

Description:
Stroke the text to the specified image.

Returns:
NA

See also:

Example:

6.3.5 Class Grid

Defined in file: jpgraph.php

Public properties
Public methods
SetLineStyle()
Show()
SetWeight()
SetColor()
SetWeight()
Private properties & methods
Grid()
Stroke()

General description
This class handles the drawing of the grid lines based on the calculations done by the Tick class which is responsible for determine the exact positions of each vertical or horizontal tick mark.

You normally manipulates the grid as an instance in the graph class, either as $graph->xgrid or as $graph->ygrid

Grid(Class Axis &$axis)

Parameters:
$axis Axis to which the grid lines belong

Description:
Handles the gridlines for the specified axis. Gridlines can be drawn on either just major ticks or on both major and minor ticks. The default is to draw grid liens on major ticks only.

Returns:
NA

See also:

Example:

SetWeight(int $weight)

Parameters:
$weight in pixels

Description:
Specify weight in pixels for the gridlines.

Returns:
NA

See also:

Example:

SetColor(Color $color)

Parameters:
$color

Description:

Specify color for gridlines. Default is a very light grey color.

Returns:
NA

See also:

Example:

SetLineStyle(String $type)

Parameters:
$type Type of gridlines, see below

Description:
Specify line style for gridlines. Allowed styles are

- ?? “solid”
- ?? “dotted”
- ?? “dashed”
- ?? “longdashed”

Deafult is “solid”.

Returns:
NA

See also:

Example:

Show(Boolean $major=true, Boolean $minor=false)

Parameters:
$major Show/Hide Major gridline
$minor Show/Hide Minor gridline

Description:
Determine what gridlines are visible. Default is to show only major gridlines.

Returns:
NA

See also:

Example:
```
$graph->ygrid(true,true);  // Show both maj and minor gridmarks
```

Stroke()

Parameters:
NA

Description:
Draw the gridlines as previously specified. The gridlines will only be drawn within the plot area of the image. This is an internal method and should never be called from user level code.

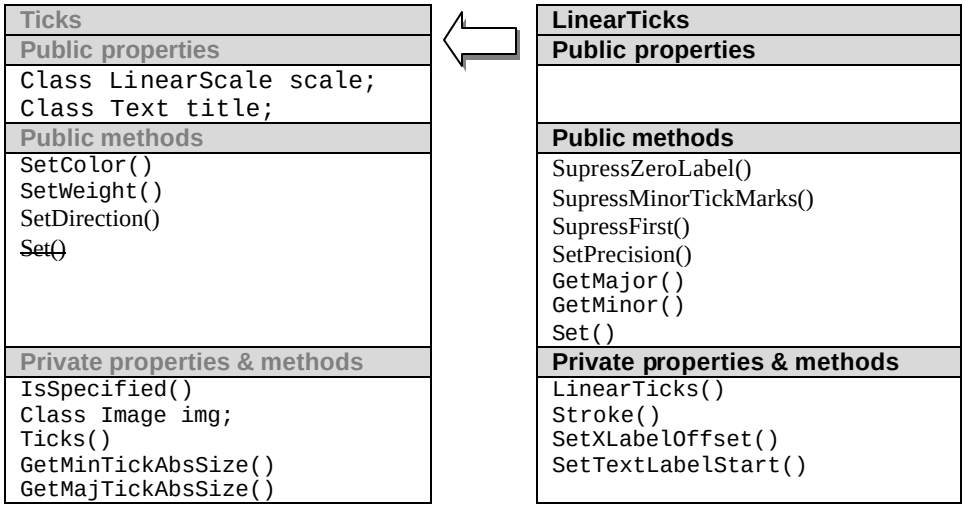
Returns:
NA

See also:

Example:

6.3.6 Class LinearTicks

Defined in file: jpgraph.php
Extends Ticks.



General description
The concrete class which implements linear ticks for X and Y axis. This class should be used through it's instance as a property of the scale.

Instantiated
$grasph->xaxis->scale->ticks

LinearTicks()

Parameters:
NA

Description:
Create a new instance. Note this is a private method which should not be called by users of this library directly.

Returns:
NA

See also:
LogTicks()

Example:

GetMajor()

Parameters:
NA

Description:
Get major step size in world coordinates

Returns:
NA

See also:
GetMinor()

Example:

GetMinor()

Parameters:
NA

Description:
Get minor step size in world coordinates

Returns:
NA

See also:

Example:

Set(real $maj_step, real $min_step)

Parameters:

$maj_step	Specify major step size in world coordinates
$min_step	Specify minor step size in world coordinates

Description:
Set the step size to beused for minor and major ticks. Note you should normally let the autoscaling handle this since that is for most practical purposes good enough.

Returns:
NA

See also:

Example:

Stroke(Class Image &$img, Class LinearScale &$scale, int $pos)

Parameters:

$img	The image to be drawn to
$scale	The scale associated with these ticks
$pos	Determine which side of the axis the ticks should go on,allowed values are
	-1 Left/Up
	1 Right/Down

Description:
Stroke the tick marks to the image. This method is private to the library and should never be called directly.

Returns:
NA

See also:

Example:

6.3.7 Class LinearScale

Defined in file: jpgraph.php

Public properties
`Class Ticks ticks`
Public methods

`GetMinVal()`
`GetMaxVal()`
`Update()`
`Translate($a)`
`SetColor()`
`SetWeight()`
`SetGrace()`

Private properties & methods

`LinearScale()`
`Init()`
`IsSpecified()`
`SetMin()`
`AutoScale()`
`CalcTicks()`
`MatchMin3()`
`InitConstants()`
`Stroke()`

General description

The general scale class which represent the scale on either a X or Y axis. If the scale is not explicitly set if will be automatically determined based on the min and max values of all the plots using this scale. Both X and Y axis may have a linear scale. A special version of the linear scale is the "text" scale which is a scale only containing whole numbers. Used to represent counting scales. A text scale may only be used for a X-axis.

LinearScale(real $min=0, real $max=0, String $type="y")

Parameters:

$min Minimum world value to be represented
$max Maximum world value to be represented
$type Determines if this is a X or Y axis

Description:

Create a new instance of a linear scale. This is a private method to the library and should as such never be called directly.

Returns:

NA

See also:

LogScale()

Example:

Init(Class Image &$img)

Parameters:

$img The image where the scale should be used

Description:

Second phase initialisation. Used to add the scale as an observer to the Image class since we need to get notified if the image changes it's parameters, for example if the margin are changed we must recalculate our scaling constants.

Note this can't be done in the constructor due to a bug in PHP4 that will not allow you to use a reference to "this" pointer in the constructor. Strictly speaking it will of course allow you to use it but it won't work!

Internal method that should never be called by users of this library directly.

Returns:
NA

See also:
Image::AddNotifyer()

Example:

IsSpecified()

Parameters:
NA

Description:
Determine if the scale has been manually specified or not. Used to determine if the scale should be auto-scaled or not.

Returns:
TRUE/FALSE

See also:

Example:

SetAutoMin(real $min)

Parameters:
$min Min value in world coordinates for auto-scaling

Description:
By default the auto-scaling will use the lowest value of the plots as the minimum value of the Y-scale. If the chosen value falls "close to" 0. Then zero will be chosen. However, it is sometimes useful to hard set the minimum value used by the auto-scaling and then just have the auto-scaling determine the maximum (normally) Y-value.

This method allows you to do just that.

Returns:
NA

See also:

Example:

GetMinVal()

Parameters:
NA

Description:
Get the minimum world coordinate.

Returns:
NA

See also:

Example:

GetMaxVal()

Parameters:
NA

Description:
Get the maximum world coordinate.

Returns:
NA

See also:

Example:

Update(Class Image &$img, Real $min, Real $max)

Parameters:
$img Image where the scale is used
$min Minimum world coordinate
$max Maximum world coordinate

Description:
This method is really design as a the observer notification method. This will update internal constants that is used to perform the scaling between world and screen. This will get automatically called if, for example, the margins of the image are changed.

Note that this method should normally never be called directly by a user using this library.

Returns:
NA

See also:

Example:

Int Translate(Real $a)

Parameters:
$a World coordinate

Description:
Translates a given world coordinate to the corresponding screen pixel position within the image.

Returns:
Screen coordinate in pixels

See also:
NA

Example:

SetGrace($grace)

Parameters:

$a Grace factor

Description:
Adds $grace percent to the max and min values used for autoscaling ti make scale larger then the actual min and max values found in the data. The grace to add is calculated as the percentage of total dynamic range., i.e. (max-min) which is then added to the max value and subtracted from the mijn value to make the scale larger. A value of 10 normally gives satisfactory result. High values will make the graph look very compressed.

Returns:
NA

See also:
NA

Example:
```
$graph->yscale->SetGrace(10); // Set 10% grace value to Y-scale
```

AutoScale(Class Image &$img, Real $min, Real $max, int $maxsteps, Boolean $majend=true)

Parameters:
```
$img          Image which is drawn to
$min          Minimum world coordinate
$max          Maximum world coordinate
$maxsteps     Maximum number of major steps allowed
$majend       Should the scale end at a major tick?
```

Description:
Performs autoscaling of the scale given the min/max and the number of maximum major ticks allowed. Note that the autoscaling algorithm will most likely adjust the minimum and maximum values to better fit within the scale chosen.

The maxsteps should in general be a function of the image size since a larger image can accommodate more ticks.

The autoscaling is quite smart in that it actually performs a small search among some standard scale step (multiple of 2, 5, 1 etc) to see which one fits best with the number of maximum steps. The autoscaling has a slight preference to steps of 5 (0.5, 0.05 etc) so if there is a close match the steps of 5's will be chosen.

The end of the scale can finish either on a minor or major tick mark. If you want the scale to end on a major ticks mark, and hence have a potential label, the parameter `$majend` should be true.

Returns:
NA

See also:
CalcTicks()

Example:

InitConstants(Class &$img)

Parameters:
Img Image to draw top

Description:
Internal method. Initiates constants. Shuld never be called directly.

Returns:

NA

See also:

Example:

CalcTicks(int $maxsteps, Real $min, Real $max, int $a, int $b, Boolean $majend=true)

Parameters:

$maxsteps	maximum number of major steps allowed on the scale
$min	Min world coordinate
$max	Max world coordinate
$a	Algorithm Parameter a
$b	Algorithm Parameter b
$majend	Should the scale end on a major tick?

Description:
The internal work routine which tries to fit a number of ticks given the parameters a and b. The parameters will control what type of ticks we will be trying, steps of 2, 5 etc.

This is completely an internal routine and should never be called. Only documented for completeness.

Returns:
NA

See also:
AutoScale()

Example:

MatchMin3(int $a, int $b, int $c, Real $weight)

Parameters:

$a	Value a
$b	Value b
$c	Value c
$weight	Weight for value c

Description:
Performs a weighted 3 way minimum, i.e. find the minimum of the three values a,b,c. The weight is used to give the $c value a certain preference.

This is completely an internal routine and should never be called. Only documented for completeness.

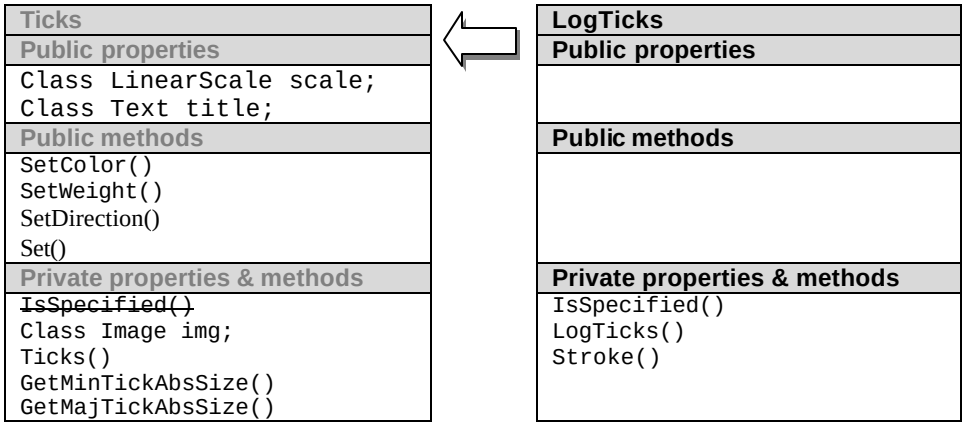
Returns:
The minimum value

See also:
CalcTicks()

Example:

6.3.8 Class LogTicks

Defined in file: jpgraph.php
Extends Ticks.



General description
Calculates the tick marks for a logarithmic scale. This differs from the LinearTicks in that ticks can't be set manually. They are always calculated to be on even logs.

LogTicks()

Parameters:
NA

Description:
Creates a new logarithmic tick

Returns:
NA

See also:
LinearScale

Example:

IsSpecified()

Parameters:
NA

Description:
Determines if the ticks has been manually specified or not.

Returns:
NA

See also:

Example:

Stroke(Class Image &$img,Class LogScale &$scale, int $pos)

Parameters:

$img	Image class to use
$scale	logarithmic sacel to which the ticks belong

$pos	Which side of the axis the ticks go, -1, 1

Description:

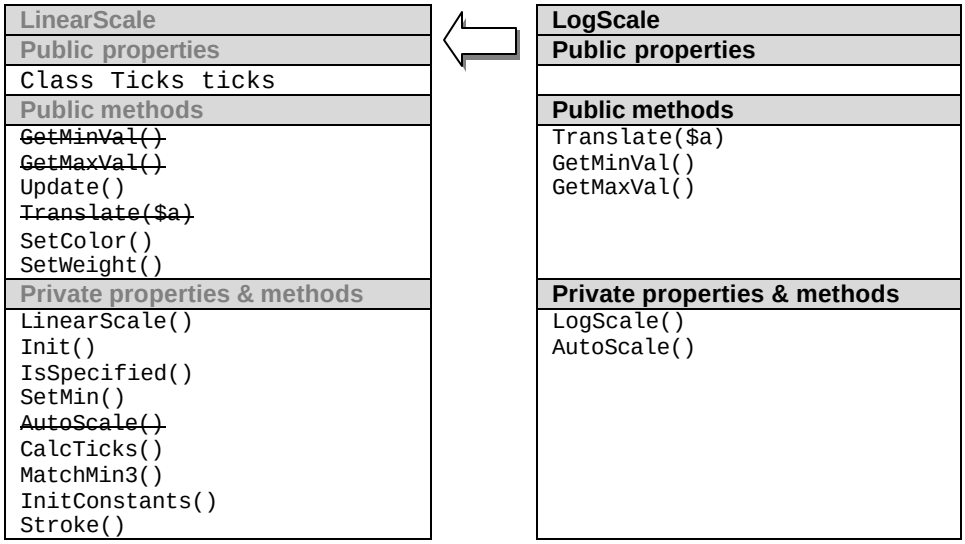
Returns:
NA

See also:

Example:

6.3.9 Class LogScale

Defined in file: jpgraph.php
Extends LinearScale.



General description

Represents a logarithmic scale. Note that plots which has an Y-value of 0 and is added to an Y axis with a logarithmic scale will be automatically adjusted to 1.

LogScale(int $min, int $max, String $type="y")

Parameters:

$min	Minimum value in whole number logs
$max	Maximum value in whole number logs
$type	X or Y axis

Description:

Creates a new logarithmic scale between the given limits. Note that the limits should be given in logs!

Returns:

NA

See also:

Example:

```
$l = new LogScale(0,2); // create a new Y scale between 1 and 100
```

int Translate(Real $a)

Parameters:

$a	World coordinate to be translated

Description:

Translate a world coordinate to screen coordinate.

Returns:

Sceren coordinate

See also:

Example:

```
$pix = $scale->Translate(110,7);
```

Real GetMinVal()

Parameters:
NA

Description:
Get lowest value on scale

Returns:
Lowest value

See also:
LogScale::GetMaxVal()

Example:

Real GetMaxVal()

Parameters:
NA

Description:
Get highest value on scale

Returns:
Highest value

See also:
LogScale::GetMinVal()

Example:

AutoScale(Class Image &$img, Real $min, Real $max, int $maxsteps)

Parameters:

$img	Drawing image
$min	Min value of plots
$max	Max value in plots
$maxsteps	Maximum number of major steps

Description:
Determines the best fit log scale to accommodate both $min and $max values.

Notes this is an internal routine and should never be called directly by a user of this library.

Returns:
NA

See also:

Example:

6.3.10 Class Legend

Defined in file: jpgraph.php

Public properties
Public methods
SetColor() Hide() SetShadow() SetLayout() SetFont() Pos() SetBackground() Add()
Private properties & methods
Stroke()

General description
Defines the appearance of the legend box in the plot. The legend box contains all the legends specified for each plot in the graph. The legend box can have both horizontal and vertical layout.

Instantiated
$graph->legend

Legend()

Parameters:
NA

Description:
Create the legend. Note internal class should never be instantiated by a user class.

Returns:
NA

See also:

Example:

SetShadow(Boolean $f=true, int $width=2)

Parameters:

$f	Shadow on/off
$width	Shadow width

Description:
Specify if the legend box should have a drop shadow or not. Default is on.

Returns:
NA

See also:

Example:

SetLayout(int $l=LEGEND_VERT)

Parameters:

$l	Determine horizontal or vertical layout

Description:
Determine if the text legend should be layout as stacked on top of each other (default) or horizontally beside each other. Legal values for $l are

?? LEGEND_VERT
?? LEGEND_HOR

Returns:
NA

See also:

Example:

SetFont(int $size, string $font="internal")

Parameters:

`$size`	`Font size`
`$font`	`Font family`

Description:
Specify font for legends. See section 5.2 for legal values of $size

Returns:
NA

See also:

Example:

Pos(Real $x, Real $y, String $halign="right", String $valign="top")

Parameters:

$x	X-coordinate in percent of image width
$y	X-coordinate in percent of image height
$haling	How to interpret the X-coord
$valign	How to interpret the Y-coord

Description:
Specify the position of the legend box.

Returns:
NA

See also:

Example:

SetBackground(Color $color)

Parameters:

$color	Color

Description:
Specify background color for the legend box.

Returns:
NA

See also:

Example:

Add(String $txt, Color $color)

Parameters:

$txt	Legend text to be added
$color	Color of marker

Description:
Add a text legend to the legend box.

Note this is a private method that never should be called by a user libray directly. If you want a plot to have a legend use the SetLegend() method for that plot.

Returns:
NA

See also:

Example:

Stroke(Class Image &$img, Class LinearScale &$xscale, Class LinearScale &$yscale)

Parameters:

$img	Image to be drawn to
$xscale	X-scale used for graph
$yscale	Y-scale used for graph

Description:
Stroke the legend to the graph.

Note internal routine and should never be directly called by a user of this library.

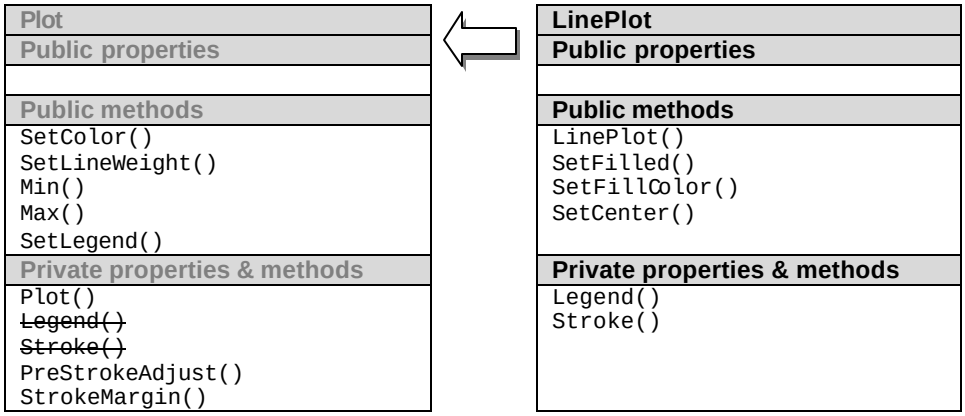
Returns:
NA

See also:

Example:

6.3.11 Class LinePlot

Defined in file: jpgraph_line.php
Extends Plot.



General description
The concrete class which implements a standard line plot.

LinePlot(Array Real &$datay, Mix $datax=false)

Parameters:

$datay	Y-values
$datax	Possible X-values

Description:
Create a line plot.

Returns:
NA

See also:

Example:

SetFilled(Boolean $f=true)

Parameters:

$f	TRUE/FALSE

Description:
Determine if the line plot should be filled. The fill color is specified through the SetFillColor() method.

Returns:
NA

See also:
SetFillColor()

Example:

SetColor(Color $color)

Parameters:

$color	Color

Description:
Specify line color.

Returns:
NA

See also:
SetFillColor()

Example:

SetFillColor(Color $color, Boolean $f=true)

Parameters:
$color Fill color
$f Should the line plot be filled ort not

Description:

Returns:
NA

See also:

Example:

Legend(Class Graph &$graph)

Parameters:
$graph Class Graph

Description:
Framework method. Gets called by framework to set the legend. Note if the plot is filled then the fill color will be used as the legend color, otherwise the line color will be used.

Internal method. Should never be called by a user of this library.

Returns:
NA

See also:

Example:

SetCenter($f=true)

Parameters:
$f Specify if x-scale ticks used with a "text" scale should be centered.

Description:
When using a text scale by default the first tick mark will coincide with the Y-axis and hence the first data point will have it's x-coordinate the same as the Y-axis. This is not always aesthetic pleasing. To change this call SetCenter() this will make each tick-mark be placed in the center of it's "tick-slot" and will have the effect of adding a vertical margin on the left and right of the plot -area.

See section "Advanced use of JpGraph – Using grace value" for an example.

Returns:
NA

See also:
NA

Example:
```
$lineplot->SetCenter()
```

Stroke(Class Image &$img, Class LinearScale &$xscale, Class LinearScale &$yscale)

Parameters:

$img	Image to draw to
$xscale	X-Scale to use
$yscale	Y-Scale to use

Description:
Stroke the line plot.

Internal method. Should never be called by a user of this library.

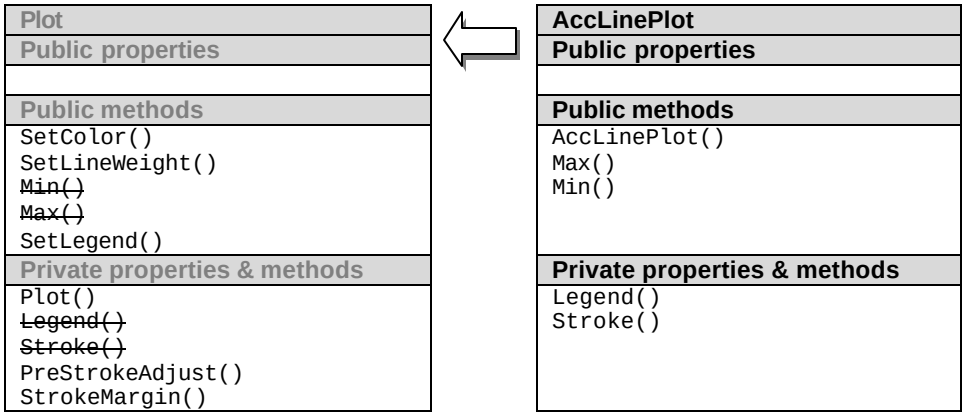
Returns:
NA

See also:

Example:

6.3.12 Class AccLinePlot

Defined in file: jpgraph_line.php
Extends Plot.



General description

The concrete class that implements an accumulated line plot. An accumulated line plot will "staple" each line plot on top of each other using each individual Y point as the distance to the previous line plot, hence the accumulation.

AccLinePlot(Array Class LinePlot $plots)

Parameters:

$plot Array of line plots

Description:

Creates a new accumulated line plot from two or more existing line plots

Returns:

NA

See also:

Example:

```
$l1=new LinePlot($data1y);
$l2=new LinePlot($data2y);
$l3=new LinePlot($data3y);
$al=new AccLinePlot( array($l1, $l2, $l3) );
```

6.3.13 Class PlotMark

Defined in file: jpgraph_line.php

Public properties
Public methods

`PlotMark()`
`SetType()`
`SetColor()`
`SetWidth()`

Private properties & methods
`Stroke()`

General description

This class encapsulates the functionality to draw and position Plot marks in a line or scatter graph. This is an internal class and should normally never be used. You should only access this class through the "mark" instance variable in the line and scatter plot.

Instantiated
$lineplot->mark

PlotMark()

Parameters:
NA

Description:
Create a new mark class.

Returns:
NA

See also:

Example:

SetType(int $t)

Parameters:
$t Specify mark type

Description:
Allowed types are:

?? MARK_SQUARE, A filled square
?? MARK_UTRIANGLE, A upward pointing triangle
?? MARK_DTRIANGLE, A downward pointing triangle
?? MARK_DIAMOND, A diamond shape
?? MARK_CIRCLE, A **non-filled** circle.
?? MARK_FILLEDCIRCLE, A filled circle

Returns:
NA

See also:

Example:

```
$lineplot->mark->SetType(MARK_DIAMOND);
```

SetColor(Color $color)

Parameters:
$color Color

Description:
Specify line color of marks

Returns:
NA

See also:
SetFillColor()

Example:

SetFillColor(Color $color)

Parameters:
$color Color

Description:
Specify fill color for marks

Returns:
NA

See also:
SetColor()

Example:

SetWidth(int $width)

Parameters:
$width Width, in pixels, of mark

Description:
Specify the width, in pixels, of the markl

Returns:
NA

See also:
SetColor()

Example:

Stroke(Class Image &$img, int $x, int $y)

Parameters:

```
$img   Image to draw to
$x     X-coordinate in pixels
$y     Y-coordinate in pixels
```

Description:
Draw the mark to the specified image using the given screen coordinates (**not** world coordinates).

Returns:

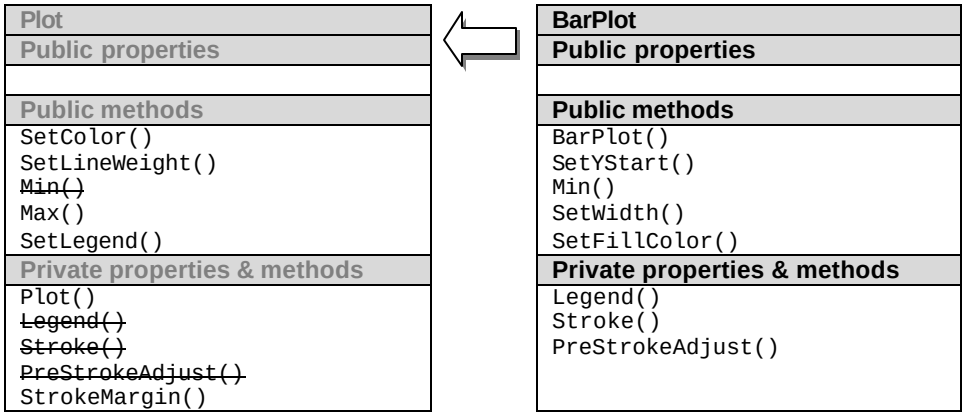
NA

See also:

Example:

6.3.14 Class BarPlot

Defined in file: jpgraph_bar.php
Extends Plot.



General description
Concrete class which implements the standard vertical bar plot functionality.

BarPlot(Array Real &$datay)

Parameters:
$datay Datapoints

Description:
Create a new bar plot from the datapoint given in $datay

Returns:
NA

See also:

Example:

SetYStart(Real $y)

Parameters:
$y Start value in world coordinates

Description:
Sets the Y value to become the base of the bars. Normally this is set to 0. For logarithmic plots this is automatically adjusted to the lowest point on the scale.

Returns:
NA

See also:

Example:

SetWidth(Real $width)

Parameters:
$width In percent of major ticks (0.0-1.0)

Description:

Specify the width of each bar as percentage of with of the major ticks. This means that if you specify a width of 1.0 there will be no gaps between the bars

Returns:
NA

See also:

Example:

SetFillColor(Color $color)

Parameters:
$color Color

Description:
Set fill colors for bars.

Returns:
NA

See also:

Example:

Stroke(Class Image &$img, Class LinearScale &$xscale, Class LinearScale &$yscale)

Parameters:

```
$img          Image to draw to
$xscale       X-scale to be used
$yscale       Y-scale to be used
```

Description:
Draw the bar plot to the specified image.

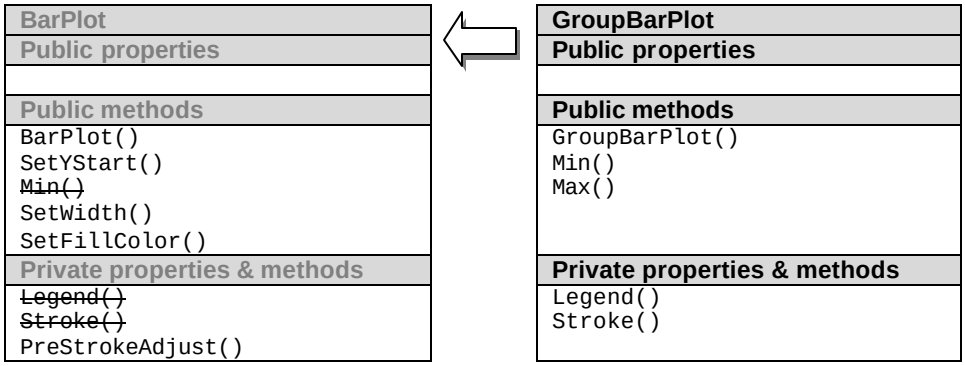
Returns:
NA

See also:

Example:

6.3.15 Class GroupBarPlot

Defined in file: jpgraph_bar.php
Extends BarPlot.



General description

Concrete class which is responsible for constricting a grouped bar plot out of two or more normal Bar Plot. Each bar in a group shares the same X-tick and the group bar is centred around that X-tick. Each bar within the group is given equal width

GroupBarPlot(Array Class BarPlot $plots)

Parameters:

$plot Array of bar plots

Description:

Create a group bar plot from the given individual bar plots. Note that there should normally be the same number of data points for each bar plot.

Returns:

NA

See also:

Example:

Stroke(Class Image &$img, Class LinearScale &$xscale, Class LinearScale &$yscale)

Parameters:

```
$img          Image to draw to
$xscale       X-scale to be used
$yscale       Y-scale to be used
```

Description:

Draw the group bar plot to the specified image.

Returns:

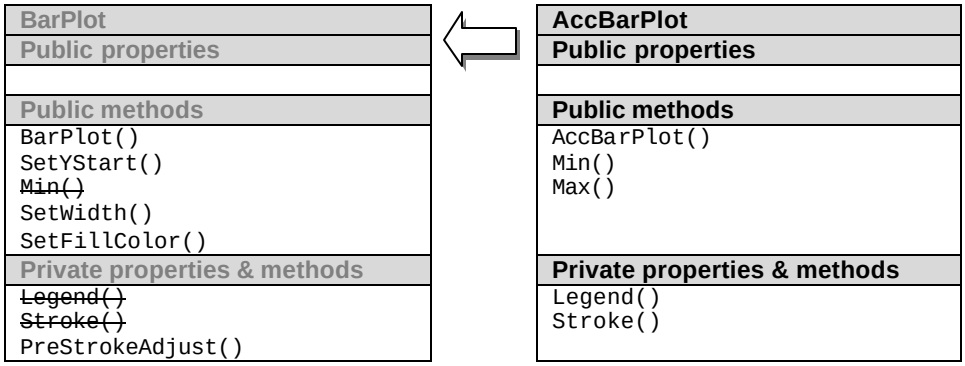
NA

See also:

Example:

6.3.16 Class AccBarPlot

Defined in file: jpgraph_bar.php
Extends BarPlot.



General description
Implements accumulated bar plots, also known as stacked bar plots. Takes two or more normal bar plots and stacks them on top of each other for same X-values. Note the individual bars Y-values are treated as deltas, so for example if two bar plots are added and the first value of each bar is 2 and 3 the resulting stacked bar will have a value of (2+3)=5.

AccBarPlot(Array $plots)

Parameters:
$plot Array of bar plots

Description:
Create an accumulated bar graph plot from two or more other bar plots
Returns:
NA

See also:

Example:

Stroke(Class Image &$img, Class LinearScale &$xscale, Class LinearScale &$yscale)

Parameters:

```
$img          Image to draw to
$xscale       X-scale to be used
$yscale       Y-scale to be used
```

Description:
Draw the accumulated bar plot to the specified image.

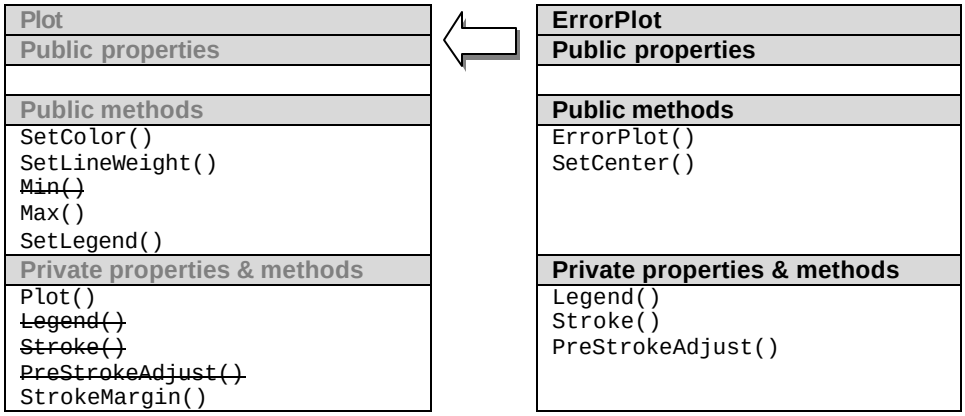
Returns:
NA

See also:

Example:

6.3.17 Class ErrorPlot

Defined in file: jpgraph_error.php
Extends Plot.



General description

Concrete class which implements error plots. Error plots takes two y-values for each X-value, min and max. It then marks each pair of min/max values with a vertical bar.

ErrorPlot(array &$datay, array $datax=false)

Parameters:

$datay Data points,contains 2-Y values for each X-point.
$datax If specified, used as X-coordinates

Description:

Create a new error plot.

Returns:

NA

See also:

Example:

SetCenter(Boolean $c=true)

Parameters:

$c Set center on/off

Description:

Specify if the data points should be place at the left end between each major tick or at the center of the major ticks.

Returns:

NA

See also:

Example:

Stroke(Class Image &$img, Class LinearScale &$xscale, Class LinearScale &$yscale)

Parameters:

```
$img          Image to draw to
$xscale       X-scale to be used
```

`$yscale` `Y-scale to be used`

Description:
Draw the error plot.

Returns:
NA

See also:

Example:

6.3.18 Class Plot

Defined in file: jpgraph.php

Public properties
Public methods
SetColor()
SetLineWeight()
Min()
Max()
SetLegend()
Private properties & methods
Plot()
Legend()
Stroke()
PreStrokeAdjust()
StrokeMargin()

General description

The abstract base class for all plots. All plots inherits from this class. Defines the basic characteristics of a plot.

Plot(Array real &$datay, Mix $datax=false)

Parameters:

$datay	Y values o be plotted
$datax	If specified used as X-coordinates.

Description:

Creates a new plot using the given data vectors. If no X-vector is given the data-points will be numbered sequentially starting with 0.

Returns:

NA

See also:

Example:

Stroke(Class Image &$img, Class LinearScale &$xscale, Class LinearScale &$yscale)

Parameters:

$img	Image to use
$xscale	X-scale to use for the plot
$yscale	Y-scale to use for the plot

Description:

Stroke the plot.

Note internal routine and should never be directly called by a user of this library.

Returns:

NA

See also:

Example:

Legend(Class Graph &$graph)

Parameters:
$graph An instance of GRaph

Description:
Framework method. Gets called to let each individual plot decide what text should be added to the legend. The standard implementation just adds the legend property to the Legend() if the legend is non-empty. This will have a slight different implementation for plot types that manages several plots or several legends. In that case this routine should add each individual legend that should be shown to the $graph->legend.

Note internal routine and should never be directly called by a user of this library.

Returns:
NA

See also:

Example:

PreStrokeAdjust(Class Image &$graph)

Parameters:
$graph Instance of Graph()

Description:
Framework method. Gets called prior to tha stroking of the graph. May be used to make any adjustment to the scales (or ticks) that is needed. For example, in the BarPlot(), this is used to adjust the X-scale so that the bars gets ceneterd in the graph and not normally put close to the left Y-axis.

Note internal routine and should never be directly called by a user of this library.

Returns:
NA

See also:

Example:

SetWeight(int $weight)

Parameters:
$weight Specify weight of graph

Description:
Set the weight for the graph. The actual meaning of this method is determined by the concrete Plot class.

Returns:
NA

See also:

Example:

Min()

Parameters:
NA

Description:

Determine the minimum X and Y value of all points.

Returns:
NA

See also:
Max()

Example:

Max()

Parameters:
NA

Description:
Determine the maximum X and Y value of all points.

Returns:
NA

See also:
Min()

Example:

SetColor(Color $color)

Parameters:
$color Color

Description:
Specify color of the plot. This color will also be used in the legend box.

Returns:
NA

See also:

Example:

SetLegend(String $txt)

Parameters:
$txt Legend text

Description:
Specify legend text for the plot. Any specified text is then automatically added to the legend box.

Returns:
NA

See also:

Example:

SetLineWeight(int $weight=1)

Parameters:
$weight Line weight of the plot

Description:
Specify line weight of the graph The actual meaning of this method is determined by the concrete Plot class.

Returns:
NA

See also:

Example:

StrokeMargin(Class Image &$img)

Parameters:

$img	Image to be used

Description:
Framework method. Gets called after the margin in the graph has been set to its color. Should be used to draw anything in margin. The default implementation does nothing.

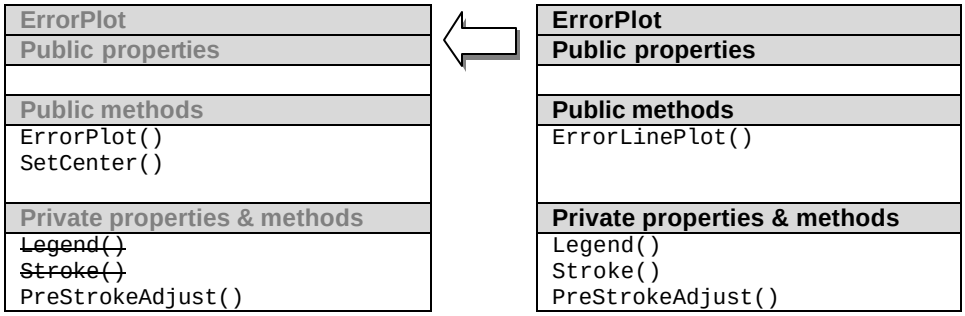
Returns:
NA

See also:

Example:

6.3.19 Class ErrorLinePlot

Defined in file: jpgraph_error.php
Extends ErrorPlot.



General description

The error line plot is much the same as the error plot with the addition of a line between the average value of each error plot pair. The properties of the line may be accessed through the 'line' property of the ErrLinePlot, so for example to draw a red line you issue the statement

$errlineplot->line->SetColor("red");

ErrorLinePlot(array &$datay, array $datax=false)

Parameters:

$datay Data points,contains 2-Y values for each X-point.
$datax If specified, used as X-coordinates

Description:

Create a new error line plot.

Returns:

NA

See also:

Example:

Stroke(Class Image &$img, Class LinearScale &$xscale, Class LinearScale &$yscale)

Parameters:

```
$img          Image to draw to
$xscale       X-scale to be used
$yscale       Y-scale to be used
```

Description:

Draw the error line plot.

Returns:

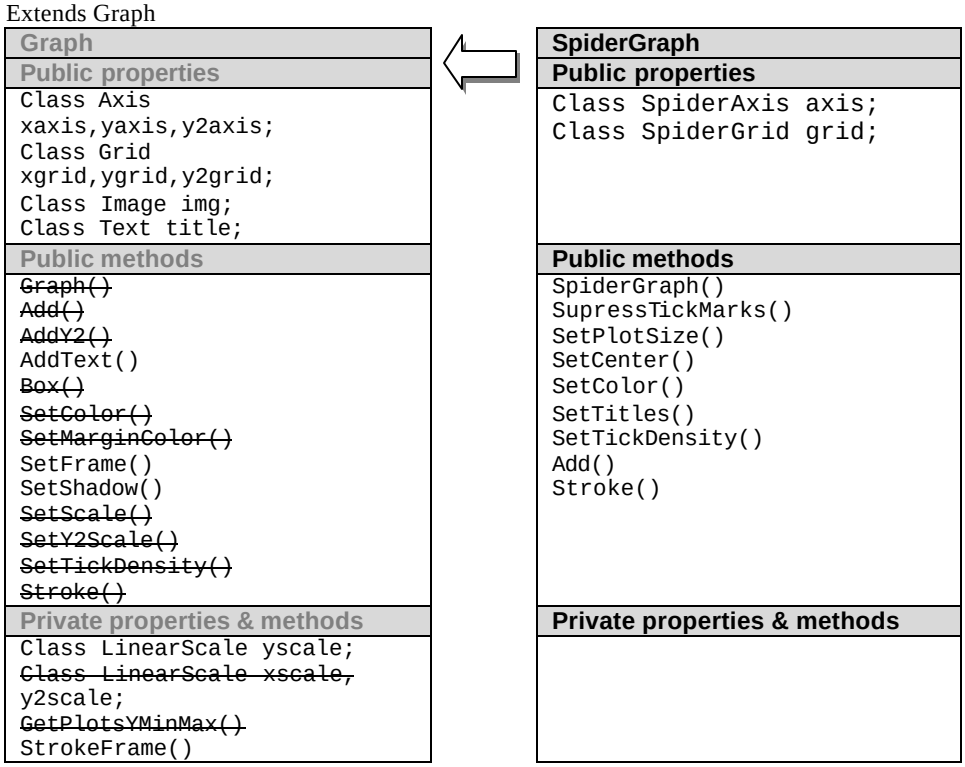
NA

See also:

Example:

6.3.20 Class SpiderGraph

Defined in file: jpgraph_spider.php
Extends Graph



General description

Represent a spider graph. This differs from Graph in that it only contains one scale and one axis which is rotated in a number of copies around the center (set by SetCenter()). The number of axis is equal to the number of datapoints in the plot and hence the angle between each axis is 2*pi/ (nbr of datapoints). The firsr axis is orientated vertically at 90 degrees. Internally the $yscale instance variable is used for the scale of the axis.

SpiderGraph($width=300,$height=200,$cachedName="")

Parameters:

Width	The width of the image used for the graph
Height	The height of the image used for the graph
CachedName	The name of the cached graphic file

Description:

Creates a new image readu for spider plots. If cachedname is given the normal JpGraph cache mechanism will kick in and save the generated image by that name. The next time the image is generated it will first try to locate a cached version of the same name if found it will read it directly from the cache, if not it will be generated.

Returns:

NA

See also:

Graph() Create a regular linear graph.

Example:

```
$graph = new SpiderGraph(300,200);
```

SupressTickMarks($f=true)

Parameters:

$f	TRUE/FALSE	Specify wether ot not tick marks should be shown.

Description:
Determine if tick marks should be displayed on each axis in the spider graph. The default is to turn the ticks off.

Returns:
NA

See also:
NA

Example:
```
$graph->SupressTickMarks();
```

SetPlotSize($size)

Parameters:

$size	Set dimater of the spider plot in percentage.

Description:
Specifies the diameter of the spider plot in terms of min($wifth,$height) of the graph.

Returns:
NA

See also:
SetCenter()

Example:
```
$graph->SetPlotSize(0.7); // 70% of the minimum of width/height
```

SetCenter($px,$py=0.5)

Parameters:

$px	Position in pixels of the center X-coordinate
$py	Position in pixels of the center Y-coordinate

Description:
Specified the center of the spider plot graph in pixels, The default is to place the graph in the center of the image.

Returns:
NA

See also:
SetPlotSize()
Example:

SetColor($color)

Parameters:

$color	Color for background in the graph

Description:
Specify the background color of the graph. The default is white.

Returns:

NA

See also:
NA

Example:
```
$graph->SetColor(“silver”);
```

SetTitles(array $title)

Parameters:
$title Array of titles for each axis.

Description:
Used to specify the title for each of the axis in the spider graph. The number of titles should match the number of data points in the plot (=number of axis in the graph).

Returns:
NA

See also:
NA

Example:
```
$graph->SetTitles(array(“Jan”,”Feb”,”Mar”,”Apr”,”May”,June”);
```

SetTickDensity($densy=TICKD_NORMAL)

Parameters:
$densy Tick density

Description:
Specify the tick density (i.e. how close should the tick marks / labels be on the axis). In spider graph only the vertical axis at 90 degrees have titles. The default setting is (of course) TICKD_NORMAL

Allowed setting are

- ?? TICKD_DENSE
- ?? TICKS_NORMAL
- ?? TICKD_SPARSE
- ?? TICKD_VERYSPARSE

Returns:
NA

See also:
NA

Example:
```
$graph->SetTickDensity(TICKD_SPARSE);
```

Add(&$splot)

Param eters:
$plot A new spider plot

Description:
Add a previously created spier plot to the spider graph. Note that each spider plot is stroked in the order it is added, i.e. the last plot added will go over previously added plots in terms of image depth.

Returns:
NA

See also:
Class SpiderPlot

Example:
```
$plot = SpiderPlot($data);
$graph->Add($plot);
```

GetPlotsYMinMax()

Parameters:
NA

Description:
Return the minimum and maximum value for all the plots in the graph.

Returns:
array($min,$max);

See also:
NA

Example:
NA

Stroke()

Parameters:
NA

Description:
Stroke the defined graph to an image. This call should be the last call in the script since this call will output the graph to the browser (and a cach file if a file name was specified when the graph was created).

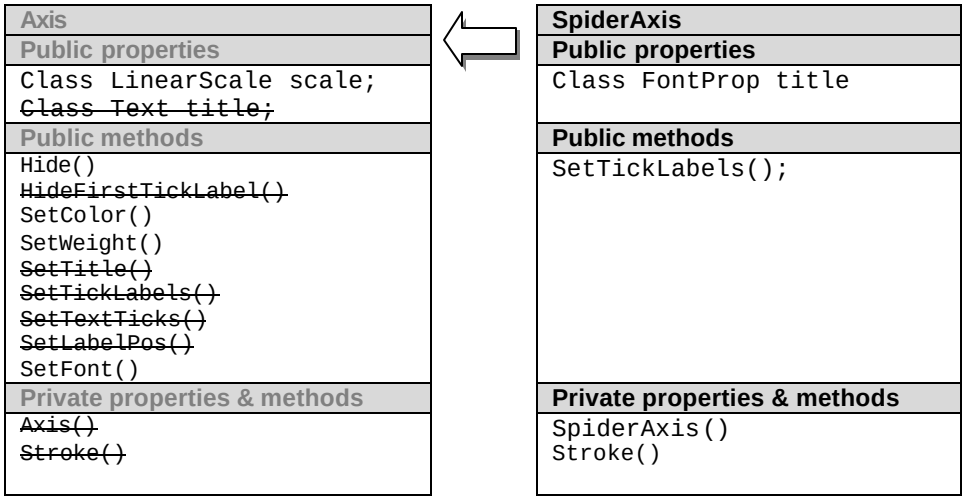
Returns:
NA

See also:
SpiderGraph()

Example:
Trivial.

6.3.21 Class SpiderAxis

Defined in file: jpgraph_spider.php
Extends Axis



General description

Handles the axis in the spider graph. Note thaht even though this class inherits most of the methods from the general Axis class some methods are not supported since it is not suitable for this kind of axis. The striked through methods which doesn't exist in class SpiderAxis are not supported.

SpiderAxis(&$img, &$scale, $color=array(0,0,0))

Parameters:

$img	Image to be drawn to
$scale	Scale to use
$color	Color of axis and labels

Description:
Create a new spider axis. This is an internal (private) routine.

Returns:
NA

See also:
NA

Example:
NA

SetTickLabels($labels)

Parameters:

$labels	Array of tick labels

Description:
Set the tick label array, i.e. the name of the tick labels. By default the normal value will be displayed to the right of the Y-axis.

Returns:
NA

See also:

NA

Example:
NA

Stroke($pos,$angle,&$grid,$title,$draw_label)

Parameters:

$pos	Y-position inpixell of the axis start position
$angle	Whiah angle dhould the axis be drawn at
$grid	Output: Cont ains pair of pixel points for each of the grid points along the axis
$title	Title of the axis
$draw_lable	TRUE if the labels should be draw for this axis

Description:
Stroke the defined axis from the center at angle $angle to the image

Returns:
NA

See also:
NA

Example:
NA

6.3.22 Class SpiderPlot

Defined in file: jpgraph_spider.php

SpiderPlot
Public properties
Public methods
SpiderPlot() Min() Max() SetLegend($legend) SetLineWeight() SetColor()
Private properties & methods
GetCount() Legend() Stroke()

General description

Creates a new spider plot. Each spider plot can only be stroked to a SpiderGraph() throught the use of SpiderGraph::Add() method.

SpiderPlot($data)

Parameters:

$data Array of datapoints

Description:

Create a new spider plot from an array of data points. From each data point an axis is created. Note that for practical purposes the number of data points really should be less then 10-12 points. Otherwise the idea behind spier plots sorts of lose its meaning.

Returns:

NA

See also:

NA

Example:

```
$plot = new SpiderPLot(array(12,36,42,55,19));
```

Min()

Parameters:

NA

Description:

Return the minimum value of all data points for this plot

Returns:

NA

See also:

Max()

Example:

```
$max = $plot->Min();
```

Max()

Parameters:
NA

Description:
Return the maximum value of all data points for this plot

Returns:
NA

See also:
Min()

Example:
```
$max = $plot->Max()
```

SetLegend($legend)

Parameters:

$legend	Legend string

Description:
Specify legend for this plot. This is a text string that will be automatically added to the legend box.

Returns:
NA

See also:
NA

Example:
```
$plot->SetLegend("Defect Goal");
```

SetLineWeight($w)

Parameters:

$w	Weight for plot lines in pixels.

Description:
Specify the weight (width) of the line in the spider plot.

Returns:
NA

See also:
NA

Example:
```
$plot->SetWeight(2);     // Specify the weight to two pixels
```

SetColor($color,$fill_color=array(160,170,180))

Parameters:

$color	Line Color
$fill_color	Fill color

Description:
Specify the color of the spider plot.

Returns:

NA

See also:
NA

Example:
(Trivial.)

GetCount()

Parameters:
NA

Description:
Return number of datapoints in plot.

Returns:
Int NumberOfDataPoints

See also:
NA

Example:
(Trivial)

Legend(&$graph)

Parameters:

$graph	An instance of the spider graph

Description:
This is a framework method that gest called in the SpiderGraph stroke() method. It is used to give each plot a chance to add the appropriate legend string and color to the legend I the graph. This helps the decoupling between the graph class and the plot class.

Returns:
NA

See also:
NA

Example:
NA

Stroke(&$img, $pos, &$scale, $startangle)

Parameters:

$img	Image to stroke to
$pos	Y-coordinate position for startpoint
$scale	Scale to use
$startangle	Startangle for first data point

Description:
Strokes the previously defined spider plot to the griven image. This is an internal method that will be called from SpiderGraph::Stroke()

Returns:
NA

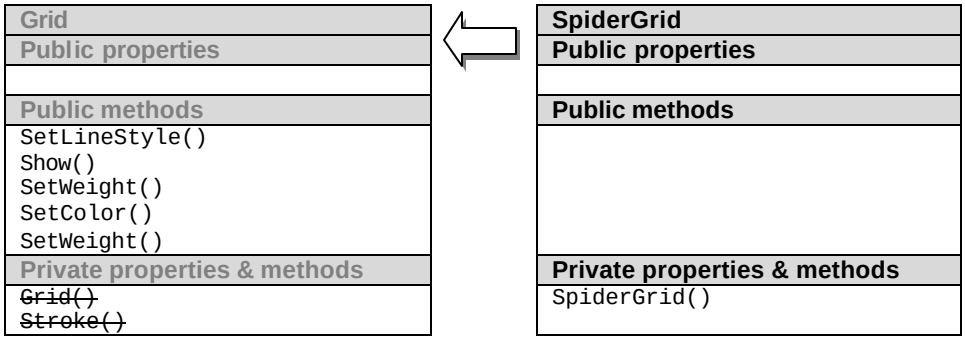
See also:

NA

Example:
NA

6.3.23 Class SpiderGrid

Defined in file: jpgraph_spider.php
Extends Grid



General description

Handles the drawing of grid lines in the spider graph. Inherits all standard properties from Grid()

SpiderGrid()

Parameters:
NA

Description:
Creates a new spider grid. This is internal grid that never should be called directly.

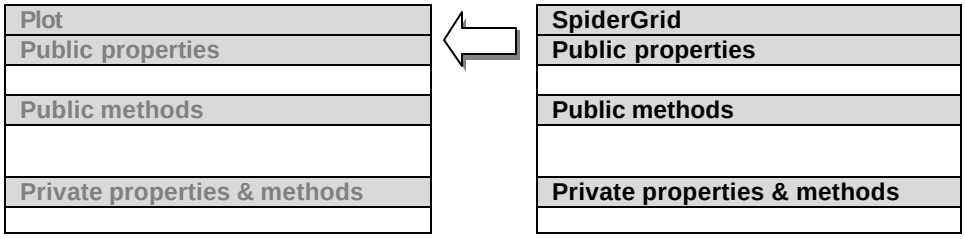
Returns:
NA

See also:
Grid()

Example:
NA

6.3.24 Class ScatterPLot

Defined in file: jpgraph_scatter.php
Extends Plot



General description

ScatterPlot($datay,$datax)

Parameters:
$datay
$datax

Description:
Creates a new scatter plot from the coordinate arrays given.

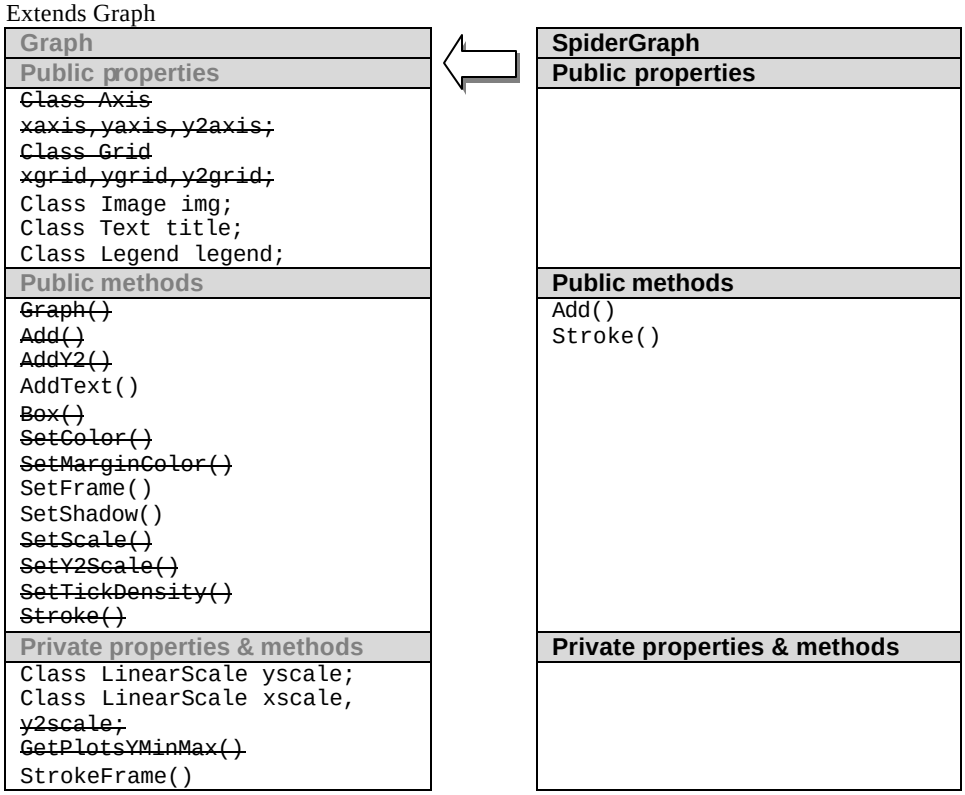
Returns:
NA

See also:

Example:

6.3.25 Class PieGraph

Defined in file: jpgraph_pie.php
Extends Graph



General description

Represent a spider graph. This differs from Graph in that it only contains one scale and one axis which is rotated in a number of copies around the center (set by SetCenter()). The number of axis is equal to the number of datapoints in the plot and hence the angle between each axis is 2*pi/ (nbr of datapoints). The firsr axis is orientated vertically at 90 degrees. Internally the $yscale instance variable is used for the scale of the axis.

PieGraph($width=300,$height=200,$cachedName="")

Parameters:

Width	The width of the image used for the graph
Height	The height of the image used for the graph
CachedName	The name of the cached graphic file

Description:

Creates a new image ready for pie plots plots. If cachedname is given the normal JpGraph cache mechanism will kick in and save the generated image by that name. The next time the image is generated it will first try to locate a cached version of the same name if found it will read it directly from the cache, if not it will be generated.

Returns:

NA

See also:

Graph(), SpiderGraph()

Example:

```
$graph = new PieGraph(300,200);
```

Stroke()

Parameters:
NA

Description:
Sends the created image back to the browser. Should be the latest call in your script since script executionends with this call.

Returns:
NA

See also:
NA

Example:
```
$graph->Stroke();
```

6.3.26 Class PiePlot

Defined in file: jpgraph_pie.php
Extends --

SpiderGrid
Public properties
Class Text title
Public methods
PiePlot() SetCenter() SetSliceColors() SetStartAngle() SetFont() SetSize() SetFontColor() SetLegends() HideLabels() SetPrecision()
Private properties & methods
Legend() Stroke() StrokeLabels()

General description

Creates a new Pie plot from the supplied data. By default each slice will have a label corresponding to the percentage of the sum it Each plot may have an arbitrary title which can be accessed through the "title" property in the PiePlot class. The title will be automatically centred on top of the PiePlot clear of any possible labels. To set the title use the Set() method , i.e. $plot->title->Set("MyTitle")

PiePlot($datay)

Parameters:
$datay

Description:
Creates a new scatter plot from the coordinate arrays given.

Returns:
NA

See also:

Example:

SetCenter($x,$y=0.5)

Parameters:
$ Center y in percentage of height
$y Center x in percentage of width

Description:
Set the center for the pie plot. Default is to be in the center of the image.

Returns:
NA

See also:
SetSize()

Example:
$plot->SetCenter(0.3, 0.4);

SetSliceColors($color)

Parameters:
$color Array of colors to use

Description:
Set an array of colors to use for the different slices. If you have more slices than colors the colors will be rotated from beginning.

Returns:
NA

See also:

Example:
$plot->SetColors(array("blue","green","red","orange"));

SetStartAngle($angle)

Parameters:
$angle Angle in radian (0<$angle<2*PI)

Description:
The first slice normally start at 0 degree. This method lets you specify at which angle the first slice should start. Note that the angle is specified in radians.

Returns:
NA

See also:

Example:
$plot->SetStartAngle(M_PI/4); // Start at 45 degree angle

SetFont($font_size, $font="internal")

Parameters:
$font_size Set Font size
$font Font type

Description:
Specify font for labels

Returns:
NA

See also:

Example:
$plot->SetFont(FONT1_BOLD);

SetSize($size)

Parameters:
$size Size in percentage of the minimum of the height or width

Description:

Set the radius of the pie plot in percentage of the minimum of the width and height of the image.

Returns:
NA

See also:

Example:
$plot->SetSize(0.3);

SetFontColor(Color $color)

Parameters:
$color Color

Description:
Specify color for labels on the pie plot

Returns:
NA

See also:

Example:

SetLegends(Array $legends)

Parameters:
$legends Array of legends for each pie slice

Description:

Returns:
NA

See also:

Example:

HideLabels(Boolean $f=true)

Parameters:
$f TRUE = Hide labels

Description:
Specify wheter or not labels should be displayed.

Returns:
NA

See also:
SetFontColor(), SetFont()

Example:

SetPrecision(int $prec, Boolean $psign=true)

Parameters:

$prec	Number of digits precision for the labels of the pie plot
$psign	TRUE if each label should have an ending '%' sign

Description:
Specified to what precision the labels should be displayed..

Returns:
NA

See also:
NA

Example:
$p1->SetPrecision(2);

Stroke(&$img)

Parameters:

$img	Image to stroke to.

Description:
Internal method should never be called directly. Stroke the pie plot to the specified image.

Returns:
NA

See also:

Example:

StrokeLabels($label,$img,$xc,$yc,$a,$r)

Parameters:

$label	Text label to print
$img	Image to print to
$xc	X-coordinate for Center of pie chart
$yc	Y-coordinate for Center of pie chart
$a	Angle to plot label at
$r	Radius of pie plot

Description:
Draws the labels for each slide. Normally the angle is choosen to be in the middle of the slice. Internal method and should never be called directly.

Returns:
NA

See also:

Example:

1.4 Internal class reference

Note: All the following classes are internal to JpGraph and should never be instantiated from clients to JpGraph. They are only documented for completeness and for those who whish to extend JpGraph.

6.3.27 Class ImgStreamCache

Defined in file: jpgraph.php

Public properties
Public methods

`ImgStreamCache()`
`PutAndStream()`
`GetAndStream()`

Private properties & methods

General description

This is an internal class which is used by the Graph to handle streaming and caching of the generated image. This class should never be instantiated by a user of the library. It is only documented here for completeness.

ImgStreamCache(Class Image &$img, String $cacheDir=CACHE_DIR)

Parameters:

$img	Image to be streamed
$cacheDir	Cache directory to look for potentially cashed version of the image

Description:

Internal class to Image which handles the streaming and pot ential caching of images to file.

Returns:

NA

See also:

Example:

PutAndStream(Class Image &$img, String $fileName)

Parameters:

$img	The image to stream
$filename	Filename of cashed version

Description:

If filename is given then the image will be stored in the cache under that name. The image wil then be streamed back to the browser.

Note1 that this should be the last call since nothing else can be sent back to the browser after this call.

Note2 This is an internal method that never should be called directly by a user of this library.

Returns:

NA

See also:

Example:

GetAndStream(String $fileName)

Parameters:

$filename	File name

Description:

Tries to find the file with specified name and then stream that file as an image back to the browser.

Returns:
False if no file was found.

See also:

Example:

6.3.28 Class Image

Defined in file: jpgraph.php

Public properties
Public methods
AddObserver() SetFont() GetFontHeight() GetTextWidth() StrokeText() StrokeBoxedText() SetMargin() SetColor() SetTransparent() SetLineWeight() SetStartPoint() Line() LineTo() Arc() Polygon() FilledPolygon() Rectangle() FilledRectangle() ShadowRectangle() Point() DashedLine() SetImgFormat()
Private properties & methods
Image() NotifyObservers() Headers() Stream() Destroy()

General description

Represent the lowest layer. Contains all the drawing primitives that directly generates an image.

Instantiated
$graph->img

Image(int $width, int $height, String $format="png")

Parameters:

$width	Width in pixel of the generated image
$height	Height in pixel of the generated image
$format	Graphic format for the generated image

Description:

Creates a new image width the specified width and heigh. Depending on the value of $format three different graphic formats are supported

- ?? png
- ?? jpg
- ?? gif

Note that the actual supported formats are dependent on the specific version of the GD library.To use jpg format and additional library must normally also be installed. See documentation on graphic formats in PHP manual.

Returns:

A handle to the newly created image

See also:

Example:

AddObserver(String $meth, Object &$obj)

Parameters:

$meth	Name of method to be called
$obj	The object where the method exists.

Description:
Adds an observer to the image class which gets called when basic values are changed, such as the margins of the image. The registred observer will be called with the a reference of the current instance of the Class Image

Returns:
NA

See also:
NotifyOnservers()

Example:
```
$img->AddObserver("InitConstants",&$this);
```

NotifyObservers()

Parameters:
NA

Description:
Calls all previously registered observers for this instance of Image. All the called observers will get called with a reference to the instance of this class as the first parameter.

This is really an internal method that never should be called. It is only described here for completeness.

Returns:
NA

See also:
AddObserver()

Example:
$img>NotifyObservers()

SetFont(int $size, String $name="internal")

Parameters:

`$size`	`Fontname/size`
`$name`	`Type of font`

Description:
Specify the font to be used for a successive call to StrokeFont(). Version 1.0 of JpGraph only supports internal fonts. The available internal fonts are specified with integers between 0-4 or with the symbolic constants according to the table

	Font style	
Size	Regular	Bold
Small	`FONT0`	
Normal	`FONT1`	`FONT1_BOLD`
Large	`FONT2`	`FONT2_BOLD`

Table 1. Available internal fonts.

Returns:
NA

See also:
StrokeFont(), GetFontHeight(),GetFontWidth(),SetColor()

Example:
```
$graph->img->SetColor("darkred");
$graph->img->SetFont(FONT1_BOLD);
$graph->img->StrokeFont(50,20,"Revenue","center");
```

GetFontHeight()

Parameters:
NA

Description:
Return the font height in pixels of the current active font.

Returns:
NA

See also:
GetTextWidth(), SetFont()

Example:

GetTextWidth(String &$txt)

Parameters:
```
$txt   text string
```

Description:
Returns the width in pixel of the entire text string supplied.

Returns:
Textwidth in pixels

See also:
SetFont()

Example:

StrokeText(int $x, int $y, String $txt, String $halign="left", String $dir="h")

Parameters:
```
$x          Horizontal coordinate for text
$y          Vertical coordinate for text
$txt        Text to be stroked
$haling     Horizontal alignment
$dir        Direction (Horizontal or vertical)
```

Description:
Draws the specified text string at the specified position. Depending on the value of $haling the x-coordinate is interpret as:

$haling	$x
"Left"	Interpret as the left edge of the textstring
"Center"	Interpret as the center of the text string

“Right”	Interpret as the right edge of the textstrin g

Returns:
NA

See also:
SetFont(), GetFontHeight(), GetTextWidth()

Example:
```
$graph->img->StrokeText(50,20,”My first title”,”center”);
```

StrokeBoxedText(int $x, int $y, String $txt, String $halign, String $dir, Color $fcolor, Color $bcolor, Boolean $sha dow=false)

Parameters:
```
$x          Horizontal coordinate for text
$y          Vertical coordinate for text
$txt        Text to be stroked
$haling     Horizontal alignment
$dir        Direction (Horizontal or vertical)
$fcolor     Fill color, if false no fill color will be used
$shadow     TRUE if the box should have a drop shadow
```

Description:
Similar to StrokeText() but this method draws, a possible filled, box around the text. The box may also have a drop shadow.

Returns:
NA

See also:
SetFont(), GetFontHeight(), GetTextWidth(), StrokeText()

Example:

SetMargin(int $lm, int $rm, int $tm, int $bm)

Parameters:
```
$lm    Left margin in pixels
$rm    Right marging in pixels
$tm    Top margin in pixels
$bm    Bottom margin in pixels
```

Description:
Specifies the margin area between the plot-area and the end of the image. The margin should be big enough to hold any titles, labels or other text you want to be visible there.

Returns:
NA

See also:

Example:
```
$graph->img->SetMargin(20,20,30,30);
```

SetColor(Color $color)

Parameters:
```
$color      Color
```

Description:

Specify drawing color for the following draw primitives. All consecutive calls to Line(), Rectangle(), Arc() etc. will be drawn using this color. A color may be specified either as the RGB-triple or as one of the predefined color names. Se chapter XX for a list of pre-defined color names.

Note. You should never call this function directly from user code since all defined drawing object (e.g. LinePlot()) have a SetColor() method which saves each objects own color which is then set (using this method) before the object is stroked to the image.

Returns:
NA

See also:
SetTransparent()

Example:
```
$graph->img->SetColor("red");
// or SetColor(array(255,0,0)) or SetColor(array(#FF,0,0))
$grph->img->Line(0,0,10,10);
```

SetTransparent(Color $color)

Parameters:

`$color`	`Transparent color`

Description:
Specify which color should be transparent. Note that if you use a shadow on the image the upper right "non-shadow" and the lower left "non-shadow" will always default to color white. This means that if your page has a background you shold normally specify white as transparent to avoid a small white area at the corner of the shadow.

Returns:
NA

See also:
SetColor()

Example:
```
$graph->img->SetTRansparent("white");
```

SetLineWeight(int $weight))

Parameters:

$weight	Line weight in pixels

Description:
Specify the line weight for Line(), LineTo() methods.

Note that the line weight will not be applied to Rectangle(), FilledRectangle(), Arc()

Returns:
NA

See also:

Example:
```
$img->SetLineWeight(2);
$img->Line(0,0,200,100);
```

SetStartPoint(int $x, int $y)

Parameters:

$x x-coordinate
$y y-coordinate

Description:
Specify a start x-y-point for the next LineTo() call.

Returns:
NA

See also:
LineTo()

Example:
```
$img->SetStartPoint(10,10);
$img->LineTo(100,100);    //Draw a line between (10,10) and (100,100)
```

Arc(int $cx, int $cy, int $width, int $height, int $start, int $end)

Parameters:

$cx	Center x-coordinate
$cy	Center y-coordinate
$width	Width of arc in pixels
$height	Height of arc in pixels
$start	Start angle (in degrees)
$end	End angle (in degrees)

Description:
Draw an arc with the given coordinates and specifications.

Returns:
NA

See also:

Example:
```
$img->Arc(100,100,25,25,0,360);  // Draw a circle width radius=25 pixels
```

Line(int $x1, int $y1, int $x2, int $y2)

Parameters:

$x1,$y1	Start point
$x2,$y2	End point

Description:
Draw a line between the specified coordinates.

Returns:
NA

See also:
SetLineWeight()

Example:
```
$img->Line(0,0,100,100);
```

Polygon(Array int $points)

Parameters:

$points	Array of coordinates

Description:
Draws an polygon between all the data points specified in the array “$points”

Returns:
NA

See also:
SetColor(), FilledPolygon()

Example:
```
$pnts = array(0,0,10,15,12,15,12,30,40,30);
$img->Polygon($pnts);
```

FilledPolygon(Array int $points)

Parameters:

$points	Array of coordinates

Description:
Draws a filled polygon between all the data points specified in the array “$points”

Returns:
NA

See also:
Polygon(), SetColor()

Example:
```
$pnts = array(0,0,10,15,12,15,12,30,40,30);
$img->FilledPolygon($pnts);
```

Rectangle(int $xl, int $yu, int $xr, int $yl)

Parameters:

$xl, $yu	Upper left corner
$xr, $yl	Lower right corner

Description:
Draw a rectangle

Returns:
NA

See also:
SetColor()

Example:
```
$img->Rectangle(20,10,50,60);
```

FilledRectangle(int $xl, int $yu, int $xr, int $yl)

Parameters:

$xl, $yu	Upper left corner
$xr, $yl	Lower right corner

Description:
Draw a filled rectangle

Returns:
NA

See also:
Rectangle()

Example:
```
$img->FilledRectangle(20,10,50,60);
```

ShadowRectangle(int $xl, int $yu, int $xr, int $yl, Boolean $fcolor=false, int $shadow_width=3, Color $shadow_color="gray40")

Parameters:

$xl, $yu	Upper left corner
$xr, $yl	Lower right corner
$fcolor	Fill color of rectangle
$shadow_width	Width of shadow
$shadow_color	Color of shadow

Description:
Draws a filled rectangle with a shadow. If fcolor=false then no fill color will be used.

Returns:
NA

See also:
Rectangle(), Filledrectangle()

Example:

LineTo(int $x, int $y)

Parameters:

$x,$y	End coordinate for the line

Description:
Draw a line between the previous end point for previous LineTo() to the point specified as parameter. The previous start point may also be specified with a call to SetStartPoint()

Returns:
NA

See also:
SetStartPoint()

Example:

Point(int $x, int $y)

Parameters:

$x,$y	End coordinate for the line

Description:
Set a single pixel.

Returns:
NA

See also:
SetColor()

Example:

DashedLine(int $x1, int $y1, int $x2, int $y2, int $dash_length=1, int $dash_space=4)

Parameters:

$x1,$y1	Start point
$x2,$y2	End point
$dash_length	Length, in pixel, of line segment
$dash_space	Spec, inpixels, between line segments

Description:
Draws a dashed line with the specified parameters.

Note that this is a much more computationally expensive then drawing a straight line with either LineTo() or Line()

Returns:
NA

See also:
Line(), LineTo(), SetColor()

Example:
```
$img->DashedLine(0,0,30,50); // Draws a “dotted” line
```

Headers ()

Parameters:
NA

Description:
Internal method. Should never ever be called by a client. Only documented for completeness.

Outputs the necessary headers to the browser in preparation to send the raw binary data that represents the image.

Implementation note: If you look at the implementation of Headers() you find that it is possible to output two versions of the header, one simple and one slightly more complicated. This is controlled by the instance variable $this->expired .

If this instance variable is true the header output will try to tell the browser not to cache the image, note that this is not foolproof since there is no standard way of guaranteeing the no-caching in browser.

The default value of $expired is TRUE.

Returns:
NA

See also:

Example:

Destroy()

Parameters:
NA

Description:
Returns resources allocated when the image was created.

Note. This is normally not used when generating on-line images but useful to free resources when images are just generated to files.

Returns:
NA

See also:
NA

Example:

Stream(Stream $file="")

Parameters:

`$file`	`File name to save image in`

Description:
Streams the generated file either to a specified file (if parameter given) or directly back to the browser if no file name has been supplied.

Returns:
NA

See also:

Example:

```
$img->Stream(“exmaple1.png”);     // Save the generated image in a file
```

SetImgFormat(String $format)

Parameters:

$format	Specifies graphic format

Description:
Specify the graphic format to be used.
This is a low level internal method. Should not be called directly. The graphic format is normally specified when creating an instance of the Image() class.

Allowed graphic formats are:

- ?? png
- ?? jpg
- ?? gif

Returns:

TRUE	If the graphic format is supported by the installation of PHP
FALSE	Otherwise

See also:
Graph()

Example:

6.3.29 Class TTF

Defined in file: jpgraph.php
Extends --

TTF
Public properties
Public methods
Private properties & methods

General description

Handles loading of TTF font files and translation to specific TTF file names. This is an internal class and should never be used directly by clients to JpGraph library.

TTF()

Parameters:
NA

Description:
Initiates TTF fonts by setting the corresponfing file names.

Returns:
NA

See also:

Example:

MethodName()

Parameters:

Description:

Returns:
NA

See also:

Example:

6.3.30 Class Gradient

Defined in file: jpgraph_bar.php
Extends --

Gradient
Public properties
Public methods
Private properties & methods

General description
Handles all aspects of Color gradient fill. Internal class.

MethodName()

Parameters:

Description:

Returns:
NA

See also:

Example:

MethodName()

Parameters:

Description:

Returns:
NA

See also:

Example:

6.3.31 Class RGB

Defined in file: jpgraph.php

Public properties
Public methods

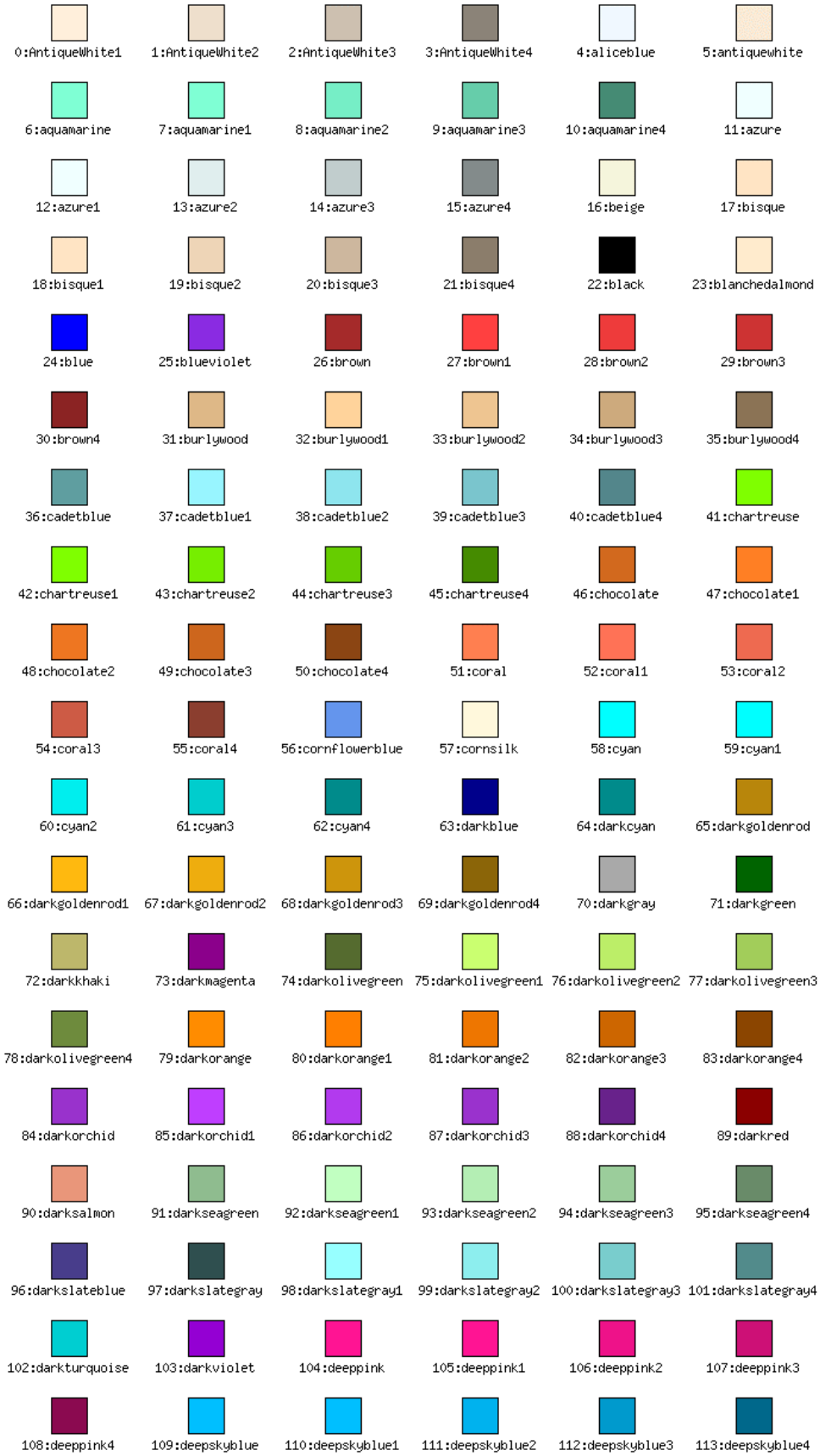
`RGB()`
`Color()`
`Allocate()`

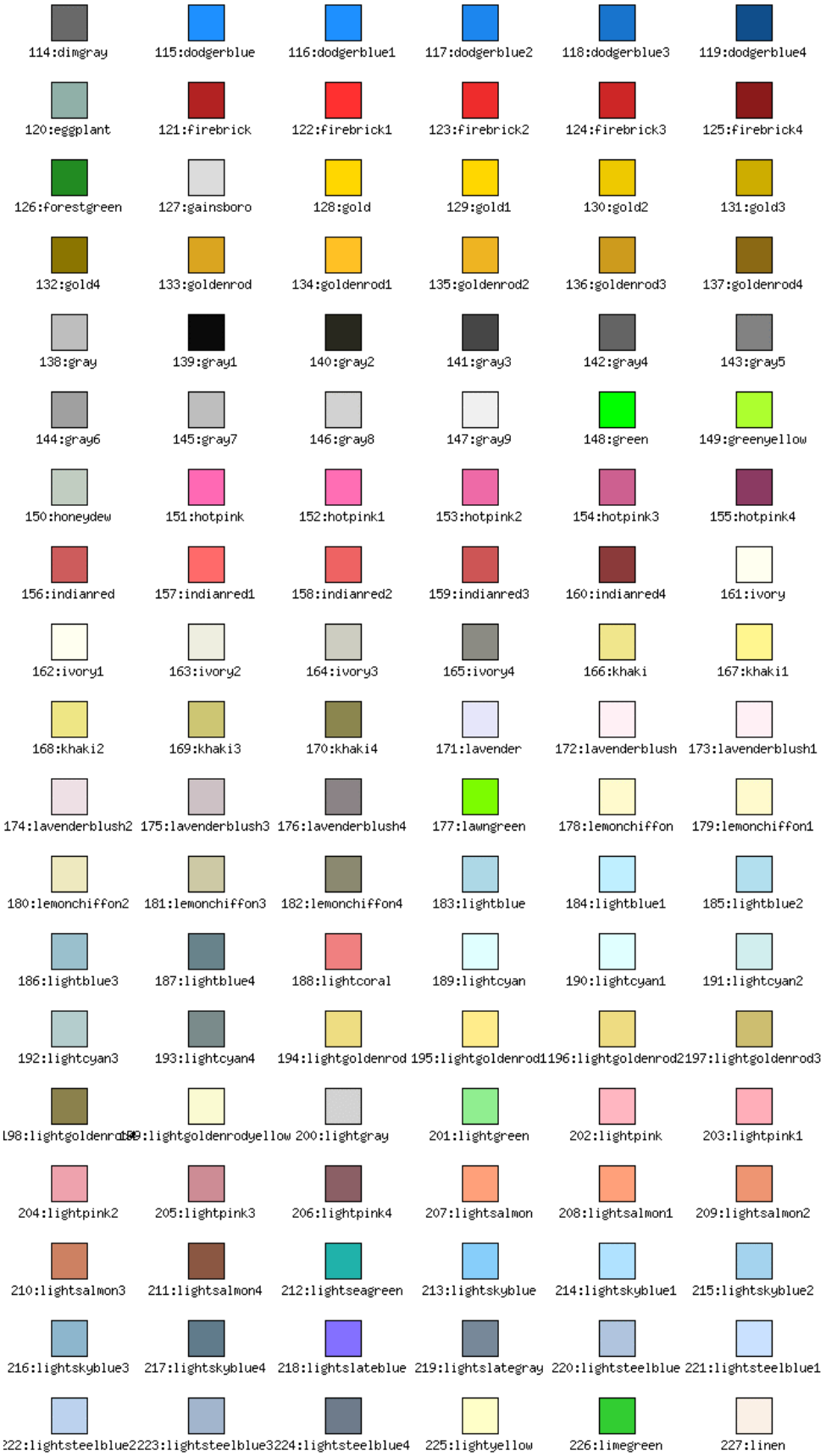
Private properties & methods

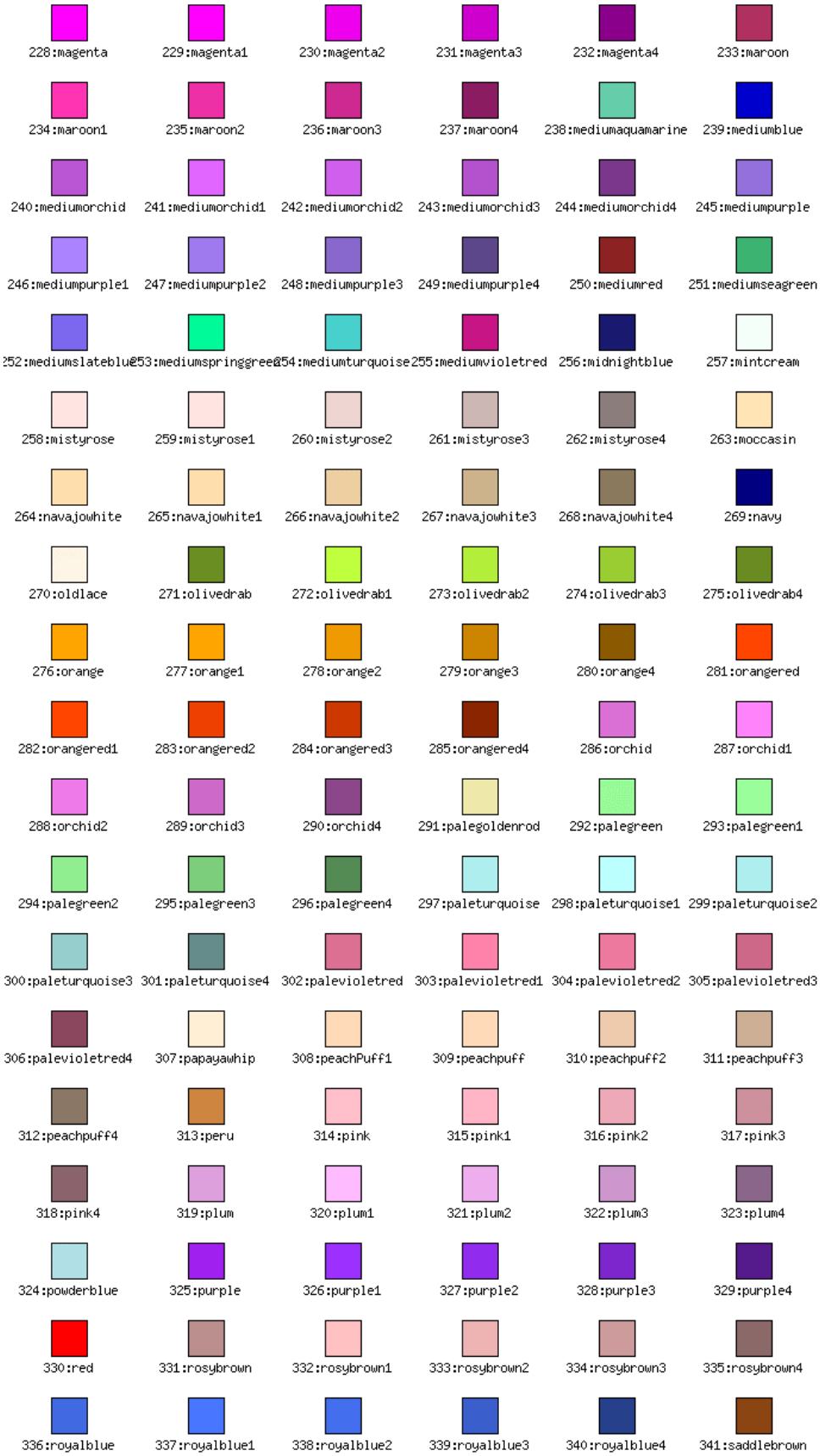
General description

Defines symbolic color names and handles allocation of colors in the image. This is an internal class used by Image.

The following colors are predefined any other color can be specified by giving it's RGB triple as the argument to any SetColor() method.


0:AntiqueWhite1
1:AntiqueWhite2
2:AntiqueWhite3
3:AntiqueWhite4
4:aliceblue
5:antiquewhite
6:aquamarine
7:aquamarine1
8:aquamarine2
9:aquamarine3
10:aquamarine4
11:azure
12:azure1
13:azure2
14:azure3
15:azure4
16:beige
17:bisque
18:bisque1
19:bisque2
20:bisque3
21:bisque4
22:black
23:blanchedalmond
24:blue
25:blueviolet
26:brown
27:brown1
28:brown2
29:brown3
30:brown4
31:burlywood
32:burlywood1
33:burlywood2
34:burlywood3
35:burlywood4
36:cadetblue
37:cadetblue1
38:cadetblue2
39:cadetblue3
40:cadetblue4
41:chartreuse
42:chartreuse1
43:chartreuse2
44:chartreuse3
45:chartreuse4
46:chocolate
47:chocolate1
48:chocolate2
49:chocolate3
50:chocolate4
51:coral
52:coral1
53:coral2
54:coral3
55:coral4
56:cornflowerblue
57:cornsilk
58:cyan
59:cyan1
60:cyan2
61:cyan3
62:cyan4
63:darkblue
64:darkcyan
65:darkgoldenrod
66:darkgoldenrod1
67:darkgoldenrod2
68:darkgoldenrod3
69:darkgoldenrod4
70:darkgray
71:darkgreen
72:darkkhaki
73:darkmagenta
74:darkolivegreen
75:darkolivegreen1
76:darkolivegreen2
77:darkolivegreen3
78:darkolivegreen4
79:darkorange
80:darkorange1
81:darkorange2
82:darkorange3
83:darkorange4
84:darkorchid
85:darkorchid1
86:darkorchid2
87:darkorchid3
88:darkorchid4
89:darkred
90:darksalmon
91:darkseagreen
92:darkseagreen1
93:darkseagreen2
94:darkseagreen3
95:darkseagreen4
96:darkslateblue
97:darkslategray
98:darkslategray1
99:darkslategray2
100:darkslategray3
101:darkslategray4
102:darkturquoise
103:darkviolet
104:deeppink
105:deeppink1
106:deeppink2
107:deeppink3
108:deeppink4
109:deepskyblue
110:deepskyblue1
111:deepskyblue2
112:deepskyblue3
113:deepskyblue4


114:dimgray
115:dodgerblue
116:dodgerblue1
117:dodgerblue2
118:dodgerblue3
119:dodgerblue4
120:eggplant
121:firebrick
122:firebrick1
123:firebrick2
124:firebrick3
125:firebrick4
126:forestgreen
127:gainsboro
128:gold
129:gold1
130:gold2
131:gold3
132:gold4
133:goldenrod
134:goldenrod1
135:goldenrod2
136:goldenrod3
137:goldenrod4
138:gray
139:gray1
140:gray2
141:gray3
142:gray4
143:gray5
144:gray6
145:gray7
146:gray8
147:gray9
148:green
149:greenyellow
150:honeydew
151:hotpink
152:hotpink1
153:hotpink2
154:hotpink3
155:hotpink4
156:indianred
157:indianred1
158:indianred2
159:indianred3
160:indianred4
161:ivory
162:ivory1
163:ivory2
164:ivory3
165:ivory4
166:khaki
167:khaki1
168:khaki2
169:khaki3
170:khaki4
171:lavender
172:lavenderblush
173:lavenderblush1
174:lavenderblush2
175:lavenderblush3
176:lavenderblush4
177:lawngreen
178:lemonchiffon
179:lemonchiffon1
180:lemonchiffon2
181:lemonchiffon3
182:lemonchiffon4
183:lightblue
184:lightblue1
185:lightblue2
186:lightblue3
187:lightblue4
188:lightcoral
189:lightcyan
190:lightcyan1
191:lightcyan2
192:lightcyan3
193:lightcyan4
194:lightgoldenrod
195:lightgoldenrod1
196:lightgoldenrod2
197:lightgoldenrod3
198:lightgoldenrod
199:lightgoldenrodyellow
200:lightgray
201:lightgreen
202:lightpink
203:lightpink1
204:lightpink2
205:lightpink3
206:lightpink4
207:lightsalmon
208:lightsalmon1
209:lightsalmon2
210:lightsalmon3
211:lightsalmon4
212:lightseagreen
213:lightskyblue
214:lightskyblue1
215:lightskyblue2
216:lightskyblue3
217:lightskyblue4
218:lightslateblue
219:lightslategray
220:lightsteelblue
221:lightsteelblue1
222:lightsteelblue2
223:lightsteelblue3
224:lightsteelblue4
225:lightyellow
226:limegreen
227:linen


228:magenta
229:magenta1
230:magenta2
231:magenta3
232:magenta4
233:maroon
234:maroon1
235:maroon2
236:maroon3
237:maroon4
238:mediumaquamarine
239:mediumblue
240:mediumorchid
241:mediumorchid1
242:mediumorchid2
243:mediumorchid3
244:mediumorchid4
245:mediumpurple
246:mediumpurple1
247:mediumpurple2
248:mediumpurple3
249:mediumpurple4
250:mediumred
251:mediumseagreen
252:mediumslateblue
253:mediumspringgree
254:mediumturquoise
255:mediumvioletred
256:midnightblue
257:mintcream
258:mistyrose
259:mistyrose1
260:mistyrose2
261:mistyrose3
262:mistyrose4
263:moccasin
264:navajowhite
265:navajowhite1
266:navajowhite2
267:navajowhite3
268:navajowhite4
269:navy
270:oldlace
271:olivedrab
272:olivedrab1
273:olivedrab2
274:olivedrab3
275:olivedrab4
276:orange
277:orange1
278:orange2
279:orange3
280:orange4
281:orangered
282:orangered1
283:orangered2
284:orangered3
285:orangered4
286:orchid
287:orchid1
288:orchid2
289:orchid3
290:orchid4
291:palegoldenrod
292:palegreen
293:palegreen1
294:palegreen2
295:palegreen3
296:palegreen4
297:paleturquoise
298:paleturquoise1
299:paleturquoise2
300:paleturquoise3
301:paleturquoise4
302:palevioletred
303:palevioletred1
304:palevioletred2
305:palevioletred3
306:palevioletred4
307:papayawhip
308:peachPuff1
309:peachpuff
310:peachpuff2
311:peachpuff3
312:peachpuff4
313:peru
314:pink
315:pink1
316:pink2
317:pink3
318:pink4
319:plum
320:plum1
321:plum2
322:plum3
323:plum4
324:powderblue
325:purple
326:purple1
327:purple2
328:purple3
329:purple4
330:red
331:rosybrown
332:rosybrown1
333:rosybrown2
334:rosybrown3
335:rosybrown4
336:royalblue
337:royalblue1
338:royalblue2
339:royalblue3
340:royalblue4
341:saddlebrown

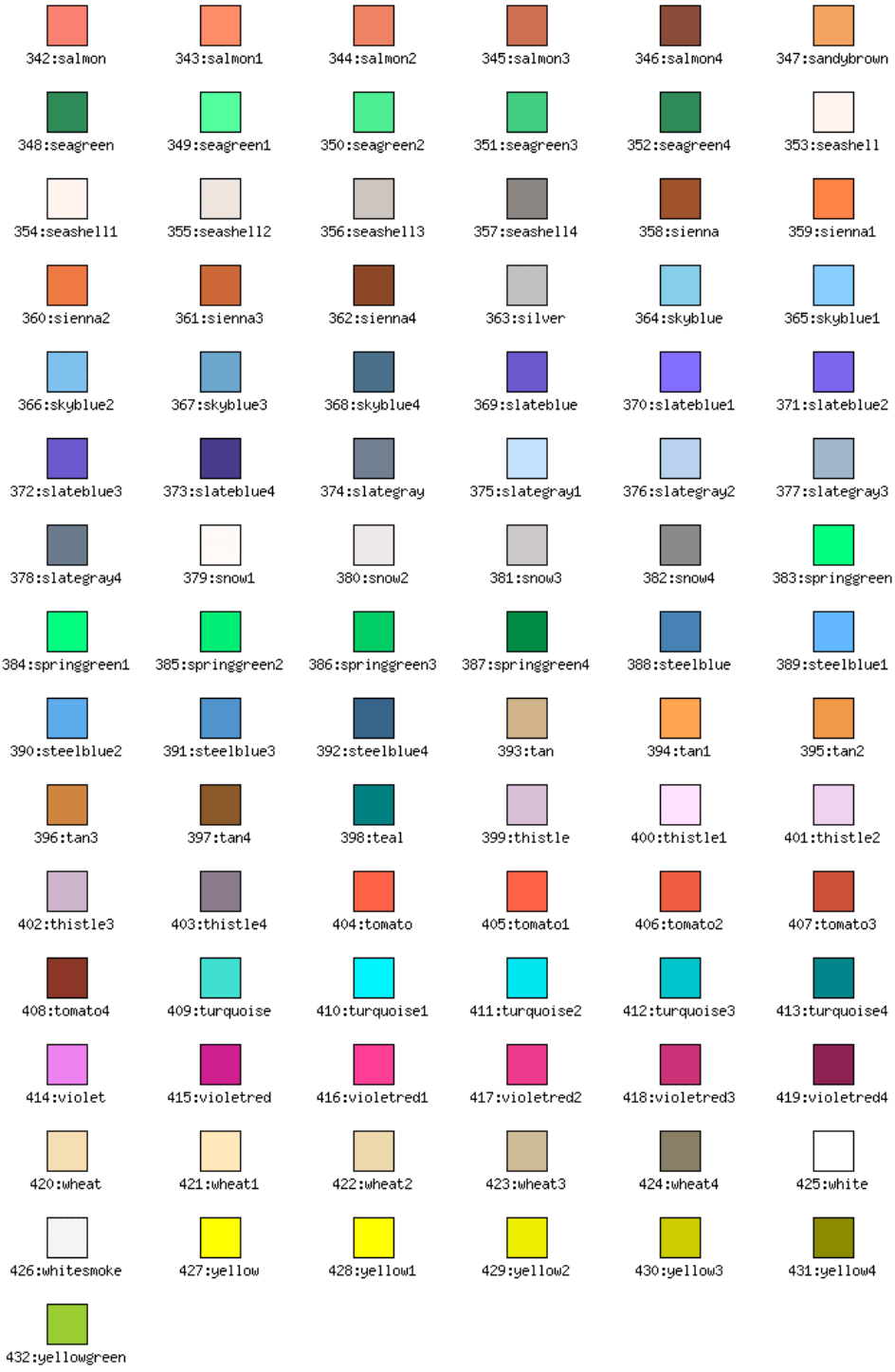

342:salmon
343:salmon1
344:salmon2
345:salmon3
346:salmon4
347:sandybrown
348:seagreen
349:seagreen1
350:seagreen2
351:seagreen3
352:seagreen4
353:seashell
354:seashell1
355:seashell2
356:seashell3
357:seashell4
358:sienna
359:sienna1
360:sienna2
361:sienna3
362:sienna4
363:silver
364:skyblue
365:skyblue1
366:skyblue2
367:skyblue3
368:skyblue4
369:slateblue
370:slateblue1
371:slateblue2
372:slateblue3
373:slateblue4
374:slategray
375:slategray1
376:slategray2
377:slategray3
378:slategray4
379:snow1
380:snow2
381:snow3
382:snow4
383:springgreen
384:springgreen1
385:springgreen2
386:springgreen3
387:springgreen4
388:steelblue
389:steelblue1
390:steelblue2
391:steelblue3
392:steelblue4
393:tan
394:tan1
395:tan2
396:tan3
397:tan4
398:teal
399:thistle
400:thistle1
401:thistle2
402:thistle3
403:thistle4
404:tomato
405:tomato1
406:tomato2
407:tomato3
408:tomato4
409:turquoise
410:turquoise1
411:turquoise2
412:turquoise3
413:turquoise4
414:violet
415:violetred
416:violetred1
417:violetred2
418:violetred3
419:violetred4
420:wheat
421:wheat1
422:wheat2
423:wheat3
424:wheat4
425:white
426:whitesmoke
427:yellow
428:yellow1
429:yellow2
430:yellow3
431:yellow4
432:yellowgreen

Table 2 . Predefined color names.

RGB(Class Image &$img)

Parameters:
$img Image where the colors should be allocated

Description:
Create a new instance of the color handling class.

Returns:
NA

See also:

Example:

Color(Mix $color)

Parameters:
$color Either a RGB triple or a color name as a string

Description:
Translates a color name to a RGB triple. If an RGB triple is passed through it is returned directly unless it is given in hex, in that case it is first translated to decimal

Returns:
An RGB triple

See also:

Example:
```
$c = RGB::Color("#FFFFFF");
// $c == array(255,255,255)
```

Allocate(Array $color)

Parameters:
$color Color given as either RGB triple, color name or hex-string

Description:
Allocates a new color in the image to which the RGB class belongs. Note that the very first color you allocate (index 0) will become the background color.

Returns:
Color index in image palette.

See also:

Example:

6.3.32 Class FontProp

Defined in file: jpgraph_spider.php
Extends --

FontProp
Public properties
SetFont() SetColor()
Public methods
Private properties & methods

General description

Internal class in spider used to enable the syntax $spider_plot->title->SetFont() by creating a sort of shadow class which is instantiated as poperty “title” in the spider plot.

SetFont($family,$style=FS_NORMAL,$size=12)

Parameters:

$family	Font family
$style	Font style
$size	Font size

Description:
Specify font

Returns:
NA

See also:
NA

Example:
SetFont(FF_ARIAL,FS_NORMAL,12);

SetColor($color)

Parameters:

$color	Named color or RGB array

Description:
Specify color

Returns:
NA

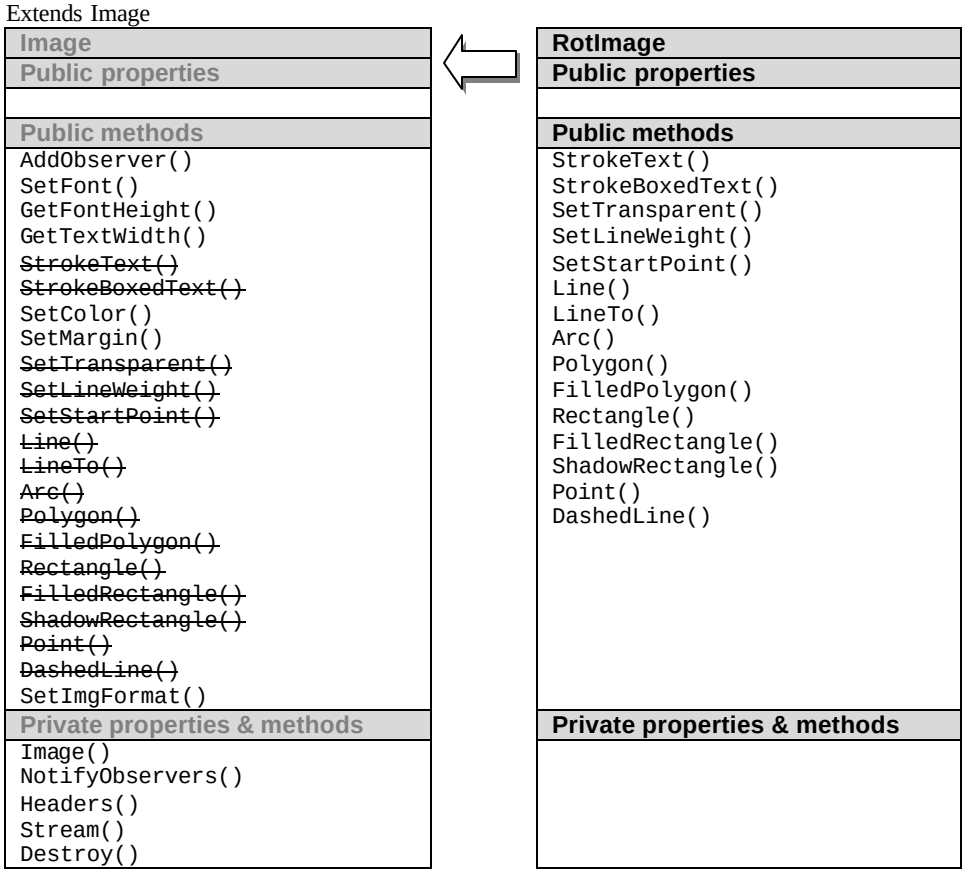
See also:
NA

Example:

```
SetColor(“gray2”);
```

6.3.33 Class RotImage

Defined in file: jpgraph.php
Extends Image



General description

Exactly the same as Image but with the added twist that it rotateds the image► degrees all the methods is exactly as in class Image().

RotImage($aWidth,$aHeight,$a,$aFormat=DEFAULT_GFORMAT)

Parameters:

$aWidth	Image width in pixels
$aHeight	Image height in pixels
$a	Rotation angle
$aFormat	Image format (encoding GIF, PNG, JPG)

Description:

Creates a RotImage class which implements the normal drawing primitives in Images but handles a rotation around (0,0) with a degree.

Returns:

NA

See also:

Image()

Example:

$img = RotImage(300,200,40,“png”);

Manifest constants
In order to control certain behaviours of the library there are a number of DEFINE's at the top of the file 'jpgraph.php'. Their purposes are briefly discussed below. The default values for all these constants should be fine for most users of the library. However, "power-users" might want to tweak these, hence this description.

Constant	Default value	Description
ERR_DEPRECATED	False	Should the use if deprecated functions and values give a fatal runtime error?
BRAND_TIMING	False	Should the time taken to generate an image be "branded" in the lower left corner of the image?
BRAND_TIME_FORMAT	"Generated in: 01.3fs"	The actual format string for the time branding.
READ_CACHE	True	Should JpGraph first look in the cache to see if the image has already been generated?
CACHE_DIR	"./jpgraph_cache"	Location of cache directory. Note this directory must be **writable** for PHP.
USE_BRESENHAM	False	Should a PHP implementation of the Bresenhams's circle algorithm be used instead of the built in GD Arc() drawing routine? (Makes circles look aesthetically better in some few cases – the drawback being that do circles in PHP are slower then native GD)
TTF_DIR	"./ttf"	Location for TTF fonts
DEFAULT_GFORMAT	"auto"	Which graphic format should be used (auto, jpg, gif, png) If this value is set to "auto" then the best available format will automatically be chosen. The preferred order is "png,gif,jpg".

Drawing arbitrary shapes (using dummy graphs)
Disclaimer: This is an unsupported part of JpGraph.

To make it easy to try out arbitrary graphic drawings with all the normal support of JpGraph (like caching, anti-aliasing etc) you can crate a dummy graph. This will in affect give you a canvas where you can use all the drawing primitives in the Image class.

As usual you need to include both jpgraph.php and also the "dummy" extension "jpgraph_dummy.php"

An example to draw a simple line would be

```
#include <jpgraph.php>
#include <jpgraph_dummy.php>

$graph = new DummyGraph(300,200);

$graph->img->SetColor("red");
$graph->img->Line(10,10,100,100);

$graph->Stroke();
```

Utilities
JpGraph 1.2 comes with two completely unsupported utility script to help with color selection and to automatically generate a test page of images. Please note that this is only tools I use myself which I thought might be useful for someone else they are not supported in any shape or form!

Automatic generation of all test images (test-suit)

Running the script "testsuit_jpgraph.php" will generate an index list of all *.php files in the current directory. This is useful if you run this script from the "Examples" directory. It will then generate an index list with a link to all the example images. This is the tool used to manage all regression tests internally in the development of JpGraph.

This script may also be called with a parameter "style" as in "testsuit_jpgraph.php?style=1" or "testsuit_jpgraph.php?style=2". In the latter case (style=2) the links will be replaced by the actual images. You may then visually inspect all the generated images.

Color selection and upcoming support for color themes
Running the script "gencolorchart.php" will generate (by default in the cache directory) a number of images with color samples and also a theme page. Running the script should generate the following output:

JpGraph color chart

Generating color chart images ...

1. ./jpgraph_cache/color_chart01.gif
2. ./jpgraph_cache/color_chart02.gif
3. ./jpgraph_cache/color_chart03.gif
4. ./jpgraph_cache/color_chart04.gif

Generating color chart index page.

Generating themes...

1. ./jpgraph_cache/theme01.gif [24 colors in theme 'earth']
2. ./jpgraph_cache/theme02.gif [19 colors in theme 'pastel']
3. ./jpgraph_cache/theme03.gif [15 colors in theme 'water']
4. ./jpgraph_cache/theme04.gif [11 colors in theme 'sand']

Generating theme index page.

Work done in: 3.64 seconds.

See Colorchart
See Index of themes

Figure 1. Output after running gencolorchart.php

The "Colorchart" is simple a page with all the named colors available in JpGraph. You can see all the colors by following the link "Colorchart" at the bottom of the page. The reason for braking up the colors in separate images is just the fact that the maximum number of colors in one image is limited by the palette size.

Note: This is a good example of the inefficiency of the GIF format as compared to PNG. Each of the above generated GIF images are roughly 100K while the corresponding images generated as PNG is only around 13K in size.

The "themes" index is just a collection of colors that make up a certain theme, i.e. "earth", "pastel" etc. Themes are upcoming feature for 1.3. This utility was just intended to help me to easily view what colors are present in a certain theme. By using a certain theme (in 1.3 and above) your graph will automatically draw colors from that theme, so for example all the default colors for the pie slices in a pie graph will be taken from the theme. Please note that the selection of colors in a specific theme is based on my personal judgement and may not agree with you. If you have additional themes you would like to use please send me a note on jpgraph@aditus.nu

As an example the "earth" (a "professional" looking color theme) have the following tentatively composition:

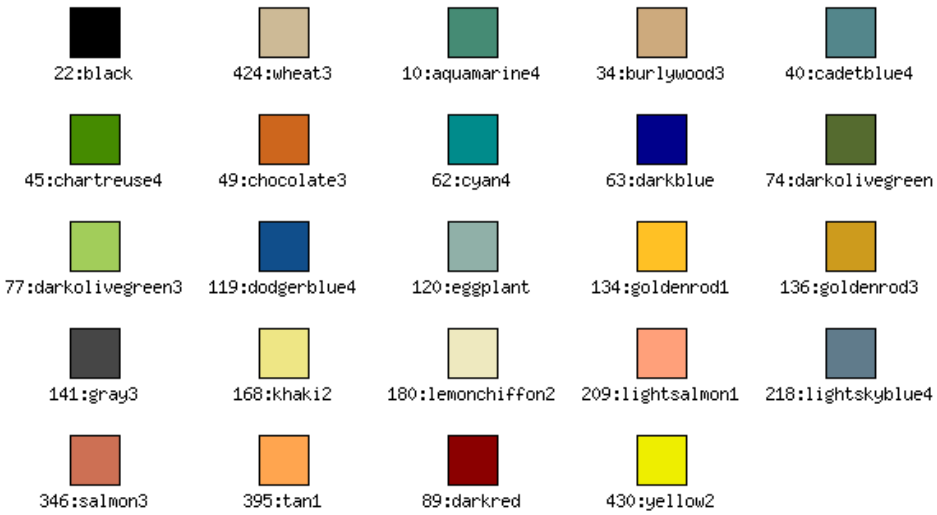
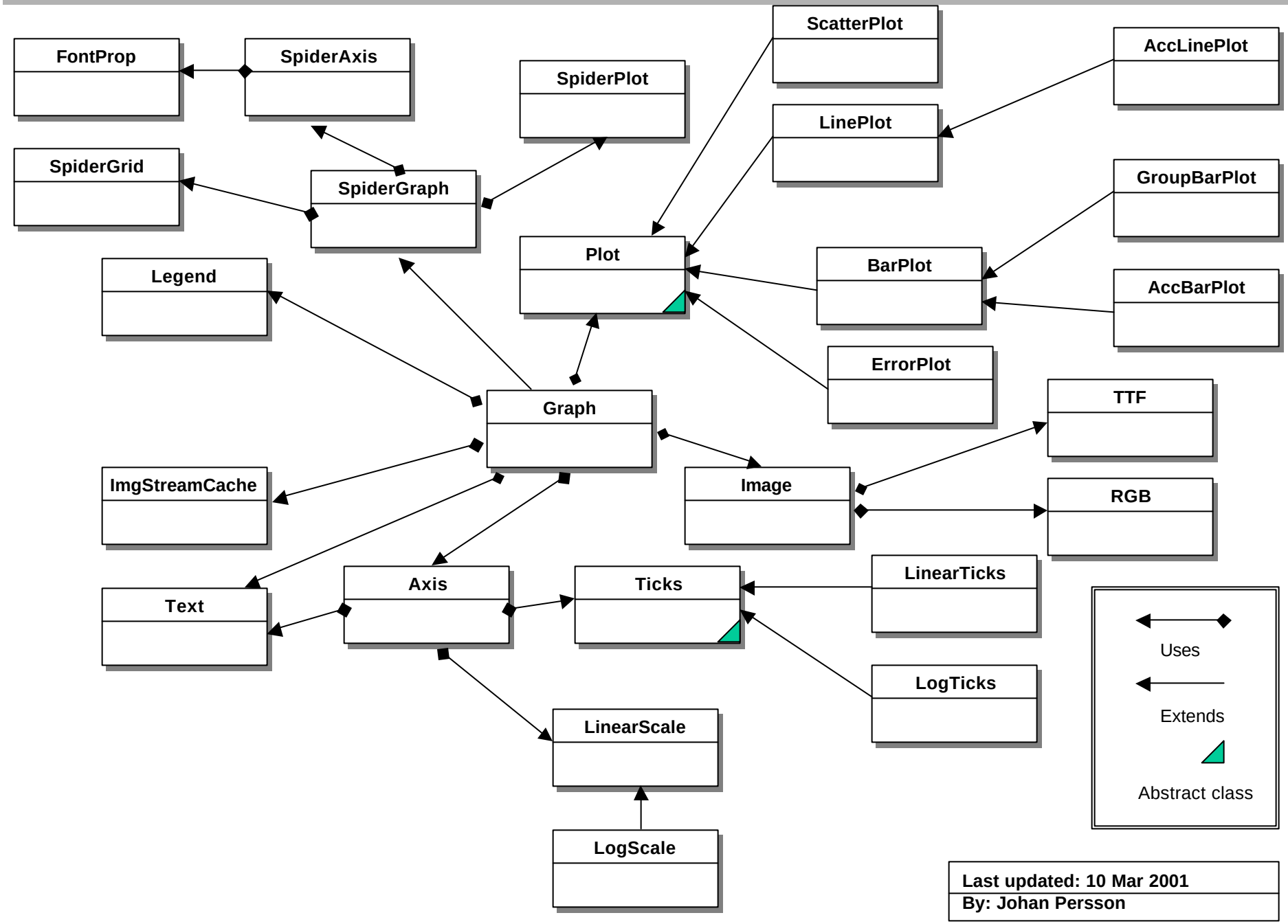


Figure 2. The colors in the "earth" theme (subject to change for 1.3).

JpGraph 1.0 Simplified Class Hierarchy



Specifying fonts

JpGraph supports both a set of built in bit-mapped font as well as True Type Fonts. For scale values on axis it is strongly recommended that you just use the built in bitmap fonts for the simple reason that they are, for most people, easier to read (they are also quicker to render). Try to use TTF only for headlines and perhaps the title for a graph and it's axis. By default the TTF will be drawn with anti-aliasing turned on.

Fonts are generally specified with three parameters

1. Font family
2. Font style
3. Font size

In the call to method SetFont(). If no specified style is dsupplied then the style will default to normal style (FS_STYLE) , size has default value of 12pt.

Built in bitmapped fonts
Built in fonts are chosen by using one of the font families

- ?? FF_FONT0 (small size, does not support bold style)
- ?? FF_FONT1 (normal size)
- ?? FF_FONT2 (large size)

Built in fonts only supports style FS_NORMAL and FS_BOLD (and in the case of FF_FONT0 only FS_NORMAL) trying to specify an unsupported combination for built in fonts will not give an error but will have no effect.

Note: To support backward compatibility with pre-1.2 bitmap fonts might also be specified with FONT0, FONT1, FONT2 (note the missing prefix FF_). However these specifications are deprecated as of 1.2. And usage of these will be a critical error in the next major release. It is strongly suggested that you use the new naming conventions since that is designed to harmonise with the TTF support.

The size parameter has no meaning for built in fonts and will be ignored. The size is implicitly set by choosing the corresponding font family .

Some examples of how to specify the built in fonts

```
SetFont(FF_FONT1,FS_BOLD);
SetFont(FF_FONT1,FS_BOLD,12);     // Size 12 is ignored
SetFont(FONT1);            // Deprecated!
SetFont(FF_FONT2);         // Use built in FONT1 using default style.
SetFont(FF_FONT0,FS_BOLD); // FONT0 does not support bold style, will be ignored
```

True Type Fonts
Before you can start using True Type Fonts you need to make sure that

1. You have downloaded the TTF files. Due to it's size they are in a separate package from the JpGraph script code.
2. The `TTF_DIR` constant in jpgraph.php points to the directory where the font files may be found.
3. You installation of PHP supports TTF (most should do)

By default JpGraph will look for fonts in directory "`./TTF/`"

In JpGraph 1.2 the font families and styles supported are listed in Table 1.

Font family		Font style			
PHP Constant	Real name	FS_NORMAL	FS_BOLD	FS_BOLDITALIC	FS_ITALIC
FF_COURIER	Courier new	✍	✍		✍
FF_VERDANA	Verdana	✍	✍		✍
FF_TIMES	Times New Roman	✍	✍	✍	
FF_HADWRT	Lucida Handwriting	✍			
FF_COMIC	Comic Sans	✍	✍		
FF_ARIAL	Arial	✍	✍		✍
FF_BOOK	Book Antiqua	✍	✍	✍	✍

Table 1 Available combination of TTF font families and styles

The use of a an illegal combination will give a runtime error indicating the type of problem, e.g. "Style not supported for font family". On additional thing to keep in mind when designing graphs is that even though TTF may look more appealing from an aesthetic point of view they are much more time consuming to render and also involves one additional disk access.

Some examples:

```
SetFont(FF_COURIER);  // Courier normal 12 points
SetFont(FF_COURIER,FS_BOLD); // Courier bold 12 points
SetFont(FF_COMIC,FS_BOLD,16); // Comic Sans Serif, bold, 16 points
```

Adding new TTF fonts

If you have a particular favourite font which doesn't come as default it is quite easy to add that font to JpGraph as an extension. There are basically 3 things you need to do:

1. Get the TTF file(s) and add it to your font directory. You need separate files for each of the styles you want to support. These different files uses the following naming conventions:
   Normal font file = <basefilename>
   Bold font file = <basefilename>"bd"
   Bold italic file = <basefilename>"bi"
   Italic file = <basefilename>"i"

2. Define a new constant FF_xxxxx in jpgraph.php which names your font (at the top of the file)

3. Update Class TTF constructor in jpgraph.php with the mapping between your new constant and the <basefilename>

That's it!

Anti-aliased line support

From version 1.2 JpGraph supports drawing of anti-aliased lines. There are a few caveats in order to use this which is discussed in this section.

Note that anti-alising will not be used for either horizontal, vertical or 45 degree lines since they are by their nature are sampled at adequate rate.

Enabling anti-aliased lines

Anti-aliased lines are enabled by calling the method SetAntiAliasing() in the Image class, so for example you would normally make the call

```
$graph->img->SetAntiAliasing()
```

to enable this feature. The anti-aliasing for lines works by “smoothing” out the edges on the line by using a progressive scale of colors interpolated between the background color and the line color. Hence the line drawing algorithm needs to know the background color. By default the line drawing algorithm looks at the first point of the line to see what the underlying color is and then uses this as the background color. This might not always give the best result since you might have several lines starting from the same point. Then the first line will correctly read the background color but the second line (which starts from the same point) will only see the previous lines color and not the real background color.

To solve this problem you can specify the background color as a parameter in the call to SetAntiAliasing() method. This will then be used for all subsequent lines. For example a call would say

```
$graph->img->SetAntiAliasing(“white”);
```

to use “white” as the background color regardless what the color at start position of the line is. An example of where you must use this is for “spider-plots” since the axis for the spider plot all overlap in the center.

Anti-aliased gotchas
There are also a couple of potential limitations (or gotchas) you probably would like to keep in mind when using anti-aliased lines

1. Anti-aliases lines are much slower then the normal lines, roughly 5 times slower. Remember that the whole line-drawing algorithm is implemented in PHP since the underlying graph library (GD) doesn’t support anti-aliased lines.

2. Anti-aliased lines uses up more of the available color-palette. The exact number of colors used is dependent on the line-angle (number of lines with different angles uses more colors). Hence it might not be possible to use anti-aliasing with color-gradient fill since the number of available colors in the palette might not be enough. The color gradient is limited to use 100 color bands between the two colors. A normal palette can keep around 256 colors (I’m not 100% sure of the exact format used in the JPG, PNG, or GIF standards)

3. All anti-aliased line should have the same background color if the color is specified in the call to SetAntiAliasing(). Otherwise only the part of the line that covers the specified background color will be anti-aliased. The same goes for lines where the color is automatically determined but here each line may have its own background.

4. Anti-aliased lines will ignore the line width specified. They will always have a width of roughly 1.

1 Using Spider Plots

1.1 Introduction

Spider plots are most often used to display how a number of results compare to some set targets. They make good use of the human ability to spot symmetry (or rather un-symmetry) . the figure below show an example of a spider (sometimes called a web-plot). Spiderplots are not suitable if you want very accurate readings from the graph since, by it's nature, it can be difficult to read out very detailed values.

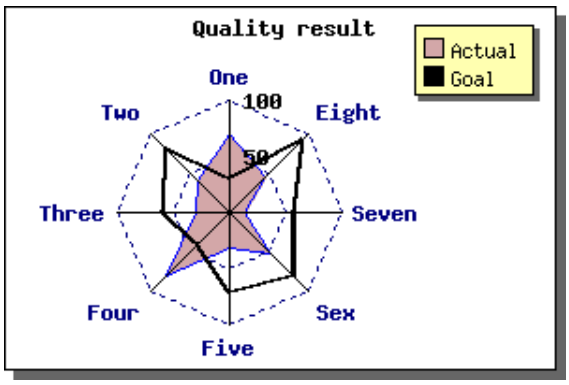


Figure 1. Example of a spider graph with two plots.

The following points are worth noting:

- There is one axis for each data point
- Each axis may have an arbitrary title which is automatically positioned
- A spider plot may be filled or open
- You can control color, weight of lines as you are already used to
- A spider plot can, as usual, have a title and a legend
- The first axis is always oriented vertical and is the only axis with labels
- Grids may be used (dashed in the figure above)
- You may have ticks (although suppressed in the figure above
- You can control the size and position within the frame of the graph
- You may have several plots within the same graph

In the following section we show how to draw both simple and complex spider graph. As we will show all the settings will follow the same pattern as for the more standard linear graphs.

1.2 Creating a simple spider graph.

Let's start by creating a very simple spider plot based on 5 data points using mostly default values.

As the first thing you must remember to include the extension module that contains spider plot. "jpgraph_spider.php".

```
<?php
	include ("jpgraph.php");
	include ("jpgraph_spider.php");

	// Some data to plot
	$data = array(55,80,46,71,95);

	// Create the graph and the plot
	$graph = new SpiderGraph(250,200);
	$plot = new SpiderPlot($data);

	// Add the plot and display the graph
	$graph->Add($plot);
	$graph->Stroke();
?>
```

If you run the above script it will generate the following image

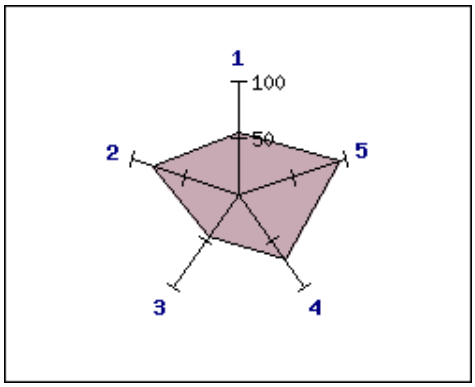


Figure 2. A very simple spider plot

From the above image you may note the following have been set automatically

- ?? Each axis have been given a default title (it's number)
- ?? Major tick marks are displayed on each axis
- ?? The scale has been determined by autoscaling
- ?? The plot is filled by default
- ?? The size of the graph has been determined to fit within the given image size

1.3 Controlling size and position of plot

One of the simplest changes we can do is change the size and the position of the graph. These two parameters are controlled by the two methods SetCenter() and SetPlotSize(). The parameters of both these methods are in percentage.

To demonstrate lets make the plot smaller and move it a little bit to the left in the image. This is accomplished by the lines

```
. . .
// Set diameter of spider graph to 40% of min(height,weight)
$graph->SetPlotSize(0.4);

// Position the centre of the graph at x=30% of the width and y=50% of
height
$graph->SetCenter(0.3,0.5);
. . .
```

The resulting image will then be as displayed below

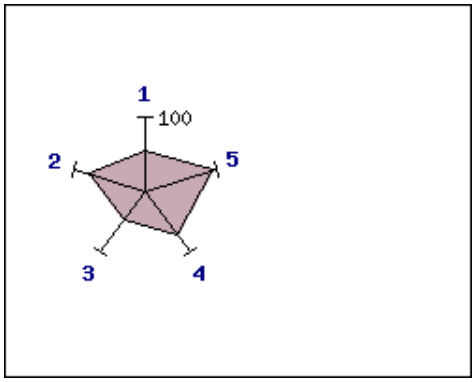


Figure 3. Resized and repositioned spider graph.

It is worth clarifying how the sizing works. Since the size is given in percentage you might, rightfully so, ask percentage of what? Image height?, image width? Since the axis can be in all directions we take the percentage of min(height,width).

1.4 Specifying titles for the axis and legends for plots

We normally would like something more meaningful as description of each axis then it's number. Specifying the titles are accomplished through the use of the method SetTitles() of the graph. Let's say that each axis corresponds to a month.

```
. . .
$axtitles=array("Jan","Feb","Mar","Apr","May");
$graph->SetTitles($axtitles);
```

Let's also specify a legend for the plot

```
. . .
$plot->SetLegend("Defects");
```

Let's also take the opportunity to set a title of the graph

```
. . .
$graph->title->Set("Result 2001");
$graph->title->SetFont(FONT1_BOLD);
```

The resulting graph are now starting to look a little bit more pleasing as the following figure illustrates

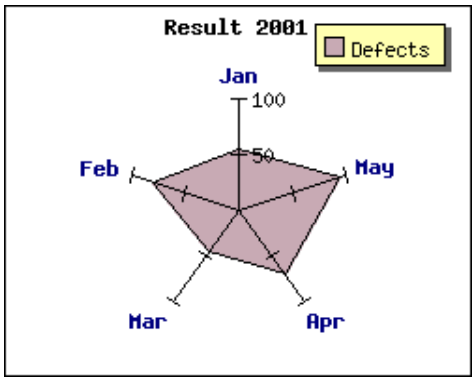


Figure 4. Spider graph with legends, and titles.

1.5 Specifying gridlines

By default the graph has tick lines but no grid lines. Let's change this so that we don't have any ticks but use "dashed" gridlines instead.

To suppress ticks we could do it the same way as for linear graphs by calling the SupressTickMarks() method of Ticks. This would be accomplished by

```
. . .
$graph->axis->scale->ticks->SupressTickMarks();
```

Since this is, in OO terms, a clean design is it still a little bit unwieldy . There is therefore a shortcut which lets you just say

```
. . .
$graph->SupressTickMarks()
```

To set a "dashed" apperance of the grid you have to invoke the SetLineStyle() method of the grid and to show the grid lines you just call, as before, the method Show() of the grid as

```
. . .
$graph->grid->SetLineStyle("dashed");
$graph->grid->Show();
```

The default color for grid is "silver" but you may of course easily change that by invoking the SetColor() method on the grid.

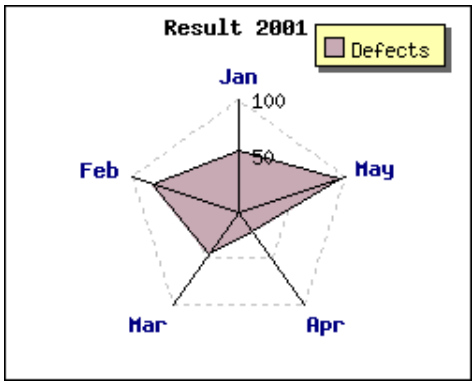


Figure 5. A Spiderplot with gridlines and no ticks.

By design the plot is above the gridline but beneath the axis in image depth, hence some part of the gridlines are hidden.

To have the gridlines more “visible” just change their color, say to, dark red by invoking the SetColor() method as

```
. . .
$graph->grid->SetColor(“darkred”);
```

The resulting image will be

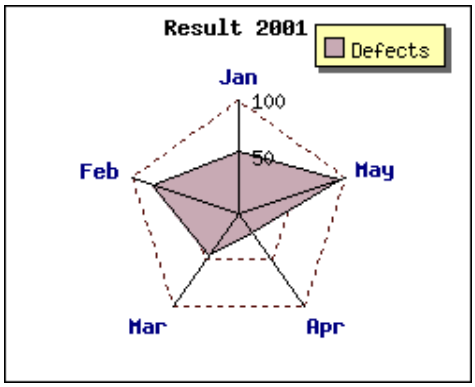


Figure 6. Spider graph with dark red grid lines

1.6 Setting background color and frame

By default the image has a frame with “white” as the background color. As you saw for normal Linear plot we can have a background and a shadow of the frame. The one difference between spider plots and linear plots is that there is no concept of margin and plot area color, there is only one background color. This is set through the Color() method of graphs.

To set a shadowed frame you just evoke the SetShadow() method of the graph. It has the same parameter as the previous introduced Linear graphs.

Lets set thye background to a very light blue-ish color and add a shadow to the frame. This is done by the two lines

```
$graph->SetShadow();
$graph->SetColor(array(200,230,230));
```

and the resulting graph

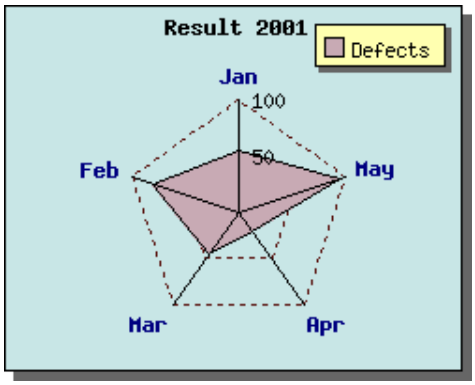


Figure 7. Spider graph with background and shadow

1.7 Adding several plots to a spider graph

This is done exactly the same way as for the other graph types, just call the method Add() of class SpiderGraph for each plot you want to add. For spider plots it is important that all the plots have the same number of data points. The library will check this and treat this as an error and abort the program.

Lets add a second plot to our previous graph and let's make that an open plot, i.e. it is not filled, and make the weight of the line 2.

```
$data2 = array(65,95,50,75,60);
$plot2 = new SpiderPlot($data2);
$plot2->SetFill(false);
$plot2->SetLineWeight(2);
$plot2->SetLegend("Target");
. . .
$graph->Add($plot2);
```

The resulting graph will now look like

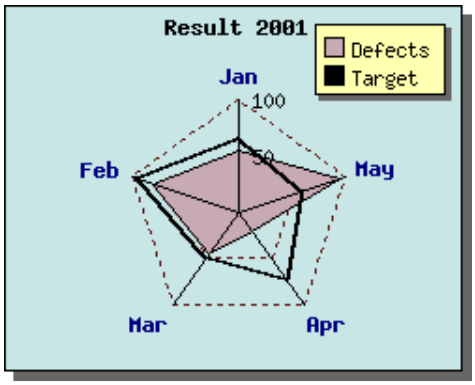


Figure 8. Spider graph with two plots.

1 Using the cache mechanism and other performance related questions

1.1 Performance considerations

Since PHP is not a compiled language and the plot generated by this library require non-trivial work by PHP this must be seriously considered in the overall design for a web site. Generating complex graphs with many data-points is bound to take time. As a rough guideline most of the graphs demonstrated above take in the order of 1-2s to be generated and send back to a browser on a local network using a rather old PC with a slow disk as my local server (PII 166MHz). Experience shows that time spend is roughly 30% parsing the actual PHP code and 70% of the overall time depends on the complexity of the graphs. Hence no matter how simple graphs are you will have to face at least a standard hit to generate each graph.

This might be unacceptable in a high volume site. There is not much we can do about the complexity of the library if we want all this functionality and there is little we can do about the speed by which PHP parses the library.

However, we can do something. If you have non-real-time graphs that might only get new data, say every, 24h. Then it would be possible to generate the image and then save it in a file cache so that the next time a user requests this graph it is read from disk instead of generated by PHP. Every night we might then clear the cache and the first user whom requests the graph the next day will take the hit of actually generating it but the rest of the user will just be fetching the generated cached version.

1.2 Using the cache mechanism

The cache mechanism kicks in if you call the graph constructor with an additional file name as an additional parameter. One of two things will now happen

1. The file cache is searched for a file with this name. If the file exists it is read and passed through to the browser with very little overhead.
2. The file does not exist in the cache. In that case the graph is generated in the normal way but before it is passed back to the browser it is saved as a file in the cache

To use the cache your call could for example be

```
$graph = new Graph(300,200,"myfilename.png")
```

Note that In the above example I used the extension "*.PNG" for the file. To use any specific extension is not necessary or any extension at all in fact.

For the cache to work you must have a directory called "jpgraph_cache" in the same place as you run your script from since the library will search for a directory called "./jpgraph_cache". This directory must be readable and writeable for PHP. This scheme is not completely free from hassle from a security point of view. The other way would be to have a "global" cache directory but this increases the risk for a name

clash and it might also not be possible if you are using some ISP that only allows you to create files within your own area.

The name of the directory used for caching might also be easily changed since it is defined in jpgraph.php as a

```
DEFINE(“CACHE_DIR”, “./jpgraph_cache”)
```

near the top of the file. You could for example change it to some ge neral temp area (perhaps /var/tmp or similar) .

I’m not completely satisfied with the way this currently works since the cache directory in practice must be writeable for everyone and his uncle unless you can persuade the administrator to add you and PHP to the same group and then just make the directory writeable for member so that group. Any suggestions on how to better cope with this potential security whole are welcome!

1 Advanced features of JpGraph

1.1 Using grace percentage on scales

By default the autoscaling algorithm tries to make best possible use of screen estate by making the scale as large as possible, i.e. the extreme values (min/max) will be on the top and bottom of the scale if they happen to fall on a scale-tick. So for example doing a simple line plot could look like the plot shown in Figure 1 below.

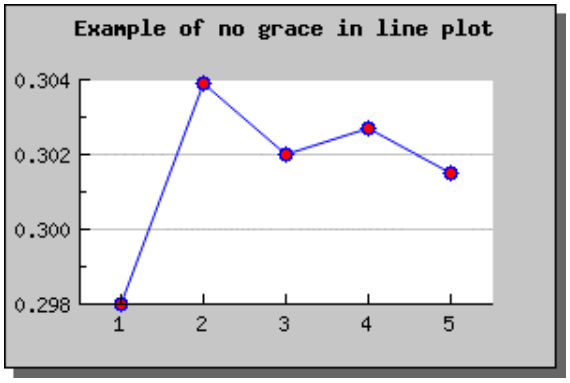


Figure 1. Example of graph with grace=0 (default values).

However you might sometime want to add some extra to the minimum and maximum values so that there is some "air" in the graph between the end of the scale values and the extreme points in the graphs. This can be done by adding a "grace" percentage to the scale. So for example adding 10% to the y-scale in the image above is done by calling the SetGrace() method on the yscale as

```
$graph->yscale->SetGrace(10);     // Set 10% grace.
```

After this the graph will look like shown in figure below

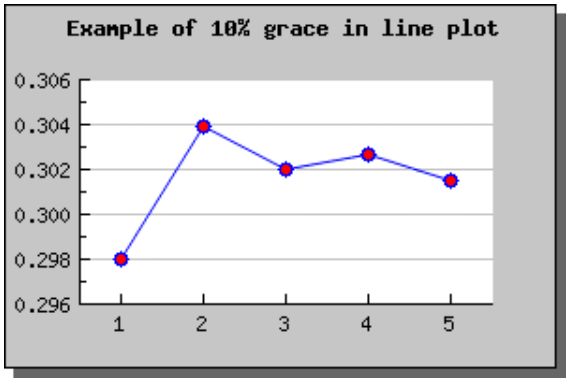


Figure 2. Example of using grace for the Y-scale.

As you can see the dynamic range has been reduced by roughly 10% . The exact value will depend on the endpoints chosen by the autoscaling algorithm. The grace simply works by adding the percentage grace value of the dynamic range (maximum-minimum) and using those values as the min and max values sent into the autoscaling algorithm.

Note: *As you can see the above graph also makes use of SetCenter() for the lineplot so thet the numbering on the x-axis is placed in the center of each tick-"slot".*

As an example the complete code for the last graph in Figure 2 above is:

```
<?php
include ("jpgraph.php");
include ("jpgraph_line.php");

$datay = array(0.2980,0.3039,0.3020,0.3027,0.3015);
$graph = new Graph(300,200);
$graph->img->SetMargin(40,40,40,40);
$graph->img->SetAntiAliasing();
$graph->SetScale("textlin");
$graph->SetShadow();
$graph->title->Set("Example of 10% grace in line plot");
$graph->title->SetFont(FF_FONT1,FS_BOLD);
$graph->yscale->SetGrace(10);

$p1 = new LinePlot($datay);
$p1->mark->SetType(MARK_FILLEDCIRCLE);
$p1->mark->SetFillColor("red");
$p1->mark->SetWidth(4);
$p1->SetColor("blue");
$p1->SetCenter();
$graph->Add($p1);

$graph->Stroke();

?>
```

Figure 3. The code that generated Figure 2 above.

1.2 Timing the generation of graphs

When evaluating the performance (or suitability for on-line graph genertation) for graphs there must be a simple way to get a knowledge on the time it takes to generate a specific image. In JpGraph this works by the possibility to brand each generated picture by the time in s (and ms) it took PHP to generate that image.

This is controlled by the definition (in jpgraph.php)

```
. . .
DEFINE("BRAND_TIMING",TRUE);
. . .
```

By specifying this constant true or false you can determine wheter or not you would like to have the image branded by the time. The actual string that gets formatted is specified by the definition

```
. . .
DEFINE("BRAND_TIME_FORMAT","Generated in: %01.3fs");
. . .
```

This let's you easy customize the actual string that gets printed on the image. This string will always be printed in the lower left corner of the graph. The image below show an example of a graph with the time branded into it.

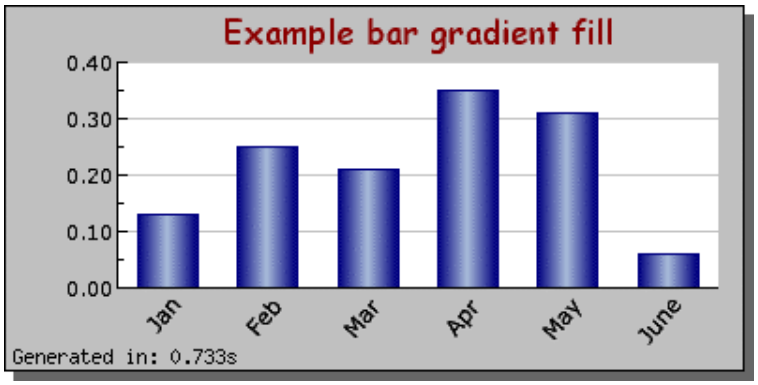


Figure 4Graph with timing information. (on my very slow old server…)

1.3 Using color gradient fill

From version 1.2 it is possible to use color gradient fill for certain graphs. As of this writing only bar graphs supports color gradient fill at the moment. In future releases of JpGraph there will be added functionality to use gradient fill for backgrounds and possible area-graphs (filled line-plots).
Color gradient fill fills the image with a smooth transition between to colors. In what direction the transition goes (from left to right, down and up, fomr the middle and out etc) is determined by the style of the gradient fill. JpGraph currently supports 7 different styles.

Before explaining how this feature is used you must be aware of two caveats with gradient filling:

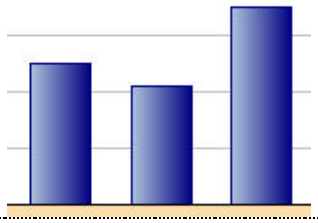
1. gradient filling is computational expensive. Large plots with gradient fill will take in the order of 6 times longer to fill then for a normal one-color fill. This might to some extent be helped by making use of the cache feature of JpGraph so that the graph is only generated once or a few times.

2. gradient filling will make use of much more colors (by definition) this will make the color palette for the image bigger and hence make the overall image larger. It might also have some severe effect on using anti-aliased line in the same image as color gradient filling since anti-aliased lines also have the possibility to make use of many colors. Hence the color palette might not be big enough for all the colors you need.

   This problem is often seen as that for no apperant reason some color you have specified in the image does appear as another color. (This is not a bug in JpGraph!) This is something to especially watch out for when enabling anti-alising since that also uses a lot of colors. Since the numbers of colors used with anti-alising depends on the angle on the lines it is impossible to foresee the number of colors used for this.

Different styles of gradient filling

The seven different styles are specified by using the specific PHP constants defined in jpgraph.php. Currently the following styles are available:

Style	Description	Example
GRAD_VER	The gradient moves from left to right	

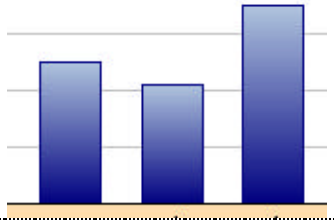
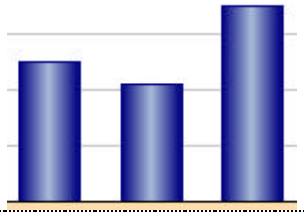
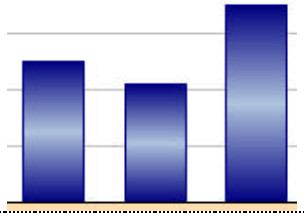
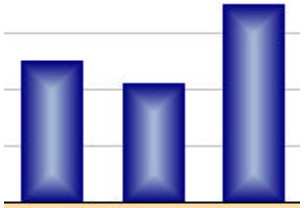
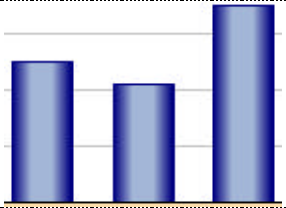
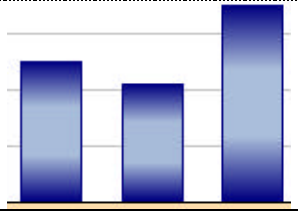
Style	Description	Example
GRAD_HOR	The gradient moves from top to bottom	
GRAD_MIDVER	The gradient goes from the middle and vertically up/down	
GRAD_MIDHOR	The gradient goes from the middle and horizontally left/right	
GRAD_CENTER	The gradient radiates from the center and outwards.	
GRAD_WIDE_MIDVER	Similar to GRAD_MIDVER but with the middle color taking up a wider area	
GRAD_WIDE_MIDHOR	Similar to GRAD_MIDHOR but with the middle color taking up a wider area	

Table 1. Different styles of color gradient filling

Using color gradient bar graphs

You only need to create the barplot as usual and then call method `SetFillGradient()` where you need to specify two colors and a gradient style. So for example to specify the GRAD_WIDE_MIDHOR style as in the last example in Table 1

```
$barplot->SetFillGradient("navy","lightsteelblue",GRAD_WIDE_MIDHOR);
```

1.4 Specifying fonts

JpGraph supports both a set of built in bit-mapped font as well as True Type Fonts. For scale values on axis it is strongly recommended that you just use the built in bitmap fonts for the simple reason that they are, for most people, easier to read (they are also quicker to render). Try to use TTF only for headlines and perhaps the title for a graph and it's axis. By default the TTF will be drawn with anti-aliasing turned on.

Fonts are generally specified with three parameters

1. Font family
2. Font style
3. Font size

In the call to method SetFont(). If no specified style is dsupplied then the style will default to normal style (FS_STYLE) , size has default value of 12pt.

Built in bitmapped fonts

Built in fonts are chosen by using one of the font families

- FF_FONT0 (small size, does not support bold style)
- FF_FONT1 (normal size)
- FF_FONT2 (large size)

Built in fonts only supports style FS_NORMAL and FS_BOLD (and in the case of FF_FONT0 only FS_NORMAL) trying to specify an unsupported combination for built in fonts will not give an error but will have no effect.

Note: To support backward compatibility with pre-1.2 bitmap fonts might also be specified with FONT0, FONT1, FONT2 (note the missing prefix FF_). However these specifications are deprecated as of 1.2. And usage of these will be a critical error in the next major release. It is strongly suggested that you use the new naming conventions since that is designed to harmonise with the TTF support.

The size parameter has no meaning for built in fonts and will be ignored. The size is implicitly set by choosing the corresponding font family .

Some examples of how to specify the built in fonts

```
SetFont(FF_FONT1,FS_BOLD);
SetFont(FF_FONT1,FS_BOLD,12);     // Size 12 is ignored
SetFont(FONT1);            // Deprecated!
SetFont(FF_FONT2);         // Use built in FONT1 using default style.
SetFont(FF_FONT0,FS_BOLD); // FONT0 does not support bold style, will be
ignored
```

True Type Fonts

Before you can start using True Type Fonts you need to make sure that

1. You have downloaded the TTF files. Due to it's size they are in a separate package from the JpGraph script code.
2. The `TTF_DIR` constant in jpgraph.php points to the directory where the font files may be found.
3. You installation of PHP supports TTF (most should do)

By default JpGraph will look for fonts in directory "`./TTF/`"

In JpGraph 1.2 the font families and styles supported are listed in Table 2.

Font family		Font style			
PHP Constant	Real name	FS_NORMAL	FS_BOLD	FS_BOLDITALIC	FS_ITALIC
FF_COURIER	Courier new	✍	✍		✍

FF_VERDANA	Verdana	✍	✍		✍
FF_TIMES	Times New Roman	✍	✍	✍	
FF_HADWRT	Lucida Handwriting	✍			
FF_COMIC	Comic Sans	✍	✍		
FF_ARIAL	Arial	✍	✍		✍
FF_BOOK	Book Antiqua	✍	✍	✍	✍

Table 2 Available combination of TTF font families and styles

The use of a an illegal combination will give a runtime error indicating the type of problem, e.g. "Style not supported for font family". On additional thing to keep in mind when designing graphs is that even though TTF may look more appealing from an aesthetic point of view they are much more time consuming to render and also involves one additional disk access.

Some examples:

```
SetFont(FF_COURIER);  // Courier normal 12 points
SetFont(FF_COURIER,FS_BOLD); // Courier bold 12 points
SetFont(FF_COMIC,FS_BOLD,16); // Comic Sans Serif, bold, 16 points
```

Adding new TTF fonts

If you have a particular favourite font which doesn't come as default it is quite easy to add that font to JpGraph as an extension. There are basically 3 things you need to do:

1. Get the TTF file(s) and add it to your font directory. You need separate files for each of the styles you want to support. These different files uses the following naming conventions:
   Normal font file = <basefilename>
   Bold font file = <basefilename>"bd"
   Bold italic file = <basefilename>"bi"
   Italic file = <basefilename>"i"

2. Define a new constant FF_xxxxx in jpgraph.php which names your font (at the top of the file)

3. Update Class TTF constructor in jpgraph.php with the mapping between your new constant and the <basefilename>

That's it!

1.5 Using Anti-Aliasing

From version 1.2 JpGraph supports drawing of anti-aliased lines. There are a few caveats in order to use this which is discussed in this section.

Note that anti-alising will not be used for either horizontal, vertical or 45 degree lines since they are by their nature are sampled at adequate rate.

Enabling anti-aliased lines

Anti-aliased lines are enabled by calling the method SetAntiAliasing() in the Image class, so for example you would normally make the call

```
$graph->img->SetAntiAliasing()
```

to enable this feature. The anti-aliasing for lines works by "smoothing" out the edges on the line by using a progressive scale of colors interpolated between the background color and the line color.

Note: The algorithm used is quite simple. It would be possible to achieve even better result by doing some real 2D signal processing. However it is my view that doing real time 2D signal processing on a WEB server would be madness so I deliberately kept it simple. To achieve best visual result always use a dark line color on a light background.

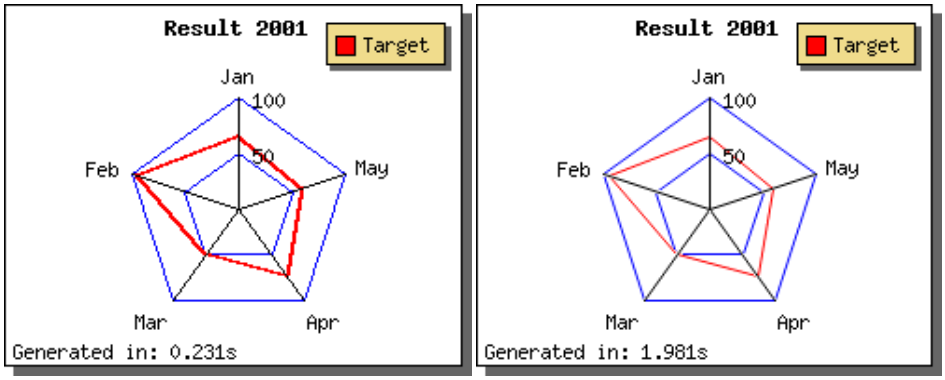


Figure 4. Spider graph with and without anti-aliasing enabled.

An example will show that this, quite simple algorithm, gives a reasonable good result. Figure 3 shows a spider graph with ant without anti-aliasing. One thing to keep in mind when deciding to use anti-aliasing is that it could have a potentially dramatic effect on the time it takes to generate the image (compare 0.2s with 2.0, a factor of ten!) (the code for this particular spider graph might be found as spiderex6.php in the Examples directory, (you might want to see how much faster your machine is to my old server, but hey it's a seven year old machine sitting in my basement and doubling as a firewall as well)

Anti-aliased "gotchas"

There are also a couple of potential limitations (or gotchas) you probably would like to keep in mind when using anti-aliased lines

1. Anti-aliases lines are much slower then the normal lines, roughly 5 times slower per line. Remember that the whole line-drawing algorithm is implemented in PHP since the underlying graph library (GD) doesn't support anti-aliased lines.

2. Anti-aliased lines uses up more of the available color-palette. The exact number of colors used is dependent on the line-angle, a near horizontal or near vertical line uses more colors (number of lines with different angles uses more colors). Hence it might not be possible to use anti-aliasing with color-gradient fill since the number of available colors in the palette might not be enough. A normal palette can keep around 256 colors (I'm not 100% sure of the exact format used in the JPG, PNG, or GIF standards)

3. Anti-aliased lines will ignore the line width specified. They will always have a width of roughly 1.

1.6 JpGraph global defines

In order to control certain behaviours of the library there are a number of DEFINE's at the top of the file 'jpgraph.php'. Their purposes are briefly discussed below. The default values for all these constants should be fine for most users of the library. However, "power-users" might want to tweak these, hence this description.

Constant	Default value	Description
ERR_DEPRECATED	False	Should the use if deprecated functions and values give a fatal runtime error?
BRAND_TIMING	False	Should the time taken to generate an image be "branded" in the lower left corner of the image?
BRAND_TIME_FORMAT	"Generated in: 01.3fs"	The actual format string for the time branding.
READ_CACHE	True	Should JpGraph first look in the cache to see if the image has already been generated?
CACHE_DIR	"./jpgraph_cache"	Location of cache directory. Note this directory must be **writable** for PHP.
USE_BRESENHAM	False	Should a PHP implementation of the Bresenhams's circle algorithm be used instead of the built in GD Arc() drawing routine? (Makes circles look aesthetically better in some few cases – the drawback being that do circles in PHP are slower then native GD)
TTF_DIR	"./ttf"	Location for TTF fonts
DEFAULT_GFORMAT	"auto"	Which graphic format should be used (auto, jpg, gif, png) If this value is set to "auto" then the best available format will automatically be chosen. The preferred order is "png,gif,jpg".

1.7 Drawing arbitrary graphic shapes using dummy graphs

Disclaimer: This is an unsupported part of JpGraph.

To make it easy to try out arbitrary graphic drawings with all the normal support of JpGraph (like caching, anti-aliasing etc) you can crate a dummy graph. This will in affect give you a canvas where you can use all the drawing primitives in the Image class.

As usual you need to include both jpgraph.php and also the "dummy" extension "jpgraph_dummy.php"

An example to draw a simple line would be

```
#include <jpgraph.php>
#include <jpgraph_dummy.php>

$graph = new DummyGraph(300,200);

$graph->img->SetColor("red");
$graph->img->Line(10,10,100,100);

$graph->Stroke();
```

1.8 Utility scripts

Disclaimer: This is an unsupported part of JpGraph.

JpGraph 1.2 comes with two utility script to help with color selection and to automatically generate a test page of images. Please note that this is only tools I use myself which I thought might be useful for someone else. They are not supported in any shape or form!

Automatic generation of all test images (test-suit)

Running the script "testsuit_jpgraph.php" will generate an index list of all *.php files in the current directory. This is useful if you run this script from the "Examples" directory. It will then generate an index list with a link to all the example images. This is the tool used to manage all regression tests internally in the development of JpGraph.

This script may also be called with a parameter "style" as in

"`testsuit_jpgraph.php?style=1`"

or

"`testsuit_jpgraph.php?style=2`"

In the latter case (`style=2`) the links will be replaced by the actual images. You may then visually inspect all the generated images.

Color selection and upcoming support for color themes

Running the script "gencolorchart.php" will generate (by default in the cache directory) a number of images with color samples and also a theme page. Running the script should generate the following output:

JpGraph color chart

Generating color chart images ...

1. ./jpgraph_cache/color_chart01.gif
2. ./jpgraph_cache/color_chart02.gif
3. ./jpgraph_cache/color_chart03.gif
4. ./jpgraph_cache/color_chart04.gif

Generating color chart index page.

Generating themes...

1. ./jpgraph_cache/theme01.gif [24 colors in theme 'earth']
2. ./jpgraph_cache/theme02.gif [19 colors in theme 'pastel']
3. ./jpgraph_cache/theme03.gif [15 colors in theme 'water']
4. ./jpgraph_cache/theme04.gif [11 colors in theme 'sand']

Generating theme index page.

Work done in: 3.64 seconds.

See Colorchart
See Index of themes

Figure 5. Output after running gencolorchart.php

The "Colorchart" is simple a page with all the named colors available in JpGraph. You can see all the colors by following the link "Colorchart" at the bottom of the page. The reason for braking up the colors in separate images is just the fact that the maximum number of colors in one image is limited by the palette size.

Note: This is a good example of the inefficiency of the GIF format as compared to PNG. Each of the above generated GIF images for the color charts are roughly 100K while the corresponding images generated as PNG are only around 13K in size.

The “themes” index is just a collection of colors that make up a certain theme, i.e. “earth”, “pastel” etc. Themes are an upcoming feature in JpGraph 1.3. This utility was just intended to help me to easily view and pick what colors are/should be present in a certain theme. By using a certain theme your graph will automatically draw colors from that theme, so for example all the default colors for the pie slices in a pie graph will be taken from the theme. Please note that the selection of colors in a specific theme is based on my personal judgement and may not agree with you. If you have additional themes you would like to use please send me a note on jpgraph@aditus.nu

As an example the “earth” (a “professional” looking color theme) have the following tentatively composition:

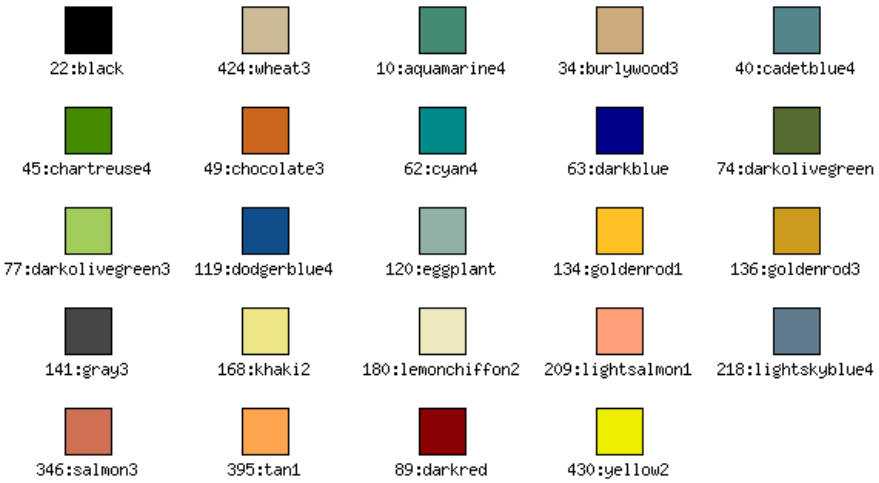


Figure 6. The colors in the "earth" theme (subject to change for 1.3!).

As opposed to the more colorful “pastel”-theme shown below

Figure 7. The colors in the "pastel" theme (subject to change fror 1.3!)