

JpGraph Manual

Version : 3.1.3

Aditus Consulting

JpGraph Manual : Version : 3.1.3

Aditus Consulting

Published 2009-10-06

Copyright © 2000-2009 Aditus Consulting

Table of Contents

Preface and history of the JpGraph library	xix
I. Installing and verifying the configuring	1
1. About the library	2
2. The Short Version: Installing the library (for PHP/Apache experts)	14
3. The Long Version: Installing the Library	17
II. Basic graph creation	38
4. Your first graph script	39
5. Fundamentals of dynamic graph generation	49
6. Error handling	59
7. Color handling	65
8. Text and font handling	69
9. Using the JpGraph cache system	85
10. Using CSIM (Client side image maps)	91
11. NuSphere PHP accelerator	103
III. Common features	104
12. Commonalities for all graphs	105
13. Getting hold of the data to be displayed	109
14. Common features for all Cartesian (x,y) graph types	120
IV. Creating linear and non-linear graphs	194
15. Different types of linear (cartesian) graph types	196
16. Non-Linear graph types	271
17. Additional graph types	348
18. Miscellaneous formatting and tools	362
V. Additional graph types available in the professional version	366
19. Graphical tables	368
20. Odometer	387
21. Windrose	411
22. Matrix graphs	463
23. Filled contour graphs	491
VI. Barcodes	498
24. Linear Barcodes (One Dimensional Barcodes)	499
25. PDF417 (2D-Barcode)	521
26. Datamatrix (2D-Barcode)	546
27. QR (2D-Barcode)	566
VII. Case studies	597
28. Synchronized Y-axis	599
29. USPS Confirmation Barcodes	602
30. Showing SPAM statistics	606
31. Creating Critical chain buffer penetration charts	621
VIII. Appendices	633
A. How this manual was produced	635
B. JpGraph Professional License	638
C. FAQ	641
D. Named color list	661
E. Available plot marks	673
F. List of all country flags	677
G. List of files included in the library	696
H. Error messages	699
I. Compiling PHP	716
J. Setting up PHP5 in parallel with PHP4 in SuSE 10.1	721
K. Why it is not possible to add a SVG backend to JpGraph	730

L. The JpGraph configuration file	733
---	-----

List of Figures

1. Link to highlighted graph source	xx
1.1. JpGraph and PHP	4
1.2. This is the very first example (example0.php)	5
2.1. phpinfo() GD sections	15
3.1. Pro-login dialogue on JpGraph Website	17
3.2. phpinfo() GD-Information	19
3.3. Verifying the GD installations (checkgd.php)	20
3.4. Verifying GD2 (checkgd2.php)	21
3.5. Verifying TTF with a known font (checkttf.php)	22
4.1. The first ten rows of data of sunspot activities from year 1700	40
4.2. Reading numeric tabulated sunspot data from a file	40
4.3. Line plot showing the number of sun spots since 1700 (sunspotsex1.php)	42
4.4. Displaying sun spots with a semi filled line graph (sunspotsex2.php)	43
4.5. Adding tick labels to the graph (sunspotsex3.php)	44
4.6. Manually specifying the X scale to use just the supplied X values (sunspotsex4.php)	45
4.7. Using a callback to get correct values on the x axis (sunspotsex5.php)	46
4.8. Changing the plot type to a bar plot instead (sunspotsex6.php)	47
4.9. Sunspots zoomed to only show the last 20 years (sunspotsex7.php)	48
6.1. Typical image error message	59
6.2. The "Header already sent error message"	59
6.3. The "Header already sent" error message using German locale	61
6.4. Using the production "pseudo" locale	61
7.1. Using alpha channel modifiers	66
7.2. Making use of transparency to combine two plots (barlinealphaex1.php)	66
7.3. Adjusting brightness of named color specifier	67
7.4. Adjusting brightness of a HTML color specifier	67
8.1. List of all latin TTF fonts. (listfontsex1.php)	74
8.2. Illustration of anchor point alignment (textalignex1.php)	75
8.3. Example of how to use rotated labels (bargradex1.php)	76
8.4. Example of using rotated data point values (example20.3.php)	76
8.5. The different types of paragraph alignments (textpalignex1.php)	77
8.6. Rendered symbol characters corresponding to	80
8.7. Rendered capital symbol characters corresponding to ???	81
8.8. Specifying manual ticks as fraction of Pi. (manuallickex2.php)	81
9.1. Library cache principle	85
10.1. Example of generated HTML code for StrokeCSIM()	94
10.2. Browser window after calling HTML page in ??? (Note: The image has been scaled down to better fit this manual.)	102
12.1. Commonly used objects in a graph (common-obj-graph.php)	105
12.2. Commonly used objects in Piegraphs (common-obj-piegraph.php)	107
13.1. Post vs. header() data direction	113
13.2. The request phase of a POST header	114
13.3. The reply phase of a POST request	115
13.4. Original graph with all values	118
13.5. Value at x=2 as "	118
13.6. Value at x=2 as '-'	118
13.7. Error when only null values are specified	118
13.8. PHP (HTML) error when missing data keys	119
14.1. Supported principle linear graph types in the library	121
14.2. Using alternating fill colors in the grid (filledgridex1.php)	124
14.3. Using different grid styles for major and minor grids (gridstylesex1.php)	125

14.4. Predefined scientific axis positions	125
14.5. Example of AXSTYLE_BOXIN axis style (funcex2.php)	126
14.6. The various titles in a graph (titleex1.php)	129
14.7. Tabtitle and gradient background (gradbkgex1.php)	129
14.8. Tabtitle and image marker in a line plot (imgmarkerex1.php)	129
14.9. Different line formatting	132
14.10. Using a footer in a graph	133
14.11. Adding a left,right and center footer (footerex1.php)	133
14.12. Adding a timer to the graph (in the footer) (example11.php)	134
14.13. Original plot without clipping (clipping_ex1.php)	136
14.14. Plot with clipping enabled (clipping_ex2.php)	136
14.15. Some example of ways to position the legend box in the graph	137
14.16. Some example of different legend layouts	138
14.17. Basic example of multiple y-axis (mulyaxisex1.php)	140
14.18. Illustration of mulyaxiscsimex1.php	142
14.19. Inverted y-scale to show a dive profile (inyaxisex2.php)	143
14.20. Setting tick density to TICKD_DENSE (manscaleex3.php)	146
14.21. Fully automatic not so good scaling (manscaleex2.php)	146
14.22. Manually specified tick distance which gives a much better appearance (manscaleex1.php)	146
14.23. Manually specifying the tick position for each month (manualtickex1.php)	147
14.24. Adjusting the side which have the tick marks and position the x-axis at the top (topxaxisex1.php)	148
14.25. LABELBKG_NONE (axislabelbkgex01.php)	150
14.26. LABELBKG_XAXIS (axislabelbkgex02.php)	150
14.27. LABELBKG_YAXIS (axislabelbkgex03.php)	150
14.28. LABELBKG_YAXISFULL (axislabelbkgex04.php)	150
14.29. LABELBKG_XAXISFULL (axislabelbkgex05.php)	150
14.30. LABELBKG_XYFULL (axislabelbkgex06.php)	150
14.31. LABELBKG_XY (axislabelbkgex07.php)	150
14.32. The original graph (grace_ex0.php)	152
14.33. Using a 10% grace (grace_ex1.php)	152
14.34. Using a 50% grace (grace_ex2.php)	152
14.35. Using a 100% grace (grace_ex3.php)	152
14.36. The original line graph (example3.2.php)	152
14.37. Adding grace values. Note x-axis position at y=0 (example3.2.1.php)	153
14.38. Adjusting the position of the x-axis manually (example3.2.2.php)	153
14.39. Manual text scale example	154
14.40. Manual text scale with month labels (manual_textscale_ex1.php)	154
14.41. Setting text tick interval=2 (manual_textscale_ex2.php)	155
14.42. Labels at every 2:nd tick mark (manual_textscale_ex3.php)	156
14.43. Tick marks every 40 points and labels every 2:nd tick mark (manual_textscale_ex4.php)	156
14.44. BAND_RDIAG (smallstaticbandsex1.php)	158
14.45. BAND_LDIAG (smallstaticbandsex2.php)	158
14.46. BAND_DIAGCROSS (smallstaticbandsex10.php)	158
14.47. BAND_HLINE (smallstaticbandsex7.php)	158
14.48. BAND_VLINE (smallstaticbandsex6.php)	158
14.49. BAND_HVCROSS (smallstaticbandsex5.php)	158
14.50. BAND_3DPLANE (smallstaticbandsex4.php)	158
14.51. BAND_SOLID (smallstaticbandsex3.php)	158
14.52. SetDensity(10) (plotbanddensity_ex0.php)	159
14.53. SetDensity(40) (plotbanddensity_ex1.php)	159
14.54. SetDensity(80) (plotbanddensity_ex2.php)	159
14.55. Creative use of plot bands	159

14.56. Creative use of plot bands (staticbandbarex7.php)	159
14.57. Use of a static line to simulate an extra x-axis at y=0	160
14.58. Adding a static line at y=0 to simulate an extra 0-axis (impulsex4.php)	160
14.59. Changing the Y2 scale from linear to logarithmic (example7.php)	161
14.60. Enabling minor grid lines on the y-axis and also grid lines on the x-axis (example8.php).....	161
14.61. An example of a log-log plot (where both the y- and x-axis use a logarithmic scale) (loglogex1.php)	162
14.62. Using a text-log scale (example9.php)	163
14.63. Adjusting the text scale so that only every second labels are displayed. (example9.1.php)...	163
14.64. Rotating the x-axis labels 90 degree (example9.2.php)	163
14.65. A first date scale example (dateaxisex2.php)	165
14.66. Manually adjusting the tick labels for a date scale (datescaleticksex01.php)	167
14.67. Adjusting label formatting of a date scale (dateaxisex4.php)	168
14.68. Manually creating a date scale (dateaxisex1.php)	169
14.69. Adding a label at the start of every month (manautickex1a.php)	172
14.70. Manually specified date scale (dateutilex01.php)	175
14.71. Using an automatic date scale (dateutilex02.php)	176
14.72. Different types of shearing transformation	177
14.73. Explaining the shearing parameters	177
14.74. Original unrotated graph (rotex0.php)	179
14.75. Rotating the plot area 45 degrees (rotex1.php)	180
14.76. Rotating the plot area 90 degrees (rotex2.php)	180
14.77. Rotating the plot area 45 degrees (rotex3.php)	180
14.78. Rotating the plot area 90 degrees (rotex4.php)	180
14.79. Rotating the plot area -30 degree around the bottom left corner (rotex5.php)	180
14.80. Plain radar plot (radarex8.php)	182
14.81. Anti-aliased radar plot (radarex8.1.php)	182
14.82. Anti-aliasing up-close. The figure shows the difference between a standard line (on-top) and the corresponding anti-aliased line (on-the bottom)	183
14.83. Affects of using anti-alias for Pie-graphs	183
14.84. Mixing an icon image into the background of the graph. The area plot in the graph uses alpha blending to achieve see-through affect (lineiconex1.php)	185
14.85. Adding a country flag icon in the background (lineiconex2.php)	185
14.86. The graph that will be used to add backgrounds to	187
14.87. Background image (a closeup of our burnt server)	187
14.88. BGIMG_COPY (background_type_ex0.php)	187
14.89. BGIMG_CENTER (background_type_ex1.php)	187
14.90. BGIMG_FREE (background_type_ex2.php)	188
14.91. BGIMG_FILLPLOT (background_type_ex3.php)	188
14.92. BGIMG_FILLFRAME (background_type_ex4.php)	188
14.93. What area of the graph the gradient should affect	189
14.94. Different types of gradient fills	190
14.95. The "mkgrad" utiliy to create gradient images	191
14.96. Adding a text object to a graph (example25.php)	192
14.97. Making the text stand out a bit more by adding a background color and frame (example25.1.php)	192
14.98. Adding a text object with multiple rows of text. Paragraph alignment is set to "center" (example25.2.php)	193
15.1. Supported linear graph types in the library	196
15.2. The most simple line graph (example0-0.php)	198
15.3. Adding some titles (example2.php)	199
15.4. Changing fonts of the axis titles and adjusting plot weight (example3.php)	200
15.5. Adding drop shadow and changing axis color (example3.0.1.php)	201
15.6. Original null values (example3.0.3.php)	201

15.7. Using '-' to get interpolated lines (example3.0.2.php)	201
15.8. Adding basic plot marks to the plot (example3.1.php)	203
15.9. Using one of the built-in images as plot mark, MARK_IMG_DIAMOND (example3.1.1.php)	203
15.10. Using country flags as plot marks (markflagex1.php)	204
15.11. (example3.3.php)	205
15.12. Changing the appearance of data labels (example3.4.php)	205
15.13. Formatting display values as roman numerals (example3.4.1.php)	206
15.14. Adding a second data series (example4.php)	207
15.15. Adding a second y-axis to the graph (example5.php)	208
15.16. Adding and adjusting the position of the legend box (example6.php)	209
15.17. Adjusting the layout of the texts in the legend box (example6.1.php)	209
15.18. Using plot marks with several data series and a legend (builtinplotmarksex1.php)	210
15.19. Using the "Step style" for line plots (example6.2.php)	210
15.20. A basic filled line graph (filledlineex01.php)	211
15.21. Having the grid line on top of a filled line plot (filledlineex01.1.php)	212
15.22. Creating the effect of an area fill with an image (lineimagefillex1.php)	213
15.23. Filling from the 0-line (The default) (manualtickex3.php)	214
15.24. Filling from the bottom (manualtickex4.php)	214
15.25. A basic gradient fill using default values (gradlinefillex1.php)	215
15.26. Using the default number of intermediate colors (gradlinefillex2.php)	215
15.27. Only using 4 colors in total between start and finish color (gradlinefillex3.php)	215
15.28. Adding two partially filled areas to a line plot (partiallyfilledlineex1.php)	216
15.29. Area plot with 'x' NULL values (filledlineex03.php)	217
15.30. A basic accumulated area plot (example17.php)	218
15.31. Area plot with specified x coordinates (prepaccdata_example.php)	223
15.32. Constructing a smooth spline curve from 8 control points (splineex1.php)	227
15.33. Different types of supported bar graphs	228
15.34. Using "int" scale for the x-axis (example19.1.php)	229
15.35. Using "text" scale for the x-axis (example19.php)	229
15.36. An accumulated bar plot (example23.php)	230
15.37. Accumulated bar with individual frame colors (accbarframeex01.php)	231
15.38. Accumulated bar with unit frame color (accbarframeex02.php)	232
15.39. Setting individual frames to weight=0 (accbarframeex03.php)	232
15.40. A grouped bar plot (example21.php)	233
15.41. Adjusting the width of a group bar plot (example22.php)	233
15.42. All data series in a grouped bar graph must have the same number of data points (groupbarex1.php)	234
15.43. A grouped accumulated bar graph (example24.php)	235
15.44. A basic horizontal bar graph (horizbarex1.php)	235
15.45. Using multiple line labels in a horizontal bar graph (horizbarex4.php)	236
15.46. Supported gradient fills for bar plots	238
15.47. GRAD_MIDVER (bargradsmallex1.php)	238
15.48. GRAD_MIDHOR (bargradsmallex2.php)	238
15.49. GRAD_HOR (bargradsmallex3.php)	238
15.50. GRAD_VER (bargradsmallex4.php)	238
15.51. GRAD_WIDE_MIDVER (bargradsmallex5.php)	238
15.52. GRAD_WIDE_MIDHOR (bargradsmallex6.php)	238
15.53. GRAD_CENTER (bargradsmallex7.php)	238
15.54. GRAD_RAISED_PANEL (bargradsmallex8.php)	238
15.55. Horizontal bar graph with gradient fill (horizbarex6.php)	239
15.56. Supported pattern fills for bar plots	240
15.57. Using a callback to format the labels on a bar (barscalecallbackex1.php)	241
15.58. A basic error plot (example13.php)	242

15.59. Making use of SetCenter() with error plots (example14.php)	242
15.60. A basic Line error plot (example15.php)	243
15.61. A line error plot with a legend (example16.php)	243
15.62. A stock graph (stockex1.php)	244
15.63. Explaining stock graphs	244
15.64. A typical boxplot (boxstockex1.php)	245
15.65. A basic scatter plot (scatterex1.php)	246
15.66. Adjusting the size and color of the marker (scatterex2.php)	247
15.67. Combining data points with a dotted line (scatterlinkex3.php)	248
15.68. Combining data points with a red line (scatterlinkex4.php)	248
15.69. Stem plot (impulsex1.php)	248
15.70. Adjusting the overall look and feel for the stem graph (impulsex3.php)	249
15.71. Possible sizes of arrow heads for field plots	250
15.72. A field plot (fieldscatterex1.php)	250
15.73. Using format callback to create a balloon plot (balloonex2.php)	252
15.74. An example with geo maps (pushpinex1.php)	254
15.75. A basic contour graph (basic_contourex01.php)	255
15.76. Adding axis on all sides (basic_contourex02.php)	258
15.77. Flipping the data around the center line (basic_contourex05.php)	258
15.78. Using only 5 isobar lines (basic_contourex04.php)	258
15.79. Interpolation factor=1 (basic_contourex03-1.php)	260
15.80. Interpolation factor=2 (basic_contourex03-2.php)	260
15.81. Interpolation factor=3 (basic_contourex03-3.php)	260
15.82. The exponential growth of the data size due to grid interpolation factor (interpolation-growth.php)	262
15.83. The exponential growth of the data size due to the grid interpolation factor (log scale) (interpolation-growth-log.php)	262
15.84. Mixing a line and area plot in the same graph (example16.1.php)	262
15.85. Mixing a line and bar plot in the same graph (example16.3.php)	263
15.86. Centering the line plot in the middle of the bar (linebarcentex1.php)	263
15.87. Mixing bar and line using an integer x-scale (example16.4.php)	264
15.88. A combination of a line graph at top and a bar graph in the bottom (combgraphex1.php)	264
15.89. Mixing a background image with two subgraphs. In this case the mixing factor was 85 for both subgraphs. (Note: To reduce load time the image is quite hard compressed in JPEG so there are some artifacts in high-frequency areas.) (combgraphex2.php)	268
15.90. Combining three graphs in one image (comb90dategraphex03.php)	269
16.1. Pie graphs	271
16.2. Pie3D graphs	271
16.3. Ring graphs	271
16.4. Radar graphs	271
16.5. Polar graphs	271
16.6. Gantt charts	271
16.7. Hare/Niemeyer pie plot integer compensation	272
16.8. A basic Pie graph (example26.php)	273
16.9. Adding a legend to a pie plot (example26.1.php)	274
16.10. Adding several pie plots to the same pie graph (pieex3.php)	276
16.11. Adding guide lines to a pie labels (pielabelsex1.php)	277
16.12. Lining up guide lines vertically (pielabelsex2.php)	277
16.13. Adjusting the distance between the labels for guide lines (pielabelsex4.php)	278
16.14. A basic 3D pie plot (example27.php)	279
16.15. Adjusting the perspective angle (example27.1.php)	279
16.16. Affect of adjusting the perspective angle for a 3D pie plot	279
16.17. A ring plot (pieceex1.php)	280
16.18. A ring graph with several formatting options adjusted (pieceex2.php)	281

16.19. Exploding the second slice (example27.2.php)	282
16.20. Exploding the second slice (example27.3.php)	282
16.21. Adjusting the position of the pie labels (pieex8.php)	282
16.22. Adding a drop shadow to exploded pie (pieex9.php)	283
16.23. Pie plots with a background image (piebkgex1.php)	285
16.24. (example28.1.php)	285
16.25. (example28.2.php)	286
16.26. (example28.php)	286
16.27. (example28.3.php)	286
16.28. Earth theme	287
16.29. Pastel theme	288
16.30. Sand theme	288
16.31. Water theme	289
16.32. A typical radar graph with two radar plots added (radarex7.php)	289
16.33. A basic radargraph with no formatting (radarex1.php)	291
16.34. A basic radargraph with minimal formatting (radarex2.php)	291
16.35. Adding plot marks to a radar graph (radarmarkex1.php)	292
16.36. Enabling a dashed grid line (radarex4.php)	295
16.37. Enabling a dashed grid line with red (radarex6.php)	295
16.38. Using a logarithmic scale (radarlogex1.php)	295
16.39. Enabling anti-alias for the logarithmic radar example (radarlogex1-aa.php)	296
16.40. A more complex example of a radar graph with a manual scale (fixscale_radarex1.php)	298
16.41. A full 360 degree polar graph (polarex0.php)	299
16.42. A 180 degree (half) polar graph (polarex0-180.php)	299
16.43. A 360 polar plot with background gradient and alpha blending (polarex7-2.php)	300
16.44. Clockwise polar graph (polarclockex1.php)	301
16.45. Rotated clockwise polar graph (polarclockex2.php)	301
16.46. Adding plot marks to a polar graph (polarex7-1.php)	302
16.47. Linear scale for radius (polarex3-lin.php)	303
16.48. Logarithmic scale for radius (polarex3.php)	303
16.49. Logarithmic scale with only major grid lines (polarex4.php)	304
16.50. Logarithmic scale with both major and minor grid lines (polarex5.php)	304
16.51. Different colors for labels, specifying both a tabbed title as well as a axis title (polarex9.php)	305
16.52. A typical small Gantt chart (ganttmonthyearex2.php)	307
16.53. Building block of a Gantt chart	307
16.54. The Gantt scale properties	310
16.55. Gantt chart with day and hour scale enabled (gantthourex1.php)	313
16.56. The simplest possible Gantt graph (ganttex00.php)	319
16.57. Making some minor alterations to the Gantt graph (ganttex01.php)	320
16.58. Specifying a large vertical position (ganttex03.php)	320
16.59. Adding a milestone marker to a gantt graph (ganttex04.php)	321
16.60. Adding a vertical line in the Gantt graph (ganttex06.php)	322
16.61. Adjusting the position of the vertical line within the day (ganttex07.php)	323
16.62. Adding several activity bars on the same row (gantt_samerowex1.php)	323
16.63. Adding a hollow "break" bar (gantt_samerowex2.php)	324
16.64. Example of adding a right marker to the activity bar	325
16.65. A large marker will force the row to become larger since it by default always fills 60% of the allocated height for each row (ganttex08.php)	325
16.66. Adding a caption to a Gantt bar	326
16.67. Adding progress bars of the gantt chart (ganttex14.php)	327
16.68. Modifying the format for the progress pattern (ganttex15.php)	327
16.69. Start to end constraint	328
16.70. Start to start constraint	328

16.71. End to start constraint	328
16.72. End to end constraint	328
16.73. Group markers	330
16.74. Using the CreateSimple() wrapper method (ganttsimpleex1.php)	332
16.75. Using multiple columns as titles for activities (ganttmonthyearex3.php)	333
16.76. Using different fonts for individual columns (ganttcolumnfontsex01.php)	335
16.77. Adding a spanning title over all title columns (ganttmonthyearex4.php)	335
16.78. Built-in icons for Gantt charts	337
16.79. Adding built in icons in titles (gantticonex1.php)	337
16.80. A zoom factor of 0.7 (ganttex13-zoom1.php)	338
16.81. A zoom factor of 1.5 (ganttex13-zoom2.php)	338
16.82. Adding a table title in the top left corner (ganttex16.php)	339
16.83. Gantt divider lines	340
16.84. Adjusting the plot box around the gantt chart (ganttex18.php)	340
16.85. (ganthgridex1.php)	341
16.86. Adding a country flag to the top left corner of the gantt graph (ganttex17-flag.php)	343
16.87. Adding two text objects to a Gantt graph (gantt_textex1.php)	344
16.88. Error message when using an unsupported Locale in Gantt chart	345
16.89. Using Swedish locale. Notice the L for Lordag instead of S for Saturday (ganttex19.php)..	345
16.90. Using multiple title columns with a scale of one day (gantthourminex1.php)	346
16.91. Using multiple markers and indenting titles in the Gantt chart (ganttex30.php)	347
16.92. Using a grouping bar together with constraints (ganttconstrainex2.php)	347
17.1. LEDC_GREEN (ledex1.php)	348
17.2. LEDC_RED (ledex2.php)	348
17.3. LEDC_YELLOW (ledex3.php)	348
17.4. LEDC_BLUE (ledex5.php)	348
17.5. LEDC_GRAY (ledex6.php)	348
17.6. LEDC_INVERTGRAY (ledex17.php)	348
17.7. LEDC_CHOCOLATE (ledex7.php)	349
17.8. LEDC_PERU (ledex8.php)	349
17.9. LEDC_GOLDENROD (ledex9.php)	349
17.10. LEDC_KHAKI (ledex10.php)	349
17.11. LEDC OLIVE (ledex11.php)	349
17.12. LEDC_LIMEGREEN (ledex12.php)	349
17.13. LEDC_FORESTGREEN (ledex13.php)	349
17.14. LEDC_TEAL (ledex14.php)	349
17.15. LEDC_STEELBLUE (ledex15.php)	349
17.16. LEDC_NAVY (ledex16.php)	350
17.17. Supersampling=1 (ledex4.php)	350
17.18. Supersampling=2 (default) (ledex4.1.php)	350
17.19. Supersampling=4 (ledex4.2.php)	350
17.20. LED 4x7 Cyrillic alphabet support	351
17.21. Sample illustration of captcha challenge (antspamex01.php)	351
17.22. A simple canvas graph to draw a text box (canvasex01.php)	355
17.23. Drawing some basic geometric shapes on a canvas (canvasex02.php)	357
17.24. Creating a canvas graph with a scale and using the shape class (canvasex03.php)	360
17.25. Changing the image size to create a smaller version of the previous example (canvasex04.php)	360
17.26. Keeping the image size but changing the scale (canvasex05.php)	360
17.27. Example of using an indented rectangle (canvasex06.php)	361
18.1. Linear regression using utility class (example16.6.php)	365
19.1. Standalone table examples	368
19.2. Combining a graphic table and a bar graph	369
19.3. The most basic 2x4 table (table_howto1.php)	371

19.4. Merging the rightmost 4 cells in the table (table_howto2.php)	372
19.5. Merging the top row (table_howto3.php)	372
19.6. Adjusting the font in the top row (table_howto5.php)	373
19.7. Merging and setting the colors the top row (table_howto4.php)	375
19.8. Setting the minimum column width to 35 pixels. (table_howto6.php)	376
19.9. Double lines 1 (table_howto7.1.php)	377
19.10. Double lines 2 (table_howto7.2.php)	377
19.11. Removing some grid lines and border. In addition we have right aligned all the cells as is common practice for numeric data. (table_howto8.php)	378
19.12. Applying a number format to the data in the cells (table_howto9.php)	378
19.13. Using country flags in the table cells (table_flagex1.php)	379
19.14. xx (table_mex00.php)	382
19.15. xx (table_mex0.php)	382
19.16. xx (table_mex1.php)	382
19.17. xx (table_mex3.php)	383
19.18. Vertical text (table_vtext.php)	383
19.19. Adding an icon (image) in a table cell (table_iconex1.php)	384
19.20. Combining a bar graph and a table (tablebarex1.php)	384
20.1. Odometers	387
20.2. A basic odometer (odotutex00.php)	389
20.3. A full circle odometer (odotutex01.php)	390
20.4. Adding titles and captions (odotutex02.php)	391
20.5. Adding a multi line caption (odotutex03.php)	392
20.6. Changing colors (odotutex04.php)	393
20.7. Possible shapes of the odometer indicator (odotutex06.php)	394
20.8. Illustration of various sizes of arrow heads (odotutex07.php)	395
20.9. Adding drop shadows (odotutex08.php)	396
20.10. Increasing the individual margins around the plots (odotutex09.php)	396
20.11. Adding colored scale indicators (odotutex10.php)	397
20.12. Adjusting the non-colored center area (odotutex11.php)	397
20.13. Adding a second scale indicator (odotutex12.php)	398
20.14. Adding a scale legend by using the label property (odotutex13.php)	399
20.15. (odotutex14.php)	400
20.16. Adjusting the scale format (odotutex15.php)	403
20.17. Adjusting start and end angles of scale (odotutex16.php)	403
20.18. Adding gray areas below the min and max scale values (odotutex16.1.php)	404
20.19. Manually specifying the position of odometer plots (odotutex17.php)	405
20.20. Using layout classes for automatic positioning (odotutex18.php)	407
20.21. Adding an icon and text to a Odometer graph (odotutex19.php)	409
21.1. An basic Windrose plot	411
21.2. WINDROSE_TYPE4	413
21.3. WINDROSE_TYPE8	413
21.4. WINDROSE_TYPE16	414
21.5. WINDROSE_TYPEFREE	414
21.6. A basic 16 direction windrose graph (windrose_ex0.php)	417
21.7. Adding two windrose plots to the same graph (windrose_2plots_ex1.php)	419
21.8. Positioning with LBLPOSITION_CENTER	423
21.9. Positioning with LBLPOSITION_EDGE	423
21.10. Windrose legend methods	425
21.11. Using chinese fonts (windrose_ex6.1.php)	425
21.12. Japanese locale (windrose_ex7.1.php)	428
21.13. Adding label background (windrose_ex8.1.php)	430
21.14. Different legend label styles	432
21.15. Interpretation of ordinal keys (windrose_ex9.1.php)	435

21.16. Adding a "tornado" icon to the top left corner (windrose_icon_ex1.php)	438
21.17. Using layout classes to position 4 windrose plots (windrose_layout_ex0.php)	440
21.18. Using layout classes to position 5 windrose plots (windrose_layout_ex1.php)	442
21.19. (windrose_ex1.php)	444
21.20. (windrose_ex2.php)	446
21.21. (windrose_ex3.php)	448
21.22. (windrose_ex4.php)	450
21.23. (windrose_ex5.php)	452
21.24. (windrose_ex6.php)	454
21.25. (windrose_ex7.php)	456
21.26. Show how to set different styles for individual radial grid lines (windrose_ex8.php)	458
21.27. (windrose_ex9.php)	460
21.28. (windrose_bgimg_ex1.php)	462
22.1. A medium complex example to shows some capabilities of matrix plots (matrix_introex.php)	464
22.2. A basic matrix graph with all default values (matrix_ex0.php)	466
22.3. The effects of changing the value range for the colormap	470
22.4. Using a circular module type (matrix_ex05.php)	471
22.5. Matrix alpha blending=0.2 (matrix_ex04.1.php)	472
22.6. Matrix alpha blending=0.7 (matrix_ex04.2.php)	472
22.7. Matrix legend positions	474
22.8. Adding row and column legends to a matrix plot (matrix_edgeex01.php)	475
22.9. Adding an icon to the lower right corner (matrix_ex03.php)	479
22.10. Adding a background image to the matrix graph (matrix_ex04.php)	481
22.11. Adding plot lines to the matrix plot (matrix_ex06.php)	482
22.12. Using layout classes with Matrix plots (matrix_layout_ex1.php)	485
22.13. Standard color maps	486
22.14. Centered color map	487
22.15. Continues color map	487
23.1. Filled contour with labels (contour2_ex1.php)	491
23.2. Filled contour with no isobar lines (contour2_ex2.php)	491
23.3. Manual colors for contour (contour2_ex3.php)	493
23.4. Labels that follows the gradients (contour2_ex4.php)	494
23.5. Labels that are always horizontal. In this example we have also shown how to change the colors. (contour2_ex5.php)	494
23.6. Triangulation step 0	495
23.7. Triangulation step 1	495
23.8. Triangulation step 2	495
23.9. "Rectangularization" step 0	495
23.10. "Rectangularization" step 1	495
23.11. "Rectangularization" step 2	495
23.12. 7 Isobars, "rect" method (contour2_ex6.php)	496
23.13. 7 Isobars, "tri" method (contour2_ex7.php)	496
23.14. Rectangular sub-division	497
23.15. Triangular sub-division	497
24.1. Understanding linear barcodes. Example with Code 25 symbology	500
24.2. Linear Barcode Demo application (screen shot from running in WEB-browser)	503
24.3. Encoding "ABC123" with CODE 39	504
24.4. Encoding "ABC123" with CODE 128	504
24.5. Image error - Failed barcode data validation	506
24.6. Encoding "ABC123" with CODE 39 adding checksum (checksum=4)	508
24.7. Encoding "ABC123" with CODE 39, hiding the text	510
24.8. UPC A Example	511
24.9. UPC E Encoding of "05510000120"	512

24.10. UPC A Encoding of "05510000120"	512
24.11. EAN 8 Example	513
24.12. EAN 13 Example	513
24.13. EAN 128 Example	514
24.14. Industrial 2 of 5, without check digit	515
24.15. Industrial 2 of 5, with check digit	515
24.16. Industrial 2 of 5, without check digit	515
24.17. Industrial 2 of 5, with check digit	515
24.18. Code 11, without check digit	516
24.19. Code 11, with check digit	516
24.20. Code 39, without check digit	517
24.21. Code 39, with check digit	517
24.22. Code 39 Extended, without check digit	517
24.23. Code 39 Extended, with check digit	517
24.24. Code 128	518
24.25. Codabar	518
24.26. Bookland (ISBN)	519
25.1. PDF417 Structure - Overview	522
25.2. PDF417 Structure - Details of a real barcode	523
25.3. PDF417 WEB-based demo application	525
25.4. Overview of the interaction between encoder and backends	526
25.5. The most basic PDF417 script (pdf417_ex0.php)	526
25.6. Adding error handling (pdf417_ex1.php)	527
25.7. Adjusting the number of columns and error correction level (pdf417_ex1b.php)	528
25.8. Adjusting the module width and showing human readable text (pdf417_ex2.php)	529
25.9. Structure for manually specified encodation schema	529
25.10. Normal PDF417	534
25.11. Truncated PDF417	534
25.12. Showing human readable text (pdf417_ex3.php)	542
25.13. Changing colors (pdf417_ex4.php)	543
25.14. (pdf417_ex6.php)	545
26.1. Datamatrix structure.	546
26.2. Datamatrix - Square symbol shape	548
26.3. Datamatrix - Rectangle symbol shape	548
26.4. Datamatrix encodation principle	551
26.5. The simplest possible datamatrix script (datamatrix_ex00.php)	552
26.6. Datamatrix error image	552
26.7. Datamatrix with basic error handling (datamatrix_ex0.php)	553
26.8. Datamatrix with modified module width (datamatrix_ex1.php)	554
26.9. Datamatrix WEB-based demo application	561
26.10. Datamatrix example, setting quiet zone and ASCII encoding (datamatrix_ex4.php)	562
26.11. Datamatrix postscript output	564
26.12. Datamatrix example, changing colors (datamatrix_ex5.php)	565
27.1. QR Code high level structure	567
27.2. Creative usage of QR Barcodes	567
27.3. A small sized QR Code (version=2)	569
27.4. A medium sized QR code (version=13)	569
27.5. QR Encodation Process	575
27.6. The first very basic QR code (qrexample00.php)	576
27.7. QR Error message	576
27.8. Adding basic exception handling (qrexample0.php)	577
27.9. Adjusting the module width for the QR code (qrexample01.php)	578
27.10. Structure for manually specify QR encodation schema	582
27.11. QR Code - Example Postscript Output	584

27.12. A QR code template (qr_template.php)	586
27.13. QR Code WEB-based demo application	587
27.14. Specified error correction level (qrexample04.php)	589
27.15. QR Barcode with manually specified encodation (qrexample05.php)	590
27.16. QR Barcode with image in JPG format (qrexample06.php)	591
27.17. multiple manually specified encodation schema. (qrexample07.php)	592
27.18. data read from file (qrexample08.php)	593
27.19. (qrexample09.php)	594
27.20. Specified error correction level (qrexample10.php)	595
27.21. QR With ASCII rendering	596
28.1. Synchronized y and y2 scales (y2synch.php)	599
28.2. Using a barplot with two different scales (y2synch2.php)	601
29.1. usps_exhibit44.png	602
29.2. USPS example 1	604
29.3. USPS example 2	605
30.1. Spam statistics	606
31.1. Critical chain buffer penetration. Each white scatter dot represents the state of one task.	622
31.2. Buffer penetration chart for example	624
31.3. Buffer penetration chart with "historic" tail	624
31.4. Complete buffer penetration example with history trace (ccbp_ex1.php)	625
31.5. Using the alternative color map (ccbp_ex2.php)	625
31.6. Steps to create the background	629
A.1. The documentation build process	636
E.1. Small size	674
E.2. Medium size	675
E.3. Large size	675
E.4. Standard size	675
E.5. Standard size	675
E.6. Standard size	676
E.7. Standard size	676
E.8. Standard size	676

List of Tables

1.1. Feature matrix for JpGraph library	7
1.2. External JpGraph tutorial	11
3.1. Latin TTF font file names	23
8.1. Supported Font Styles	70
8.2. Supported Latin Font family specifiers	70
8.3. Supported character entities in class SymChar	79
8.4. Japanese encoding options	82
8.5. Cyrillic encoding options	83
8.6. Greek encoding options	84
10.1. CSIM Examples (in Examples/ directory)	91
14.1. Axis configurations	126
16.1. Gantt bar patterns	326
19.1. Table API Overview	380
20.1. Available arrow size	394
21.1. Default windrose buckets	411
25.1. Available error levels	532
25.2. Recommended error levels	532
26.1. Datamatrix encodation efficiency	549
26.2. Maximum data capacity for the different symbol sizes in ECC-200 Data Matrix subset.	550
26.3. Datamatrix error messages	555
26.4. Datamatrix encodation schemas	556
26.5. Tilde processing translating	557
27.1. QR Data capacity	569
27.2. Maximum data capacity for the different symbol sizes in the QR-code.	570
27.3. QR Error correction levels	574
27.4. QR Error messages	579
D.1. Named color list	661
E.1. Built in line based plot marks	673
F.1. List of all country flags sorted by country name	678
G.1. List of files included in the library	696
G.2. List of subdirectories in main src directory	698
H.1. English error messages	699
H.2. English error messages	710
H.3. English error messages	713

List of Examples

1.1. This is the very first example (example0.php)	5
3.1. Verifying the GD installations (checkgd.php)	20
3.2. Verifying GD2 (checkgd2.php)	20
3.3. Installing GD through RPM packet manager	21
3.4. Verifying TTF with a known font (checkttf.php)	22
3.5. Alias configuration for a development server running Apache with Eclipse-PDT	31
6.1. Throwing a JpGraph exception	62
6.2. Catching a JpGraph exception and sending it back as an image to the client	63
8.1. Specifying and installing a user specified font	78
8.2. Using the SymChar class to display the Greek letter "pi"	79
9.1. Using an automatic cache filename and a 60min timeout of the cached images.	87
9.2. General structure for a CSIM script that uses CSIM	90
10.1. Principles of including CSIM graph in a HTML page	97
10.2. Creating CSIM URL targets to open in same browser window	99
10.3. Creating CSIM URL targets to open in a fresh window	100
10.4. Example of HTML page that includes two Graph CSIM scripts ("Examples/csim_in_html_ex2.html")	101
13.1. Illustration of different types of NULL values in graphs	118
14.1. Adjusting the axis font and color	149
14.2. Adjusting manual text tick interval	155
14.3. Adjusting the interval for the labels	156
14.4. Adjusting both text tick mark and label interval	156
14.5. Different densities for plot bands	158
14.6. Different densities for various plot bands	159
14.7. We want the time adjustment to start on an even quarter of an hour, i.e. an even 15 minute period.	166
14.8. We want the time to start on an even 2 hour	166
14.9. Manually creating a date/time scale (timestampex01.php)	171
15.1. The most simple line graph (example0-0.php)	197
15.2. Adding some titles (example2.php)	199
15.3. A basic scatter plot (scatterex1.php)	246
15.4. An example with geo maps (pushpinex1.php)	253
16.1. A basic Pie graph (example26.php)	272
16.2. Adding several pie plots to the same pie graph (pieex3.php)	275
16.3. Pie plots with a background image (piebkgex1.php)	284
16.4. Having the name of the months as title of the axis	293
16.5. A more complex example of a radar graph with a manual scale (fixscale_radarex1.php)	297
16.6. The simplest possible Gantt graph (ganttex00.php)	319
17.1. A simple canvas graph to draw a text box (canvasex01.php)	354
17.2. Drawing some basic geometric shapes on a canvas (canvasex02.php)	356
17.3. Creating a canvas graph with a scale and using the shape class (canvasex03.php)	359
18.1. Linear regression using utility class (example16.6.php)	364
19.1. The most basic 2x4 table (table_howto1.php)	371
20.1. A basic odometer (odotutex00.php)	389
20.2. Adding titles and captions (odotutex02.php)	391
20.3. Adding a multi line caption (odotutex03.php)	392
20.4. Adjusting the scale format (odotutex15.php)	402
20.5. Using layout classes for automatic positioning (odotutex18.php)	407
21.1. Examples of input data for compass (regular) Windrose plots	415
21.2. Examples of input data for free Windrose plots	415
21.3. A basic 16 direction windrose graph (windrose_ex0.php)	416

21.4. Adding two windrose plots to the same graph (windrose_2plots_ex1.php)	418
21.5. Adding a "tornado" icon to the top left corner (windrose_icon_ex1.php)	437
21.6. Using layout classes to position 5 windrose plots (windrose_layout_ex1.php)	441
21.7. (windrose_ex1.php)	443
21.8. (windrose_ex2.php)	445
21.9. (windrose_ex3.php)	447
21.10. (windrose_ex4.php)	449
21.11. (windrose_ex5.php)	451
21.12. (windrose_ex6.php)	453
21.13. (windrose_ex7.php)	455
21.14. Show how to set different styles for individual radial grid lines (windrose_ex8.php)	457
21.15. (windrose_ex9.php)	459
21.16. (windrose_bgimg_ex1.php)	461
22.1. A basic matrix graph with all default values (matrix_ex0.php)	466
22.2. Adding an icon to the lower right corner (matrix_ex03.php)	478
22.3. Adding a background image to the matrix graph (matrix_ex04.php)	480
22.4. Using layout classes with Matrix plots (matrix_layout_ex1.php)	484
22.5. Matrix example with CSIM (matrix_csimex01.php)	489
25.1. The most basic PDF417 script (pdf417_ex0.php)	526
25.2. Adding error handling (pdf417_ex1.php)	527
25.3. Adjusting the number of columns and error correction level (pdf417_ex1b.php)	527
25.4. Adjusting the module width and showing human readable text (pdf417_ex2.php)	528
25.5. Showing human readable text (pdf417_ex3.php)	542
25.6. Changing colors (pdf417_ex4.php)	543
25.7. (pdf417_ex5.php)	544
25.8. (pdf417_ex6.php)	545
26.1. The simplest possible datamatrix script (datamatrix_ex00.php)	552
26.2. Datamatrix with basic error handling (datamatrix_ex0.php)	553
26.3. Datamatrix with modified module width (datamatrix_ex1.php)	553
26.4. Datamatrix example, setting quiet zone and ASCII encoding (datamatrix_ex4.php)	562
26.5. Datamatrix example, writing to a file (datamatrix_ex6.php)	563
26.6. Datamatrix example, creating postscript output (datamatrix_ex7.php)	564
26.7. Datamatrix example, changing colors (datamatrix_ex5.php)	565
27.1. The first very basic QR code (qrexample00.php)	576
27.2. Adding basic exception handling (qrexample0.php)	577
27.3. Adjusting the module width for the QR code (qrexample01.php)	578
27.4. Generating Postscript output (qrexample11.php)	584
27.5. A QR code template (qr_template.php)	586
27.6. Storing image to a file (qrexample03.php)	588
27.7. Specified error correction level (qrexample04.php)	589
27.8. QR Barcode with manually specified encodation (qrexample05.php)	590
27.9. QR Barcode with image in JPG format (qrexample06.php)	591
27.10. multiple manually specified encodation schema. (qrexample07.php)	592
27.11. data read from file (qrexample08.php)	593
27.12. (qrexample09.php)	594
27.13. Specified error correction level (qrexample10.php)	595
27.14. Using the ASCII backend (qrexample12.php)	596
28.1. (y2synch.php)	600
31.1. Buffer penetration example	624
31.2. (ccbpgraph.class.php)	631

Preface and history of the JpGraph library

PHP has over the last 10 years developed from a rather small language for the enthusiasts to a major platform for WEB based application development. The reason we call it a platform rather than a script language is the fact that when referring to PHP it is impossible not to include the very large library of included standard modules which makes it more of a platform than a scripting language. As such there are more extensions to PHP both in the form of external libraries as well as core language extension than any single person can have a working knowledge of. This means that development teams have to select carefully where best to invest time and training. So how can the JpGraph extension library motivate its existence? Perhaps a quick personal reflection of how this library came about might give some insights.

When I first became aware of PHP at the end of 1998 the PHP 3 version was the new shiny kid on the block after a complete rewrite of the previous PHP2. Initially I proposed to a client I was then working for to use this new script language to build some - what was then - state of the art interfaces to a legacy mainframe system. As part of that some basic graphic capabilities was needed and as part of that work I put together some fairly crude and not very generic small library that would just fit the need to my client. This was not a major part of the development and it didn't make sense at the time to spend a lot of complex design effort into this very small part of the system. After the work for that client was done I felt that I should rewrite the small library I put together so that it would be both a little bit more generic and also a bit more maintainable. (Side note: As part of the contract with the client I was explicitly forbidden to use any object-oriented-technology since my client had been terrible burnt by some previous contractors in this area). Hence the original library was basically a function library and the first thing that was needed was a proper design using a more maintainable OO-design philosophy.

This rewrite of the library and the "plugin" architecture/framework it is based on tried to make the best of the very crude object oriented support available in PHP3 at that time. The support available then was almost nothing more then some syntactic sugar. However, the initial design from then is almost unchanged to this day, almost 11 years later. To the best of my knowledge this makes it one (if not the) oldest object oriented library for PHP that is still actively maintained and developed.

Philosophy of the manual

Writing a good user manual for feature rich library is a bit of a challenge. Learning to efficiently use a library of this size is not a linear undertaking. One start with some basics, jump to a bit that is needed at the moment, (perhaps not fully understanding the details), and then goes back to the beginning again. This makes writing a manual that suits everyone impossible. What we have tried is to mainly follow a "*from-the-basics-to-the-advanced*" style as much as possible.

At certain times we find ourself in the need of some more advanced concept that have not yet been introduced. In those circumstances we will briefly introduce the concept before making use of it and refer to other parts of the manual that discuss the particular feature in more detail. In this way we hope that the more advanced features will at least register as an area for further exploration even early on in the study of the library and help build up the mental map on what is possible to do in the library. This of course has the result that the same information might be in two or three places, but from a slightly different angle, when it is needed. This manual is deliberately not a long list of API - a detailed description of all API and classes are available in the reference manual.

A final note on the examples shown in this manual might be in place. It is custom to keep introductory examples clean of any real error handling in order to keep them short. We do not believe that this is a good idea. Error and exception handling is one of the most crucial part in design and implementation of any system. For this reason we have included, where appropriate, a basic error handling even though it will make some examples slightly larger.

Typographic conventions used in this manual

- Inline code are set in monotype black fonts. Example: Graph::Stroke()
- Filenames mentioned in running text and in titles are set in bluish monotype font. Example: php.ini
- Code snippets or examples are set as block with gray background separated from the text with line numbering. If the code snippet is PHP code it will also be syntax highlighted.

```
<?php
echo "Hello world!";
?>
```

Note that for technical reasons with typesetting this manual even one lines code snippets will have a surrounding of PHP tags i.e. <?php and ?>

- Described inline markup tags, i.e. HTML/XML, are set in a dark red color with light gray background. Example:
- All generated graphs will have the name of the graph file in the title which is hyper-linked to a syntax highlighted full version of the actual source that generated the graph. The link is marked with a small icon and an example is shown in Figure 1, “Link to highlighted graph source”

Figure 1. Link to highlighted graph source

- Numbering of sections are only done down to three level depths, e.g. 3.4.2

Part I. Installing and verifying the configuring

The goal of this first part is to introduce the necessary configurations options in order to install and use the library. It will describe all necessary steps in order to successfully install and verify the installation of the library.

Caution

All files in the library uses utf-8 character encoding to be able to show a wide range of locales. The same applies to all examples. Since some example shows how to use extended character sets , i.e. Chinese and Japanese, and the extended characters are entered directly in the source file the editor/viewer used to view these files must be able to show utf-8 encoded files properly. Since utf-8 character set coincide with the ASCII character set for the usual latin (e.g. a-z, 0-9, etc) characters most files can however be seen with non utf-8 compliant editor/viewers.

Chapter 1. About the library

What you will learn in this chapter. This chapter will go through what the library is, the licensing model, how it is typically used and the detailed steps of installing it. This chapter will not explain how to use the API in the library. It will help answering typical questions such as "*Should I use this library?*", "*Is the library suitable for use in my particular environment?*", "*Do I have enough knowledge to use this library?*"

1.1. What is JpGraph?

The JpGraph library is a 2D graph plotting library for PHP4 and PHP5. It is meant to significantly simplify the creation of dynamic graphs using PHP scripting. The library can be used on its own or as an embedded part of a large WEB development undertaking. In addition the library allows images to be created using the command line version of PHP (the cli version).

1.2. Software license

JpGraph is released under a dual license. For non-commercial usage the library is released under QPL 1.0 (Qt-License) and for professional use it is released under the JpGraph Professional License. See Appendix B, *JpGraph Professional License* for more details regarding the exact wording of the license. The goal the license have is to be simple, fair and help recover some of the development cost and hosting services that the library development brings with it.

Broadly speaking commercial use is defined as

- a) The library is included as an integral part of a product that is sold with a cost that exceeds the cost of the distribution medium.
- b) The library is offered as a WEB service for a fee
- c) The library is used in an intranet in a company with more than 2 employees

One license entitles the user to install the library on one single physical machine which may run one or several logical servers. In addition it also entitles the license holder to install one version of the library on a separate development server. The professional license is perpetual and is valid for one major version.

1.3. Versions of the library covered

This manual covers versions up to v3.1 of JpGraph. There are three main branches of the library:

The "1.x.y" branch

The 1.x.y series is only intended for PHP4 and is not compatible with PHP5 running in strict mode. If you are running an older installation with only PHP4 you must use this branch of the library. In addition you must use this branch if for some obscure reason only the very old GD 1.x library is available in the installation (the GD library is the low level graphic primitives library used by JpGraph and available in PHP).

Caution

The 1.x.y series is since 31 Dec 2008 no longer maintained and should be considered deprecated. There will be no more maintenance releases made on this branch.

The "3.x.y" branch

The "3.x.y" series is the current one and is only intended to run on PHP5. This code is optimized for use both with PHP5 as well as the new (and bundled version) of the GD 2.x library. Hence this will not work if you only have the older GD 1.x graphic primitive library installed. However, all modern PHP installations since 2006 have shipped with GD 2.x so this should not in reality be any problem.

Caution

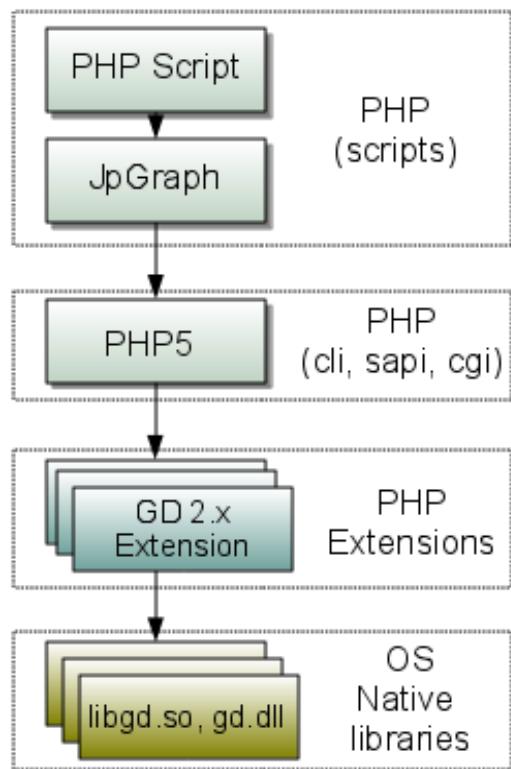
Please note that it is not possible to run the "3.x.y" series on PHP4. If you need to run on PHP4 then you **must** use the 1.x version of the library.

The "3.x.y-p" branch (The pro-version)

This is in principle the same as the "3.x.y" branch with the difference that the pro-version includes a number of additional modules (see Part V, “Additional graph types available in the professional version”) not available in the free version. This includes both some additional graph types as well as 1D and 2D barcodes. In addition the professional license gives three months of email support for installing and configuring the library.

1.4. Purpose and usage

The purpose of the library is to make it possible to (very) easily create dynamic images (a.k.a. graphs) using PHP scripting. The library hides as much as possible all the details necessary to create dynamic images. Strictly speaking all the basic low level functionality to draw images are already available in PHP since PHP comes with the core GD library which is the very lowest layer to create images. The GD library has graphic primitives to create lines, circles, points etc. but it has no built-in intelligence to handle scales, labelling , colors etc. This logically makes JpGraph library a higher type library more easily accessible than the raw GD library.

Figure 1.1. JpGraph and PHP

There are several possible usage scenario for the library and it's different parts.

The most common usage is most likely to visualize numeric data by creating basic charts (for example line, bar or pie charts) that is included dynamically in a WEB-page via a straight forward `` tag. The details on how to create dynamic graphs will be fully covered in later sections of this manual. The library itself is agnostic to where the data comes from so it could for example be retrieved from a database, from a plain text file or perhaps from some WEB-service. In addition to this scenario the library could be used as a tool to create dynamic charts that are stored as image files in a directory. This makes it possible to use the library in an off-line batch mode from the command line (most likely using the cli =command line version of PHP). For an example of using JpGraph in batch mode see Chapter 30, *Showing SPAM statistics*.

In addition to these basic usage scenarios both the free and the slightly more advanced pro-version of the library includes a multitude of additional functionality which includes for example more advanced graph types (like spider graphs, polar plots, contour plots etc.) and some non graph capabilities like the possibility to create barcodes (only available in the pro-version) or to create Gantt-charts. The usage of all of these types of graphs are explained in this manual.

Caution

In order to use JpGraph the PHP installation must have support for the GD libraries enabled. See Chapter 2 and 3 for details on checking the installation.

In order to get a quick feel for how the library can be used we have included in Example 1.1, “This is the very first example (example0.php)” a very basic type of line graph. Don’t worry right now about the details, the message here is that it only takes six lines of real script code to create a basic graph. Have a look at the example and see just how much of the script makes sense without us even having discussed any API yet.

Example 1.1. This is the very first example (`example0.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_line.php");

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

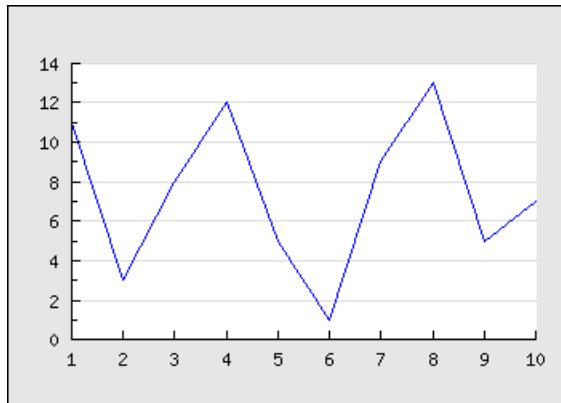
// Create the graph. These two calls are always required
$graph = new Graph(350,250);
$graph->SetScale('textlin');

// Create the linear plot
$lineplot=new LinePlot($ydata);
$lineplot->SetColor('blue');

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

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

Figure 1.2. This is the very first example (`example0.php`) [example_src/`example0.html`]



1.5. Prerequisites for running the library

- Any 32bit operating system capable of running PHP5
- PHP5, requires version $\geq 5.1.0$ (it might work with earlier versions but this is not officially supported)
- Enabled GD 2.x library in PHP installation. This is normally bundled with the PHP5 distribution. It is strongly recommended that you use the included version of GD with PHP5 and not try to install it separately.

Caution

There are known issues so running the library under a 64 bit OS so the library does not officially support any 64 bit OS:es

For more detailed information on PHP settings in `php.ini` see Section 3.2.1, “Verifying the PHP/GD installation”

1.6. Who can use the library

Even though the library hides a lot of details it needs to be pointed out that this is still a programmers library. It is assumed that the user of the library has basic skills in programming PHP and is familiar, at least to a basic extent, with the concept of objects and object oriented programming since the entire library is built around this paradigm. Fully understanding basic concepts such as classes, methods, class instances and class inheritance will substantially help in trying to understand how to use the library.

1.7. What you can do with the library

One should probably differentiate between the two basic usage scenarios

1. **Online.** That is, the image is dynamically generated when a user is viewing a particular WEB-page. This means that the time it takes to generate the image will add to the delay the user is experiencing when trying to view the page. (The library supports a caching mechanism to reduce the number of times an image has to be generated, see Section 5.6, “Efficient graph generation using the built-in cache subsystem” for a thorough discussion). For this scenario one should probably keep the images as basic as possible in order to have as small latency as possible.

In practice this means that the number of data points to visualize should be kept in the order of hundreds and not thousands. In later sections we will discuss in details what can be done to improve the performance of the library.

2. **Offline.** That is, the images are generated by some "batch" processing (possibly command line based). In this scenario the delay is not an issue and one could create much more complicated images and process many more data points. Even though the library in itself does not impose any restriction of the number of data points to process the memory and time limits set for PHP will.

In practice if you need to process images with sizes above 2000x2000 pixels resulting from processing 500,000 data points then it is probably better to find a more suitable way to produce these graphs rather than a PHP script (unless you are prepared to give PHP a couple of 100 MB of allowed memory)

1.8. What you shouldn't or cannot do with this library

If the primary usage is heavy scientific processing where you need to visualize complex 2D/3D scenarios then this library is not for you. In this case it is probably better to use one of the scientific tools like Matlab or Maple.

In addition, if you for example intend to implement a large scale project planning tool with several thousand activities that you want to manipulate and display using the support Gantt charts in this library this is probably stretching it a bit too far. Again, this is probably better done with more dedicated tools like MS Project.

In addition may we again point out that the library does not officially support any 64bit OS due to issues with PHP/GD.

Note

Some of the 2D barcodes will simply not work in a 64bit OS. This is consequence of the computation of the error correcting codes which in some instances assumes 32bit integers. For the basic graphs there is no known issues but since the library is not verified on a 64bit OS we do not officially support this.

There is also a question on sever load that should be taken into account. Due to the CPU intensive nature of image processing the complexity of the generated images needs to be kept as low as possible for any sites that would expect heavy load. Since it is normally necessary to increase the allowed memory for PHP (see the section called "Setting up your php.ini file") when working with images this could easily make the server hit its physical memory limit if the load is very high.

For example, the practical lowest memory that should be set for PHP when working with images is 32MB (recommended is at least 64MB), if your system must cope with 50 simultaneous users it means that the HTTP processes alone will need roughly 1.6GB just to secure the basics. Handling 50 simultaneous executing image scripts will also require some heavy processing and the server needs to have a CPU capacity to handle this. Some discussions about dimensioning a server can be found in Section 5.6, "Efficient graph generation using the built-in cache subsystem".

1.9. Feature-matrix for the library

JpGraph library is an 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 are required to draw graphs with a pleasing esthetic look.

The following list makes no claim to be complete but it will give a birds view of some of the main (and in some cases unique) features of the library. The list will also illustrate the difference between the free and the pro-version.

Table 1.1. Feature matrix for JpGraph library

Feature	Free version	Pro-version
Supports PNG, GIF, JPG image formats		
Flexible scales, supports text-lin, text-log, lin-lin, lin-log, log-lin and log-log and integer scales		
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.		
Supports batch mode to only generate images to a file		
Supports client side image maps which makes it easy to produce drill down images.		

Feature	Free version	Pro-version
Intelligent auto-scaling which gravitates towards esthetic values, i.e. multiples of 2:s and 5:s		
Fully supports manual scaling, with fine grain control of position of ticks.		
Multiple Y-axes (and scales) and multiple data series in the same graph		
Supports background images with different formatting options		
User specified grace for auto-scaling		
Supports unlimited number of y-axes,		
Supports, line-plots, filled line-plots, accumulated line-plots, bar plots, accumulated bar plots, grouped bar plots, error plots, line error plots, scatter plots, gantt-charts, radar plots, 2D and 3D 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		
Designed as a flexible OO framework which makes it easy to add new types of plots		
Supports automatic legend generation with custom formatting		
Supports both vertical and horizontal grids		
Supports anti-aliasing of lines		
Supports background images as well as unlimited number of icons in the graph		
Supports rotation of linear graphs		
More than 400 named colors		
Designed modularly - you don't have to include code which isn't used		
Supports user specified callback for fine tuning scale labels		

Feature	Free version	Pro-version
Support for text augmentation of graphs		
Support for PHP Accelerator		
Support for a large set of 1D bar-codes (EAN-128, ...)		
Support for Windrose plots		
Support for discontinuities in graphs		
Enhanced anti-aliasing for PieCharts		
More advanced formatting of graph titles including 3D Bevel effects.		
Additional 3D Bevel formatting feature for the entire Graph		
Footer text on all graph types		
Full support for color alpha blending		
Advanced interpolation with cubic splines to get smooth curves from just a few data points.		
Several different fill styles for line plots		
Some image 3D effects built-in without external image manipulation programs		
Additional built-in images for plot marks including 3D rendered markers like diamonds, squares, bevels, balls, pins etc		
Support for calculation of linear regression		
Text strings can be added to the plot using scale coordinates		
Support for all primitive URL parameter types with CSIM graphs.		
Hare/Niemeyer Integer compensation for Pie Plots		
Possibility to use Vertical Gradient fill for line plots.		
Improved error handling. The visual appearance of the error handling now tries to mimic any windows system window (in graphic)		

Feature	Free version	Pro-version
Builtin support to display over 200 country flag and the possibility to use them as icons or markers in the graphs. All the flag images are builtin with JpGraph in an efficient pre-compiled data format.		
Support for both Chinese and Japanese character sets		
Support for custom TTF fonts		
Support for 2D contour plots of 3D functions		
Unlimited number of data points (up to memory and CPU limit of server)		
Windrose plots		
Odometer plots		
Graphic excel like tables		
1D Linear barcodes (e.g. EAN8,13,128, Code39, 128, 2of5, Code-11, Codabar etc)		
2D-Barcode PDF417		
2D-Barcode Datamatrix		
2D-Barcode QR-code		
Matrix visualization		

In addition to these high level features the library has been designed to be orthogonal and consistent 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.

1.10. Where to find additional information

The primary source for information is this manual together with the JpGraph web site [<http://www.aditus.nu/jpgraph/>] which also contains further links to many external sources of information.

1.10.1. Manuals and distributed documentation

1. The JpGraph user manual (this document)
2. The JpGraph API reference manual. This reference contains details about all public APIs available together with the classes.
3. JpGraph UML static class diagram. This is an experimental documentation which shows the static class dependency between all classes in the library.
4. The JpGraph community forum [<http://jpgraph.intellit.nl/index.php>]. This is a discussion board for users of the JpGraph library where users both give examples on how they have used the library as well as asking and answering generic and specific questions.

5. The FAQ. This is available both on-line [<http://www.aditus.nu/jpgraph/>] and in Appendix C, *FAQ* to this manual.

1.10.2. On-line documentation and resources

The JpGraph document portal can be found at <http://www.aditus.nu/jpgraph/documentation.php> in addition to the distributed documentation there are several "HowTo" sections which explains with some larger examples how to accomplish some specific type of graphs and solve common problems.

In addition to the JpGraph Web site [<http://www.aditus.nu/jpgraph/>] there are a number of external sites with tutorials and examples on how JpGraph has been used. This is not a complete list (nor will it ever be) but it lists a few of the tutorials/examples that we are aware of. The following is a partial list of known external tutorial showing more or less advanced examples on how to use specific features of the library.

Caution

Usual caveats apply in that we can take no responsibility for the correctness of these tutorials since we have no influence over them.

Table 1.2. External JpGraph tutorial

Name/Link	Description
Chart Dog Application.	Application example
http://www.jimwrightonline.com/php/chartdog_2_0/chartdog.php	
Découverte de la librairie Php JpGraph	French tutorial
http://eric-pommereau.developpez.com/tutoriels/découverte-jpgraph/	
Einführung zu JPGraph	German tutorial
http://www.binnendijk.net/jpgraph/index.php	
Des graphes en Php avec JpGraph	French tutorial
http://www.journaldunet.com/developpeur/tutoriel/php/011121php_jpgraph.shtml	
Developing Professional Quality Graphs with PHP	English tutorial
http://devzone.zend.com/article/1260-Developing-Professional-Quality-Graphs-with-PHP	
PHPHacks.com Posts Creating charts with JP-Graph	English tutorial
http://devzone.zend.com/article/994-PHPHacks.com-Posts-Creating-charts-with-JPGraph	
Simple linear regression with PHP	IBM developer works article
http://www.ibm.com/developerworks/web/library/wa-linphp2/	
Dreamweaver Article	How to integrate with Dreamweaver
http://www.adobe.com/devnet/dreamweaver/articles/php_graphics_11.html	
JpGraph Library V2.2 Tutorial	Typo3 integration

Name/Link	Description
http://typo3.org/documentation/document-library/extension-manuals/rt_jpgraphtutor/current/	
Introduction to JPGraph (Part I)	Basic introduction
http://www.devtutorials.info/articles/1334_Introduction_to_JPGraph_Part_I_.asp	
PHP Graphics With JpGraph	Downloadable tutorials
http://rosihanari.net/web-tutorial/php-graphics/	
JpGraph: PHP Graphs & Charts On-The-Fly	Basic introduction
http://www.communitymx.com/abstract.cfm?cid=2AB8E	
Using JpGraph	JpGraph and Cake
http://bakery.cakephp.org/articles/view/using-jpgraph	
Create High Quality Graphs with Jpgraph	Basic introduction
http://phpkitchen.com/2002/05/create-high-quality-graphs-with-jpgraph/	
JpGraph tutorial	Integration with Prado
http://www.pradosoft.com/wiki/index.php/JpGraph_tutorial	

1.10.3. Defect database

Unfortunately there will be defects (bugs) even in this library. If you suspect that you have found a bug may we ask you to first check the existing bug database if this problem has already been reported. If the problem you have found has not been reported then we would appreciate hearing about your issue.

You can find our defect database at <http://www.aditus.nu/bugtraq/>

Note

We have been forced to add some simple spam protection on the defect reporting system to prevent a lot of phony robot signups

In order to sign-up for a new user account do the following

1. Goto the issue tracker at <http://www.aditus.nu/bugtraq/?do=register> in order to signup for a new account
2. Select an arbitrary user name and prefix it with an "XY_", for example, if your chosen user id would be "adam12" then signup as "XY_adam12".
3. Fill in the rest of the values, your name, e-mail etc.
4. You will receive a confirmation mail in order to activate your account.
5. In order to login you just need to use your chosen user name **BUT WITHOUT** the "XY_" prefix. This is only used when signing up for an account.

1.11. Known bugs and omissions

In general a new version of the library is released twice a year. If you are in a hurry to get a fix that may already be available in the development branch you can always try the latest nightly build available from the JpGraph home page. However be aware that a nightly build could be broken on rare occasions. However, our development philosophy is to always have a releasable repository so for the most of the time the snapshot will work fine. The snapshot is available from:

<http://www.aditus.nu/jpgraph/jpsnapshot.php>

The following is a partial list of what we believe to be the most irritating known bugs (and omissions). See the Bugtracker [<http://www.aditus.nu/bugtraq/index.php>] for a more current view on additional known minor issues with the library.

1. For performance reasons background images are not rotated along with graphs in rotated graphs. Images rotation must be made outside the library with some image manipulation program. This will never be included as a feature of the library since PHP is simply too slow for this kind of pixel-by-pixel image manipulation.
2. The library does not work with a 64bit OS (This is partly due to PHP and partly due to some calculation of Error codes that assumes 32 bit integers)
3. Truetype fonts are aligned according to their bounding box. This means that text lines that contain characters below the baseline (e.g. "j") will be aligned slightly different from texts that only contain characters that are above the baseline (e.g. "a").

Chapter 2. The Short Version: Installing the library (for PHP/Apache experts)

What you will learn in this chapter. This chapter will show you how to unpack the library and lists the minimum prerequisites that is needed to get the library running. It is assumed that you are familiar with adjusting the PHP configuration file `php.ini` and that you already have a working PHP installation.

2.1. Installing

Please follow the steps below:

1. Either download the free library from <http://www.aditus.nu/jpgraph/jpdownload.php> or use your license information and download the pro-version from http://www.aditus.nu/jpgraph/pro_login.php. The download is just a packed zip (or tar.gz) of php files. There are no automatic installation scripts.
2. Unpack the library where you normally store PHP libraries. This should be in your PHP include path. When you unpack the library it will be named "jpgraph-3.x" (where x corresponds to the version you have downloaded).
3. Now either rename the unpacked directory to just `jpgraph` or if the system us Unix based create a soft symbolic link, for example

```
ln -s jpgraph-2.x jpgraph
```

This will allow you to access the library files in your own program for with a `require_once('jpgraph/jpgraph.php')`

4. Make sure that the GD extension is enabled in your `php.ini` file (check the output from `phpinfo()`)
5. [Optional] Verify that the paths defined in `jpg-config.inc.php` corresponds to the server setup. The path to your TTF fonts must be correct. If you get an error saying that some TTF fonts cannot be found or read then the path needs to be adjusted in this configuration file.
6. [Optional] Check that you have sufficient memory and execution time set in your `php.ini` file. It is recommended to allow at least 32Mb memory for PHP if you intend to run anything else than very basic graph scripts.
7. [Optional] To allow for better debugging in conjunction with graph scripts the output buffering should also be disabled in `php.ini`
8. [Optional] To avoid warning messages it might be necessary to set the default timezone in `php.ini` if this has not already been done. Starting with PHP 5.2 a warning is generated if the timezone for PHP is not set.

Tip

When you install the library on a production server then you should not install the library in the document root. Instead it should be installed so that only the script can directly access the library files (somewhere in the PHP path).

If you have the pro-version you really should install the pre-compiled version of the library (available under directory `PhpExpress-src`). This will significantly increase the performance of the library. The only prerequisite is that you also need to install the (free) `PhpExpress PHP Accelerator` from NuSphere Corporation. See Chapter 11, *NuSphere PHP accelerator* for information on how to install the freely available PHP Accelerator.

2.2. Running the examples

The best way to verify that the installation is working is to run one of the included example scripts. Assuming that the library is installed locally in the document root on the HTTP server under `jpgraph` pointing the browser to "`http://localhost/jpgraph/Examples/example0.php`" should show the same image as can be seen in Example 1.1, "This is the very first example (`example0.php`)".

In order to generate all the examples available there is a "meta script" that will just do this. Point the browser to

```
"http://localhost/jpgraph/Examples/testsuit.php"
```

just note that this will generate a page with more than 300 examples. Depending on the capacity of the server it might take a few seconds before this script is done.

2.3. Basic trouble shooting

If none of the examples above work there is a couple of things that needs to be checked. If not any of these quick fixes solves the problem then it is best to read the long version of the installation instructions before continuing the trouble shooting.

1. Verify that the GD library is really working by running `phpinfo()` and check that your output includes a GD section which should show similar information as the image below

Figure 2.1. `phpinfo()` GD sections

gd	
GD Support	enabled
GD Version	bundled (2.0.34 compatible)
FreeType Support	enabled
FreeType Linkage	with freetype
FreeType Version	2.3.7
T1Lib Support	enabled
GIF Read Support	enabled
GIF Create Support	enabled
JPG Support	enabled
PNG Support	enabled
WBMP Support	enabled
XPM Support	enabled
XBM Support	enabled

2. Make sure you have enabled strict error checking and disabled output buffering in `php.ini`. This might not solve the problem by itself but it will enable (better) error messages to trouble shoot the installation.
3. Make sure that you are using a recent version of PHP (e.g. # PHP 5.2.x)

4. Make sure that the path to the TTF directories are correctly specified in `jpg-config.inc.php`

Chapter 3. The Long Version: Installing the Library

What you will learn in this chapter. You will learn in detail how to configure and setup an environment to be able to run the JpGraph library. The chapter will explain all configuration necessary in great details and will also list some more extensive trouble shooting steps if you encounter problems trying to get the library running. The only assumption is that you have a working PHP installation together with your HTTP server.

3.1. Downloading the library

Downloading the free version

Point the browser to "http://www.aditus.nu/jp-graph/jpdownload.php" and select the 3.x series if you are running PHP5 and the 1.x series if you are still running a PHP4 installation.

Note

The 1.x series of the library is no longer maintained so in case you have not yet upgraded top PHP5 you should strongly consider this.

Downloading the pro version

1. Locate your license file that was sent to you when you purchased the library. You will need both the license number as well as the registered license mail as stated in the license. Your license information should look similar to:

```
-----  
: Your license key : JPGP-0920-1234657  
: Name on license : A. JpGraph User  
: E-mail : jpgraph-user@example.com  
-----
```

2. Point the browser to http://www.aditus.nu/jp-graph/pro_login.php and enter your licence email and key on the website.

Figure 3.1. Pro-login dialogue on JpGraph Website

> [Home](#) > [Professional Version](#) > [Pro Login](#)

JpGraph Pro Login

E-mail:	<input type="text"/>
License key:	<input type="text"/>
<input type="button" value="Login"/>	

3. Depending on your system you should now download either the 1.x or the (preferred) 3.x series. There is a choice to either download the library as a "*.zip" file or as a compressed "*.tar.gz" file (which is the preferred format in a Unix environment). Save the downloaded file to a temporary directory of your choice.
4. [Optional] Verify your downloaded file against the given MD5 sum on the website. On a Unix system this can be done by running the **md5** program and give the downloaded library as the argument.

For example: \$> **md5** jprgraph-3.0.0p.tar.gz

5. Unpack the library to a suitable directory that is in your PHP include path as is described in Section 3.3, "Installing the library"

3.2. Necessary system requirements for the library

In order to run the JpGraph library there are some prerequisites that must be fulfilled. The PHP installation must have the low level graphic primitive library "GD" enabled and if TTF fonts shall work in the graphs the FreeType 2.x library must be enabled and installed in the PHP setup. Most modern PHP systems (since at least 2006) usually have these extensions enabled and installed by default which means it is usually rather simple to get the library working.

However, there are many systems out there and some older non-standard systems will require some manual intervention to work correctly.

Caution

One more time; The library is not guaranteed to run on a 64Bit OS. We will point out this several more time during this manual.

3.2.1. Verifying the PHP/GD installation

The first and most important step is to make sure that your PHP installations has the GD library enabled. The easiest way to find out if this has been enabled or not is to create a one line PHP program that has only one instruction, `phpinfo()`

```
<?php phpinfo(); ?>
```

Store this basic program as `phpinfo.php` in the document root. Then point the browser to

`http://localhost/phpinfo.php`

If the steps above was followed the browser should now show a lot of information about the PHP installation. In order to find out if the GD have been installed and enabled look for the GD section in the output. The figure below shows a typical output of this section

Figure 3.2. phphinfo() GD-Information

gd	
GD Support	enabled
GD Version	bundled (2.0.34 compatible)
FreeType Support	enabled
FreeType Linkage	with freetype
FreeType Version	2.3.7
T1Lib Support	enabled
GIF Read Support	enabled
GIF Create Support	enabled
JPG Support	enabled
PNG Support	enabled
WBMP Support	enabled
XPM Support	enabled
XBM Support	enabled

There are three important points to notice here.

1. The GD Version. It should be 2.0.x. The GD version has been shipped with PHP for the last four years so this should not really be a problem. If for some odd reason the system only have GD 1.x installed then JpGraph 2.x, and 3.x cannot be used but it is still possible to use JpGraph 1.x.

In addition the "bundled" version of GD should be used. This version is maintained together with PHP and is usually much more up to date then the stand alone GD version.

2. FreeType version. This is required in order to use TTF fonts. This should be at least version 2.3.x Previous versions of the FreeType library have had known issues which has caused issues when used together with JpGraph.
3. The final point worth checking is what type of image encoding the installation supports. The most common formats are PNG and JPEG. By default the library uses the PNG encoding format so it is important to check that the line that says "PNG Support" has "enabled" as value. As a rule of thumb PNG usually gives the smallest sizes of graph (best compression) so this is the recommended format. The only exception to this rule might be if a photo is used as background in the graph. Then it is possible that JPEG encoding gives a better compression since this is a destructive format (while PNG is a lossless format).

For legacy reason it is also possible to use GIF encoding if the server supports this but if PNG is installed there are no good technical reasons to use GIF compression format. (The only possible usage of GIF is the ability to create an animated image by concatenating a number of GIF images which are then displayed in sequence. This is not possible with the (old) PNG standard.)

In order to remove any remaining doubts that the installation works as intended the following PHP script that only uses the GD library primitives should be run. This allows troubleshooting the installation without involving the additional complexity of JpGraph to make sure that the basics are in place.

Store the script in Example 3.1, “Verifying the GD installations (checkgd.php) ” your document root as “checkgd.php”

Example 3.1. Verifying the GD installations (`checkgd.php`)

```
<?php // content="text/plain; charset=utf-8"
$im = @imagecreate (200, 100) or die ( "cannot create a new gd image." );
$background_color = imagecolorallocate ($im, 240, 240, 240);
$border_color = imagecolorallocate ($im, 50, 50, 50);
$text_color = imagecolorallocate ($im, 233, 14, 91);

imagerectangle($im,0,0,199,99,$border_color);
imagestring ($im, 5, 10, 40, "a simple text string", $text_color );
header ("Content-type: image/png");
imagepng ($im);
?>
```

Figure 3.3. Verifying the GD installations (`checkgd.php`) [example_src/`checkgd.html`]



Now point the browser to this file and fetch the script. This should show an image in the browser with the text "a simple text string" similar to what is shown in Figure 3.3, "Verifying the GD installations (`checkgd.php`)"

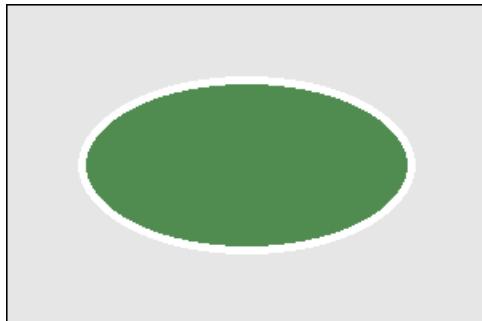
In order to be really sure that the GD 2.x version of the GD library is installed (as opposed to just GD 1.x) we need to use a slightly more advanced script that makes use of some features available in GD 2.x but not in GD 1.x. The largest difference is the support for TrueColor images in GD 2.x so we create a script that makes use of a truecolor canvas..

Example 3.2. Verifying GD2 (`checkgd2.php`)

```
<?php // content="text/plain; charset=utf-8"
$im = imagecreatetruecolor ( 300, 200);
$black = imagecolorallocate ($im, 0, 0, 0);
$lightgray = imagecolorallocate ($im, 230, 230, 230);
$darkgreen = imagecolorallocate ($im, 80, 140, 80);
$white = imagecolorallocate ($im, 255, 255, 255);

imagefilledrectangle ($im,0,0,299,199 , $lightgray);
imagerectangle ($im,0,0,299,199,$black);
imagefilledellipse ($im,150,100,210,110,$white);
imagefilledellipse ($im,150,100,200,100,$darkgreen);
header ("Content-type: image/png");
imagepng ($im);
?>
```

Figure 3.4. Verifying GD2 (checkgd2.php) [example_src/checkgd2.html]



In the same way as before store this file on the server and point the browser to it. The result should be the same image as is shown in Figure 3.4, “Verifying GD2 (checkgd2.php) ”

In order to use the library it is absolutely necessary that the above two scripts work as described. If this is not the case the php.ini file needs to be checked as described in the next section.

3.2.2. Enabling GD library in php.ini for PH5

If neither of the two examples above worked and the GD section didn't exist in the output from `phpinfo()` the chances are that the GD library has not been enabled in the `php.ini` file.

The first problem to solve is therefore to locate the `php.ini` file in the installation. Depending on the type of server (and OS) this can be in different places. If the system is running on a Unix server the `php.ini` file is most likely stored in either `/etc/php.ini`, `/etc/php/php.ini`, `/etc/php5/php.ini` or `/etc/php5/apache/php.ini`

Tip

Check the output of `phpinfo()` to find out what `php.ini` file the installation is reading.

On a Windows server there is no standard location since it completely depends on how the installation of the HTTP server and PHP was done. For example if the WAMP server was installed and put in the top directory `c:\wamp\` the `php.ini` file is installed under `c:\wamp\bin\apache\apache2.2.11\bin\php.ini`

Open `php.ini` in a editor and locate the line

```
extension_dir = <some-path-here>
```

The directory path specified above is the directory where all PHP extension modules are stored. On a Unix system this is typically specified as `/usr/lib/php5/extensions` now examine that directory and check if an extension called "gd.so" (for Unix) or "gd.dll" (for Windows) exists (and is un-commented). If this extension cannot be found then this extension needs to be installed. For example, many Unix/Linux installations require that the extra modules in PHP are manually enabled/installed through the appropriate Package manager as the example below shows.

Example 3.3. Installing GD through RPM packet manager

```
$ /> sudo zypper install php5-gd
```

3.2.3. Verifying TTF fonts

This section is only intended to verify if that support exists for TTF fonts. If this does not work or if it is already known that there are no support for TTF fonts then please go directly to Section 3.4, “Installing and configuring Font support”

In order to check if basic TTF font support is available create the following script in the document root and name it `checkttf.php`

Then it is necessary to locate on the system exactly where the TTF fonts are stored and update the defines for the paths and/or the name of the TTF fonts that is know to exist.

Example 3.4. Verifying TTF with a known font (`checkttf.php`)

```
<?php // content="text/plain; charset=utf-8"
// Change this defines to where Your fonts are stored
DEFINE("TTF_DIR","/usr/share/fonts/truetype/");
// Change this define to a font file that You know that You have
DEFINE("TTF_FONTFILE","arial.ttf");
// Text to display
DEFINE("TTF_TEXT","Hello World!");
$im = imagecreatetruecolor(400, 100);
$white = imagecolorallocate($im, 255, 255, 255);
$black = imagecolorallocate($im, 0, 0, 0);
$border_color = imagecolorallocate($im, 50, 50, 50);
imagefilledrectangle($im, 0, 0, 399, 99, $white);
imagerectangle($im, 0, 0, 399, 99, $border_color);
imagettftext($im, 30, 0, 90, 60, $black, TTF_DIR.TTF_FONTFILE, TTF_TEXT);
header ("Content-type: image/png");
imagepng ($im);
?>
```

Figure 3.5. Verifying TTF with a known font (`checkttf.php`)
[[example_src/checkttf.html](#)]



Hello World!

As usual point the browser to this script and fetch the image. If thing works an image identical to what is shown in Figure 3.5, “Verifying TTF with a known font (`checkttf.php`) ” should be visible.

Name of TTF font files

The library has built in support for large number of TTF fonts, this includes both the standard MS WEB core fonts as well as Vera and DejaVu fonts. In addition a number of non-latin fonts are supported. This includes both Japanese, Chinese, Hebrew and Russian fonts. See Section 3.4.2, “Using non-latin based

fonts with JpGraph” and Section 8.8, “Character encoding” for more on how to use non-latin fonts and encodings.

In order for the fonts to be usable by the library the font files must have the following names (these are the standard names)

More information on how to use TTF fonts in scripts can be found in Chapter 8, *Text and font handling*

Table 3.1. Latin TTF font file names

Font family name	Symbolic name	Normal, FS_NORMAL	Bold, FS_BOLD	Italic, FS_ITALIC	Bold italic, FS_BOLDITALIC
Courier	FF_COURIER	cour.ttf	courbd.ttf	couri.ttf	courbi.ttf
Georgia	FF_GEORGIA	georgia.ttf	georgiab.ttf	georgiai.ttf	
Trebuchet	FF_TREBUCHE	trebuc.ttf	trebucbd.ttf	trebucit.ttf	trebucbi.ttf
Verdana	FF_VARDANA	verdana.ttf	verdanab.ttf	verdanai.ttf	
Times roman	FF_TIMES	times.ttf	timesbd.ttf	timesi.ttf	timesbi.ttf
Comic	FF_COMIC	comic.ttf	comicbd.ttf		
Arial	FF_ARIAL	arial.ttf	arialbd.ttf	ariali.ttf	arialbi.ttf
Vera	FF_VERA	Vera.ttf	VeraBd.ttf	VeraIt.ttf	VeraBI.ttf
Vera mono	FF_VERAMONO	VeraMono.ttf	VeraMoBd.ttf	VeraMoIt.ttf	VeraMoBI.ttf
Vera serif	FF_VERASERIF	VeraSe.ttf	VeraSeBd.ttf		
(Chinese) Simsun	FF_SIMSUN	simsun.ttc	simhei.ttf		
Chinese	FF_CHINESE	bkai00mp.ttf			
(Japanese) Mincho	FF_MINCHO	ipamp.ttf			
(Japanese) Mincho	P FF_PMINCHO	ipamp.ttf			
(Japanese) Gothic	FF_GOTHIC	ipag.ttf			
(Japanese) Gothic	P FF_PGOTHIC	ipagg.ttf			
(Hebrew) David	FF_DAVID	DAVIDNEW.TTF			
(Hebrew) Miriam	FF_MIRIAM	MRIAMY.TTF			
(Hebrew) Ahron	FF_AHRON	ahronbd.ttf			
DejaVu Sans Serif	FF_DV_SANSSEIF	DejaVuSans.ttf	DejaVu-Sans-Bold.ttf	DejaVu-Sans-Oblique.ttf	DejaVu-Sans-BoldOblique.ttf
DejaVu Sans Serif Mono	FF_DV_SANSSEIFMONO	DejaVuSansMono.ttf	DejaVu-Sans-Mono-Bold.ttf	DejaVu-Sans-Mono-Oblique.ttf	DejaVu-Sans-Mono-BoldOblique.ttf
DejaVu Sans Con-densed	SansCon-FF_DV_SANSSEIFCONDENSED	DejaVuSansCondensed.ttf	DejaVu-SansCon-densed-Bold.ttf	DejaVu-SansCon-densed-Oblique.ttf	DejaVu-SansCon-densed-BoldOblique.ttf

Font family name	Symbolic name	Normal, FS_NORMAL	Bold, FS_BOLD	Italic, FS_ITALIC	Bold italic, FS_BOLDITALIC
DejaVu Serif	FF_DV_SERIF	DejaVuSerif	DejaVuSerif-Bold	DejaVuSerif-Italic	DejaVuSerif-BoldItalic
DejaVu Serif Condensed	FF_DV_SERIFCONDENSED	DejaVuSerifCondensed	DejaVuSerifCondensed-Bold	DejaVuSerifCondensed-Italic	DejaVuSerifCondensed-BoldItalic

Note

For the DejaVu fonts there are actually two common sets of name in usage depending on how long it has been since the fonts where installed. The library knows of both sets of names and will try them in order selecting to use the first one available.

3.2.4. Support for different image formats

By default the standard GD image library supports the PNG graphic format.

Hence by default only PNG is supported. If JPEG support is needed this might require additional libraries that must be installed and enabled for usage by PHP/GD (in the `php.ini` file), again please see PHP documentation for specifics. For most practical purposes PNG is a better format since it normally achieves better compression then GIF (typically by a factor of 2 for the types of images generated by JpGraph).

In comparison with JPEG format PNG is also better for the type of images generated by this library.

So, the bottom line is that there should be very good reasons to choose any other image encoding formats than PNG. By default the image format is set to "auto". This means that JpGraph automatically chooses the best available image encoding format using the preferred order "PNG", "GIF" and "JPG".

No support for SVG

We have received may requests to add support for SVG image output. Unfortunately we have investigated this and concluded that it is (surprisingly!) not technically possible to do this. The main reason is that with the current standard there is no way to statically determine the bounding boxes for texts. This is further described in Appendix K, *Why it is not possible to add a SVG backend to JpGraph*

3.3. Installing the library

When you have verified the necessary preconditions as described in the previous paragraphs it is time to install the library. The "installing" part is nothing more than copying the files in the distribution to a place in the directory structure where your script can find the library files. On a Unix system it is common to install PHP libraries under `"/usr/share/php/"`. On a windows system there is really no standard path for installing PHP libraries so you have to decide your self.

The important thing here is that the path to the library is included in the PHP search path, i.e. it is in one of the paths that PHP searches when it tries to resolve a `"require_once"` or `"include"` statements. Furthermore, the included examples and demo applications (included in the pro version) assumes that the library is installed under the directory `"jpgraph/"`.

As an example the following commands will install a specific version of the library on a Unix server. If we assume that you have downloaded the library to the `" /tmp/"` directory and are already standing in this directory the following commands will setup the library to be used

```
root:/tmp> tar xzf jpgraph-2.5.tar.gz
root:/tmp> cp -r jpgraph-2.5 /usr/shar/php/
root:/tmp> ln -s /usr/shar/php/jpgraph-2.5 /usr/shar/php/jpgraph
```

The last line makes a symbolic link from "jpgraph" to the actual version of the library. This way you can try out different versions of the library without having to make any changes in your scripts. You just point out a different version of the library in the symbolic link.

3.3.1. Configuring JpGraph/PHP on a development server

Setting up your php.ini file

Tip

To find the location of your `php.ini` file create and run a script with the single line `<?php phpinfo(); ?>`. The look at the output for a line saying "php.ini file used" and you will see which `php.ini` file is used.

Setting the memory limits

In many default configuration the allowed memory for PHP is not enough for complex graph script since they (as many other image manipulation programs) can require a lot of memory. On a development server there should be at least 32MB memory allowed for the HTTP/PHP process. To verify this do the following

1. Open `php.ini` for editing.
2. Locate the line saying

```
memory_limit = xx
```

where "xx" is some number. Now make sure that you have at least 32MB allowed by making sure the line reads

```
memory_limit = 32M
```

Note that for very large images this might not be enough. Consider the following example.

Assume you need to create an 1200x1024 image in true color. Just the plain image in itself will require 1200x1020x4 bytes, which is roughly 4.7MB RAM during internal processing the library can need up to three times that amount of memory so this means that just for the image the library needs around of ~15MB of RAM. If we then take the memory needed to load PHP as well as the entire JpGraph library and dynamically execute and parse the library it can easily consume another ~15MB RAM. If the image is very complex and requires a huge number of objects to be created (a typical example is a large Gantt chart) it might be necessary to double the allowed memory to 64MB RAM.

Setting maximum allowed run time

By default many installations have very short maximum run time for the PHP scripts. Common figures are 10s. For normal interactive use involving plain text processing this is usually adequate. However, producing large and complex images might take considerable time (as do all images processing). For this reason the maximum time limit for PHP should be increased to a minimum of 20s (depending on the complexity of your images as well as any associated data processing it might be necessary to allow up to 30-40s).

The allowed running time is controlled by the `php.ini` setting

```
max_execution_time = xx
```

where "xx" is some number. Recommended setting is therefore

```
max_execution_time = 30
```

Disabling output buffer

The next part of the `php.ini` file that might need changing is the output buffer. In short this should be disabled and we will shortly explain why. To check this do the following

1. Open `php.ini` for editing
2. Locate the line saying

```
output_buffering = xx
```

where "xx" is some number. Make sure that this line is commented out, i.e. it reads

```
; output_buffering = xx
```

This reason we want this to be commented out is that during development we want to be able to see the potential error messages produced by the library and having the output buffering enabled will actually prevent this. Fully understanding why this is the case is good first step into the added complexity of producing images with PHP compared with just outputting text. Understanding this requires us to understand a few basic principles about the HTTP protocol. Especially how MIME encodings of data works.

The following explanation is slightly simplified since a full description of the HTTP protocol would bring us a bit to far in this manual

1. A client (e.g. browser) requests data from the server by issuing a GET (or possible a POST) command to the server. This is what happens when you enter a URI in the address bar in the browser.
2. The server replies with a data stream (or an error if the requested data wasn't available). This data stream is prepended with header (MIME header) that tells the client (e.g. the browser) how to interpret the data that follows. The most common type (and the default type if no header is sent by a faulty server) is "text/html". This tells the client to interpret the data as plain text with embedded HTML encoding.

When the data is to be interpreted as an image the header will instead be one of the image headers, for example "image/png" or "image/jpeg". When the client receives this header it will Interpret all the following data as an image encoded in the indicated format.

The important thing to keep in mind here is that each server reply can have one and only one MIME type. This is the key to further understanding the specific issues with dynamic image generation. This explains why if a PHP script running on the server sends a header first indicating that the following data it sends should be interpreted by the client as an image it cannot send both image data and some text.

We are now in a position to explain how output buffering would make debugging more difficult.

Normally all output from a PHP script is sequentially, i.e. the header must first be sent and then the data. If no header is sent or plain text is sent without a header the client will interpret this as "text/html". One purpose with "output_buffer" is to circumvent this to allow a certain amount of output to be put in a buffer for a while and later when some processing has determined what header should be sent the data is prepended with the correct header and the rest of the data is then sent.

What could now happen is the following (not unlikely scenario):

1. The script starts executing and the image starts to be build.
2. Your script has some minor issues which produces some warnings from PHP. These warning does not get sent directly back to the client (the browser) to allow you to act on these warnings instead they will be put into the output buffer. When later the script starts outputting the proper image header and the image data it gets added to the output buffer where your previous textual PHP warning already are stored.
3. Your client now receives the header that indicates that the following data should be interpreted as an image but since that image data is mixed with the textual warning messages it will fail to decode the data (since it is not proper image data) and will typically just show the image as a square with a red-cross (FireFox) or some message along the lines of "*Cannot decode image*". This is all depending on how a certain client handles a corrupt image.

The above scenario makes it impossible to debug your script since it will give no clue to what caused or where in your script these warnings were generated. The way to counteract this scenario is to disable output buffering. In this way the warning will be sent back to the client as soon as they are generated by PHP and will allow you to act on them.

Enabling adequate error checking

The final part of the `php.ini` file that should be adjusted (and this is not only for the JpGraph library) is the error level. To ensure maximum interoperability of the developed scripts they should all run completely silent no matter what error levels are set on the server. This means that development of all scripts should always be done with maximum error checking enabled. The JpGraph library can safely run completely silent even when all error checking is enabled.

The error checking should therefore be specified as

```
error_reporting = E_ALL | E_STRICT
```

to enable the highest degree of PHP error checking

Tip

In addition to the above setting it is a good idea to also make sure that the following options are set

```
zend.zend_compatibility_mode = Off
```

Zend engine 1 compatibility might cause problems with the library

```
implicit_flush = On
```

This can reduce the performance and shouldn't be used on a production server but will make all outputs sent back to the client as soon as possible and will aid in debugging.

```
allow_call_time_pass_reference = Off
```

This is just a general good idea since call time pass references is deprecated in PHP 5.0 and higher

```
display_errors = On
```

This makes sure all error are displayed

```
display_startup_errors = On
```

This makes sure that any initial errors thrown by PHP will be reported

Setting default timezone

Starting with PHP 5.2 a warning will now be generated unless a default time zone is explicitly specified in `php.ini`. To set this find the line `date.timezone` in the [Date] section and set this to valid zone. For example to specify GMT+1 one could specify

```
date.timezone = Europe/Paris
```

Note: There should be no citation signs around the time zone.

Caution

In order to use the LED module (See Section 17.1, “LED bill boards”) the PHP installation must have multi-byte strings enabled so that the function `mb_strlen()` is available. This is normally enabled at compile time for PHP by specifying the options `--enable-mbstring --enable-mbregex` when configuring the compile options.

Caution

In order to use the PDF417 barcode module (See Chapter 25, *PDF417 (2D-Barcode)*) it is necessary for the PHP installation to support the function `bcmath()`. This is enabled when compiling PHP by making sure that the option `--enable-bcmath` is given when configuring PHP at compile time.

Setting up your jpg-config.inc.php

Apart from the standard configuration described in Section 3.4, “Installing and configuring Font support” and Section 3.5, “Adapting and customizing the installation” there is only one important configuration that is specific for a development server and that is the localization setting for error messages.

As of version 3.0.0 there are three localization options

1. English error messages ("en")
2. German error messages ("de")
3. Production error messages ("prod"). This is not really a localization but a different set of error messages which does not give detailed error messages but a generic message suitable for a production server where the end user is not helped by detailed graph script errors. Instead a generic message is shown together with an error code that corresponds to the detailed error. ("prod")

In order to specify the error message localization the following define in `jpg-config.inc.php` must be set the

```
define('DEFAULT_ERR_LOCALE', 'en');
```

The possible options are

1. "en", English locale
2. "de", German locale

3. "prod", The production version of the error messages.

Tip

In addition to specifying the locale in the `jpg-config.inc.php` file it can also be specified dynamically in each script by calling

```
JpGraphError::SetErrLocale($aLocale);
```

3.3.2. Configuring JpGraph/PHP on a production server

Setting up your `php.ini` file

Apart from what is applicable to a development server as described in Section 3.3.1, “Configuring JpGraph/PHP on a development server” the following changes should be considered in a production environment.

Setting the memory limits

The one thing to keep in mind here is that each active connection will spawn a unique PHP instance (HTTP process). This means that the memory limit set per PHP process can cause a very high memory demand on a busy server with many simultaneous connections. For this reason it is important that during system test (before going into production) the actual needed memory limit is determined.

For a busy server it is not uncommon to dimension it so it can handle 100 simultaneous connections. If the limit of each PHP process is set to 32MB this means that the server needs at least ~3.2GB memory just to handle the PHP processes (if they are all using their maximum allowed memory).

Setting maximum allowed run time

The same principle applies to the allowed run time. For a production server with high load and many simultaneous users it might be necessary to increase the maximum allowed execution time just to be sure no process is terminated due to it reaching its maximum allowed run time. When that happens the PHP process will be killed and no output sent back to the client (e.g. the browser).

Disabling output buffer

The output buffer should be disabled on the production server as well since enabling this will slow down the PHP and put a higher demand on the memory requirements.

Enabling adequate error checking

On a production server it is not a good idea to display all PHP error messages to the end user so the display of error messages should be disabled and the error messages should only be logged to a file.

Tip

On a production server it is also a good idea to have the following settings:

```
display_errors = Off
```

This makes sure that now PHP errors are displayed

```
display_startup_errors = Off
```

This makes sure that any initial errors thrown by PHP is not displayed to the end user

```
log_errors = On  
error_log = <name-of-log-file>
```

This makes sure all server PHP errors are logged to a specified file

Setting up your jpg-config.inc.php

On a production server it is best not to show detailed error messages to an end user. Instead it is better to have a generic error message that indicates a server problem and give an error code which can be decoded by looking it up in the table in Appendix H, *Error messages*. Using a generic error message is achieved by setting the following define:

```
define('DEFAULT_ERR_LOCALE', 'prod');
```

3.3.3. Adjusting PHP include path

As was mentioned before the library should be installed somewhere in the PHP include path. There are two ways of configuring the include path:

1. setting the include path in php.ini

```
include_path = <file-path>
```

2. adjusting the include path directly in the code by using the PHP command `php_ini_set()` at the top of the script

The library examples assume that the library is available under a directory called "jpgraph/". This will allow the scripts to include the library files by, for example, writing "include" or "require_once" statements such as

```
require_once( 'jpgraph/jpgraph.php' )
```

3.3.4. Using Apache2 alias configuration during development

Note

This section only discusses alias setting using the Apache HTTP server so this section can be skipped at first time reading the manual without loss of continuation.

Note

More detailed information on the alias directive is also available in the official Apache documentation at http://httpd.apache.org/docs/2.2/mod/mod_alias.html [http://httpd.apache.org/docs/2.2/mod/mod_alias.html#alias]

When accessing examples and test code through a regular browser during development the scripts must be available in document root (or somewhere beneath that root) the root is traditionally named /htdocs. Having a development code/repository directly under this root directory is not a good idea. For example, write access to a document root (even on a development server) should be restricted, in addition the paths given to a test team should be the same whatever version is currently under test so storing different versions with different names under the root is also a poor setup.

A much better and easy, approach is to use the powerful concept of alias in Apache. This is a way of mapping a URL to a specific directory on the server. For example if Eclipse-PDT [http://www.eclipse.org/projects/project-plan.php?projectid=tools.pdt] is used as an IDE to develop PHP it is mandatory to have a workspace setup where all the working files resides. Assuming that a local developer has his workspace directly in his home directory, say ~joe/workspace, we could configure an alias so that the workspace is accessible by the alias "http://localhost/ws/" by adding the following configuration in the Apache setup file(s)

```
Alias /ws /home/joe/workspace
<Directory /home/joe/workspace>
    Order allow,deny
    Allow from all
</Directory>
```

In this particular setup we use very liberal settings, allowing basically everyone with server access to access the directory. Using this approach makes it very easy to use the same test setup but allow testing of different branches/versions of the code.

Depending on the system these configurations can reside in different places. However, a very common structure is to keep all these small configuration files under /etc/apache/conf.d/. The main Apache configuration then reads all the files (regardless of there name) that are stored under this directory.

As a final example we show a further slightly more complex example (which actually shows how most of our developers have there systems setup for PHP5 development). This example adds options to do directory listing and allows the server to follow symbolic links. More information on available argument for the directive option is available in the official Apache documentation http://httpd.apache.org/docs/2.2/mod/core.html#options

Example 3.5. Alias configuration for a development server running Apache with Eclipse-PDT

```
# Configuration for Eclipse workspace
#
<IfModule mod_php5.c>
    Alias /ws/ /home/joe/workspace/
    <Directory /home/joe/workspace/>
        Options           +Indexes +Multiviews +FollowSymLinks
        order allow,deny
        allow from all
    </Directory>
</IfModule>
```

3.4. Installing and configuring Font support

JpGraph supports two types of fonts, bitmap and TTF fonts. The advantage with bitmap fonts are that they are supported from within the library without further configuration. TTF fonts require some configuration and possible installation of the actual TTF font specification file. (For more on installing TTF fonts see Section 3.4.1, “Configuring TTF fonts”.)

The drawbacks, on the other hand, with bitmap fonts are quite a few

- they look quite crude and are only available in a few sizes

- italic font style is not supported for bitmap fonts
- they only support 7-bit ASCII characters
- they only support 0 and 90 degree rotated text

You can read more about how to use and select among the different fonts in Chapter 8, *Text and font handling*

3.4.1. Configuring TTF fonts

Note

This could be considered an optional section since the library will work even without TTF fonts. However, for the reasons listed previously it is strongly recommended that TTF support is configured.

In the following we will assume that the FreeType library is enabled and verified in the PHP installation (See Figure 3.2, “`phpinfo()` GD-Information”)

Due to various legal issues no TTF fonts are included in the distribution of JpGraph since many commonly used TTF font files are copyrighted and their distribution restricted. For the most commonly used WEB-fonts (the Microsoft Core Fonts) the status is unclear. For many years Microsoft distributed them freely but they are no longer available from Microsoft's home page. Instead they are available from <http://corefonts.sourceforge.net/>

There are however a fair amount of freely available high quality TTF fonts (see below). The first thing needed is to make sure that the path defines in the file `jpg-config.inc.php` corresponds to the server setup (so that the library can find the font files)

1. Open `jpg-config.inc.php` for editing
2. Locate the define `TTF_DIR` (this is the define that possibly needs updating depending on your system). By default this path will have a sensible value depending on if the library is installed on a Windows or a Unix system.
3. If you are on a Windows platform you can just point the TTF directory path in JpGraph to use the standard Windows font directory (e.g `C:\windows\fonts\`)
4. If you are on a Unix platform, which may not have any TTF fonts installed, you can download and install the core MS WEB-initiative fonts from <http://corefonts.sourceforge.net/> many Linux distributions also have automatic ways to install these fonts. These fonts were put in the public domain by the Microsoft Corporation but they are no longer available directly from Microsoft.
5. You can also choose to install the freely available Vera Bitstream TTF fonts available from <http://www.gnome.org/fonts/>
6. You can also choose to install the freely available DejaVu TTF fonts [http://sourceforge.net/projects/dejavu/](http://sourceforge.net/projects/dejavu)

JpGraph uses a standard naming convention for the TTF font files in order to be able to find the correct font file that corresponds to a particular font family. This naming convention follows the standard naming of the available font files from the distributions listed above.

If the installation of the library is made on a computer running MS Windows then it is recommended to use the already available font files in Windows (usually located in `C:\WINDOWS\FONTS`).

If the installation is made on a UNIX derivate running X11 then the font location can differ between versions and UNIX brands. One commonly used path in modern installations are "/usr/share/fonts/truetype/" (In older installations it was common to put the truetype fonts under "/usr/X11R6/lib/X11/fonts/truetype/".)

Finally we note that it is possible to install additional fonts not natively supported by the library by using the `SetUserFont()` family of methods. (In theory it is also possible to patch the `jpgraph` source files to include support for other font files natively but since this requires code modifications of the library we do not discuss this further here in the introduction.)

Freely available TTF fonts can be found, for example, from

- <http://www.webfontlist.com>
- <http://www.webpagepublicity.com/free-fonts.html>
- <http://www.fontonic.com/fonts.asp?width=do&offset=120>
- <http://www.fontspace.com/category/famous>

3.4.2. Using non-latin based fonts with JpGraph

In addition to European fonts it is also possible to use non-latin based fonts such as Cyrillic, Japanese, Chinese, Hebrew and Greek. For any of these languages a suitable TTF font that supports the non-latin based language must be made available to the library. Some specific rules also applies to each of the supported languages due to the necessary character encoding needed due to different convention when writing the character for some languages.

- For Cyrillic support the define `LANGUAGE_CYRILLIC` in `jpg-config.php` must be set to `true`. It is then possible to use a suitable Cyrillic font as replacement of the ordinary font. This setting combined with the `CYRILLIC_FROM_WINDOWS` and `LANGUAGE_CHARSET` is used to fine tune the handling of Cyrillic input. The rules are as follows:

1. If `LANGUAGE_CYRILLIC` is **false** no specific handling of Cyrillic characters at all will be done.
2. If `CYRILLIC_FROM_WINDOWS` is **true** then it will be assumed the input coding by default is encoded using **WINDOWS-1251**.

The conversion is then done via a call to `convert_cyr_string($aTxt, 'w', 'k')` where `$aTxt` is replaced with the input string to be encoded.

3. If `CYRILLIC_FROM_WINDOWS` is **false** then it will be assumed the input coding by default is encoded using **KOI8-R**.

The conversion is then done via the sequence of calls

```
<?php  
$isostring = convert_cyr_string($aTxt, "k", "i");  
$unistring = LanguageConv::iso2uni($isostring);  
?>
```

in order to get a proper utf-8 internal encoding (internally the library only uses utf-8 encoding)

4. `LANGUAGE_CHARSET` can be used to dynamically adjust the conversion of the input character set when using Cyrillic characters. This constant is used to auto-detect whether Cyrillic conversion is really necessary if enabled (via the If `LANGUAGE_CYRILLIC=true`). This constant can be set to a variable containing the currently used character input set. A typical such string would be 'UTF-8' or

'utf-8' (the comparison is case-insensitive). If this charset is not 'koi8-r' nor 'windows-1251' derivate then no conversion is done.

- For Chinese character set JpGraph supports both BIG5 and GB2312 encoding. For BIG5 encoding the PHP installation must have support for the "iconv()" function. Furthermore the define CHINESE_TTF_FONT must be set to the name of the Chinese BIG5 font that is to be used. By default this is set to "bkai00mp.ttf". To use the Chinese BIG5 font in the scripts one must then specify the font family as FF_CHINESE.

To use the alternative font files "simsun.ttc" and "simhei.ttf" (which uses the GB2312 encoding) the only step needed is to install those fonts in the normal TTF font directory and then specify the font family as FF_SIMSUN, the "simhei.ttf" is used when the font style is specified as FS_BOLD.

3.5. Adapting and customizing the installation

All configuration of the library is done in the file `jpg-config.inc.php`. Each option in the file is extensively documented in Appendix L, *The JpGraph configuration file*. Here we will only cover the most important configuration that it likely to have to be customized in order to successfully run the library.

Note

All configuration settings comes with default values that should in normal cases be enough for a "standard" system. However for a production system you might want to fine tune this according to your specific system setup.

Tip

During development (and to some extent even on a production server) it might be necessary to adjust some parameters in your `php.ini` file. For more on this see Section 3.3, "Installing the library"

3.5.1. Setting up necessary paths

There are three categories of directory paths that can be specified. The categories are:

1. Font directories, this we already touched upon in Section 3.4.1, "Configuring TTF fonts".
2. Cache directory, the cache feature of the library is extensively discussed in Chapter 9, *Using the JpGraph cache system*. The important thing to notice here is that whatever directory you chose to use as a cache directory it must be a directory that is writable for the process running PHP (normally the HTTP server process, e.g. Apache)
3. CSIM Cache directory (if CSIM cache feature is used). CSIM stands for Client Side Image Maps and is a way to construct client side drill down charts. (Today this is the only inpractice used image mapping method but it is also possible to have Server Side Image Maps).

3.6. Verifying the library installation

After installing the library a good check that everything works is to run the included example scripts. Worth noting is that some of the examples uses TTF fonts. This means that if you try to run those examples on an installation that does not have any TTF fonts configured an error message will be displayed.

Running your first example

If we assume that you have installed the library directly under the document root on the HTTP server in a directory named "jpgraph/" you can access the examples in the "jpgraph/Examples/" directory by pointing your browser for example to "jpgraph/Examples/example0.php". This should then show the same image as is displayed in Figure 1.2, "This is the very first example (example0.php)"

If the above example works correctly you can run all the included examples by pointing the browser to the script "jpgraph/Examples/testsuit.php". This will automatically generate all non-CSIM examples (for an explanation of CSIM graphs see ??) and show them in one long list. Note that since this has more than 300 images it can take some time to generate all example images.

Tip

In order to generate all CSIM examples you can run the testsuit with the URL argument ?type=2

```
jpgraph/Examples/testsuit.php?type=2
```

You will then get a number of examples of so called drill-down charts where a number of areas of the graphs are hotspots which can be clicked to open a specified URL. However, in these examples all the click-URL have just dummy values which points back to the same file.

3.7. Troubleshooting the installation

Unfortunately there are many parameters in a server installation that may affect the execution of PHP so the steps below can only give some indications of how to further investigate some potential problems.

Experience shows that most of the trouble are caused by either an old buggy version of the free-type TTF library or using an old antiquated version of the GD library. So before starting trouble shooting the scripts please make sure that you have an up to date PHP installation with the bundled version of the GD library (as described in the previous sections) and a working FreeType library installed.

1. No image is displayed.

The first thing you should do is to isolate the problem by calling your graph script directly in the browser. There are then two variants of this problem.

- a. No data is sent back from the server.

You can verify this by calling your graph script directly in the browser and then check the source (click "view source" menu item in the browser). If this is a truly blank image then no data was sent back from the browser.

This means that the PHP process has been terminated prematurely before it could send any data back. This could be caused by either the PHP process crashing (due to a bug in either PHP or your HTTP server) or the HTTP server crashed. This is often due to a broken PHP installation and more than often a problem with the True Type libraries.

It could also be caused by the PHP script running longer than the maximum allowed execution time (as specified in `php.ini`). The first thing you should do is to increase the maximum allowed execution time. If this does not solve the problem you should look in the log files for both your HTTP server and PHP to try to find clues to if the PHP process really crashed. Another possibility is that the PHP process uses more than the maximum allowed memory (as set in `php.ini`) and then it is terminated. So a good first step is to put some really high values for memory and time just to take away these parameters.

- b. The data sent back is corrupt.

Depending on your browser this can show up differently but a common symptom is a "red X" in the browser. In order to debug this you should make sure that you have followed the steps in Section 3.3.1, "Configuring JpGraph/PHP on a development server" to make sure you have output buffering disabled and have maximum error checking enabled in PHP. The most common cases for this type of problem is having enabled output buffering and some minor errors in the script which causes PHP to emit warnings which gets included in the image data.

A very common mistake is to have some white spaces in the script before the opening "<?php". This white space will be added to the output buffer and then get mixed up with the image data causing the image data to be corrupt. A similar problem can occur if multiple newlines are added after the final "?>"

2. An error message saying "Fonts are not available or not readable"

When an image contains TTF fonts you might get an error message saying that the fonts are not available or not readable. If this is the case it is first necessary to check that the font files really exist in the directory that is specified in `jpg-config.inc.php` and that they are also readable by the HTTP/PHP process. If this is the case then it is necessary to check that the names of the font files are the one that JpGraph assumes, see the section called "Name of TTF font files". Another problem can be if the PHP installation is running in "safe mode" (See [PHP Manual: Security and safe mode](#) [`http://se.php.net/manual/en/ini.sect.safe-mode.php`]) and has enabled strict directory policy via an "open_basedir" restriction. This will prevent the PHP process from reading any files outside the specified base directory. If this is enabled there is no way around for PHP to read any files outside this restriction and any TTF files necessary must be moved so that they can be accessed within the realms of the specified basedirectory.

If you are running IIS and Win2k and get the error "Can't find font" when trying to use TTF fonts then try to change the paths to UNIX style, i.e. `"/usr/local/fonts/ttf/"`. Remember that the path is absolute and not relative to the `htdocs` catalogue.

If you are running on Unix server please keep in mind that file names are case sensitive.

3. An error message saying "Headers have already been sent"

A common mistake is to have a space in the beginning of the image script which the HTTP server will send back to the browser. The browser now assumes that the data coming back from this script is text since it hasn't received an explicit header. When then the image headers get sent back to the browser to forewarn the browser of the forthcoming image data the browser will not like that as it has already assumed the data stream was a text stream. The browser will then give the infamous "Headers already sent error".

Make sure that your script has no white space before the opening "<?php" statement or a number of blank lines after the concluding "?>"

4. Issues specific to Windows and IIS

Some windows installations seems to have a problem with a PHP script ending in more than one newline (This newline seems to be sent to the browser and will cause a *Header already sent error*). To correct this problem check all your scripts for more than one empty newline after the ending "?>" statement. All files provided with the library end in exactly one final newline and should not be a problem.

5. TTF fonts are not displayed correctly

If the TTF fonts only shows up as yellow then the installation is used a buggy (too old) installation of the FreeType font library and the only thing to do is to re-install and setup PHP+GD again with a newer

version of the FreeType library. Another symptom of a an (old) buggy FreeType library is that the fonts are not correctly rotated (the text string is rotated but not the individual characters).

As a final advise you should read the FAQ (available in this manual see Appendix C, *FAQ*) or on the JpGraph website at <http://www.aditus.nu/jpgraph/jpgraphfaq.php>

Depending on your sever it might also help to recompile PHP yourself instead of the version included with the system. You can find typical configuration scripts to compile PHP4 and PHP5 in the appendices, see Appendix I, *Compiling PHP*

Part II. Basic graph creation

The goal of this part is to introduce some basic concepts that must be known when creating graphs with the JpGraph library. It will also serve as a gentle introduction on how to structure a basic graph script.

The challenge of writing a good manual is that it is sometimes necessary to use concepts that might not have been fully explained in order to clarify some even more basic concept. So if there are parts that you do not fully understand do not despair. The remainder of the manual will explain a lot more of the details.

Chapter 4. Your first graph script

What you will learn in this chapter. This chapter will illustrate how a basic line and bar graph can be created and it will also show how input data should be prepared so it can be read and used by the library.

4.1. Some words of caution

In order to quickly show a very first example we will create both a basic line graph and a basic bar graph that depicts the number of sun spots (a.k.a. solar flares) during the 19th century. The goal of this example is not to show every possible configuration and parameter supported by the library but rather show how simple it is to create a basic graph.

In the sections following this one we will describe more in details the idiosyncrasies about dynamic graph generation and JpGraph so even if you don't fully understand all the detail it will give a flavor of what is to come. As with all complex libraries one has to start somewhere and sometimes accept some practices without yet fully understand them. However, the goal of this manual is that after reading it through you will fully understand every single detail shown in this script.

So, without further due, let's start.

4.2. Graphing the number of sun spots during the 19th Century

4.2.1. The historic data

It is a well known fact that sun spots have a certain pattern and regularity. The cause of these regular patterns are not currently fully understood (even though investigation into this phenomenon has been made since beginning of the 17:th century). The fact that solar storms affects the earth in terms of interference with radio traffic and other sensitive electronic devices makes it very interesting to keep careful records of the suns activities.

For this reason the data of solar storm is readily available and makes an interesting first example. The data used here is taken from SIDC (The Solar Influences Data Analysis Center) in Belgium (<http://sidc.oma.be/sunspot-data/SIDCpub.php>). In this example we will use the summary historical data that shows the total number of sun spots per year since 1700..

4.2.2. Preparing the data

The first step is to get the data into our PHP script which makes for a first good discussion since all graphs needs to get data from some source. The library itself is agnostic in regards to from where the data is collected and only needs (and requires) data stored in a PHP array of numbers (integers or floats).

In principle the data to be plotted in the graph can come from :

1. Hard-coded data in the script. This is the least flexible and can only really be recommended for examples and really static data.
2. Data stored in plain text files. (This is what we will use in this example.)
3. Data stored in binary format in flat files.
4. Data stored in a database

5. Data sent to the script via URI parameter passing (either GET or POST HTTP constructs can be used).

What is common among all these methods is that the creator of the script has to read the data into one (or several) data arrays that can be used by the library. For our example the data of sunspots are stored in a plain text file in two columns, one column for the year (with a ".5" added which indicates the average of the year) and one column for the number of sunspots for the corresponding year. As illustration the first 10 lines of data is shown in Figure 4.1, "The first ten rows of data of sunspot activities from year 1700".

Figure 4.1. The first ten rows of data of sunspot activities from year 1700

```
1700.5 5.0
1701.5 11.0
1702.5 16.0
1703.5 23.0
1704.5 36.0
1705.5 58.0
1706.5 29.0
1707.5 20.0
1708.5 10.0
1709.5 8.0
```

From this data we need to create two arrays, one with the number of sunspots and one with the corresponding years. If we assume that the data is stored in a text file named "yearssn.txt" in the same directory as the script file the function in Figure 4.2, "Reading numeric tabulated sunspot data from a file" will read the data into two arrays

Figure 4.2. Reading numeric tabulated sunspot data from a file

```
<?php
function readsunspotdata($aFile, &$aYears, &$aSunspots) {
    $lines = @file($aFile,FILE_IGNORE_NEW_LINES|FILE_SKIP_EMPTY_LINES);
    if( $lines === false ) {
        throw new JpGraphException('Can not read sunspot data file.');
    }
    foreach( $lines as $line => $datarow ) {
        $split = preg_split('/[\s]+/', $datarow);
        $aYears[] = substr(trim($split[0]),0,4);
        $aSunspots[] = trim($split[1]);
    }
}

$year = array();
$ydata = array();
readsunspotdata('yearssn.txt', $year, $ydata);

?>
```

In the function above we have deviated from the common practice of not including even the most basic error handling in examples by adding an exception in case the data file could not be read. This is to emphasize that graph scripts which reads data from potentially disconnected sources must have real quality error and exception handling. As this is the first example we will not discuss the details of the error handling other than saying that the library provides one exception class `JpGraphException` that is meant to be used by clients to signal unrecoverable errors in the code. The full details on error handling in the library is discussed in Chapter 6, *Error handling*

Tip

In the library there is an auxiliary utility class `ReadFileData` to help read data from text files. In this class there are methods to read data from a file in either of the following formats

- CSV (Comma Separated Values) format. `ReadFileData::FromCSV()`
- two column format (almost) as we did manually above with `ReadFileData::From2Col()`
- one column format `ReadFileData::From1Col()`

Armed with the data in the two arrays `$year` and `$ydata` we will now plot the data in a basic line graph with some variations and then show the data in a bar graph.

4.2.3. A basic line graph

As the very first start we will create a line graph which shows sun spots as a line graph. To keep the code focused on the graph we do not include the previous function to read the data again in the code snippet shown below. Before we get into the code we start by briefly discuss how your script can include the necessary library files.

All graph scripts must include at least two files, `jpgraph.php` and some plot module. If we want to create a line plot we must include `jpgraph_line.php`. Slightly depending on the server setup and what paths are defined for PHP include files (as discussed in Section 3.3.3, “Adjusting PHP include path”) the include paths for the library might look a bit different. However, we recommend that you install the library files so they can be accessed, for example using, `require_once('jpgraph/jpgraph.php')` (since this is what is assumed by the library examples). Furthermore, we would recommend that the `require_once()` construct is used to avoid including the same file multiple times.

```
<?php
    // Width and height of the graph
$width = 600; $height = 200;

    // Create a graph instance
$graph = new Graph($width,$height);

    // Specify what scale we want to use,
    // int = integer scale for the X-axis
    // int = integer scale for the Y-axis
$graph->SetScale('intint');

    // Setup a title for the graph
$graph->title->Set('Sunspot example');

    // Setup titles and X-axis labels
$graph->xaxis->title->Set('(year from 1701)');

    // Setup Y-axis title
$graph->yaxis->title->Set('(# sunspots)');

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

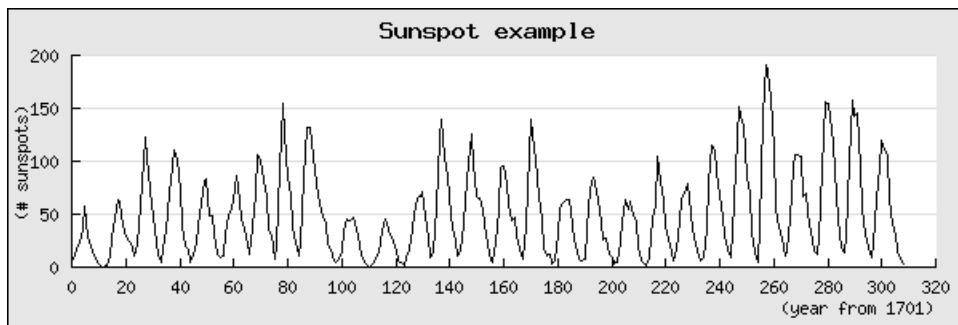
    // Add the plot to the graph
```

```
$graph->Add($lineplot);

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

Before we explain this code in some more detail it is a good idea to visualize what we get when we execute this as script. The result of running this script is shown in Figure 4.3, “Line plot showing the number of sun spots since 1700 (`sunspotsex1.php`)”

Figure 4.3. Line plot showing the number of sun spots since 1700 (`sunspotsex1.php`) [[example_src/sunspotsex1.html](#)]



Tip

You can always click on the filename in the title of a figure to view the complete source code.

Let us now walk through this code in some details.

Line 1-12 The size of the graph must always be specified so the first thing to do is to create a new graph object and set the width and height of the overall graph. All graph scripts will need to create at least one instance of the `Graph()` class. By convention in all our scripts we will name the created instance of the `Graph` class "`$graph`"..

The second thing that all graph scripts must specify is what kind of scales should be used. The library supports linear, integer, logarithmic, text and date scales. Since we know that our data consist of only integers we keep things simple and set both the X and the Y axis scale to be integers. The scale is specified as a string where the first half of the string denominates the X-axis scale and the second half denominates the Y-axis scale. So in our example we specify '`'int int'`'. With this explanation you can probably guess what '`'int log'`' or '`'lin log'`' would do. Why not try it ?

Line 13-21 These lines sets some different text labels. By the naming convention used in the library you can probably guess what all those lines are doing. They set the overall graph title as well as the X- and Y-axis titles. To keep the example as lean as possible we use the default font and default size and color of the text strings.

Line 22-27 Each graph must have at least one plot (data series) that is added to the graph. In our case we wanted to create a line graph so we must create an instance of the class `'LinePlot'`. We create a new instance of this class and as a parameter use the data for the data series we want to create the line plot from, in our case the data array with the sun spot numbers.

Line 29 Understanding this single line is key to understanding dynamic graph generation with PHP. This line instructs the library to actually create the graph as an image, encode it in

the chosen image format (e.g. png, jpg, gif, etc) and stream it back to the browser with the correct header identifying the data stream that the client receives as a valid image stream. When the client (most often a browser) calls the PHP script it will return data that will make the browser think it is receiving an image and not, as you might have done up to now from PHP scripts, text.

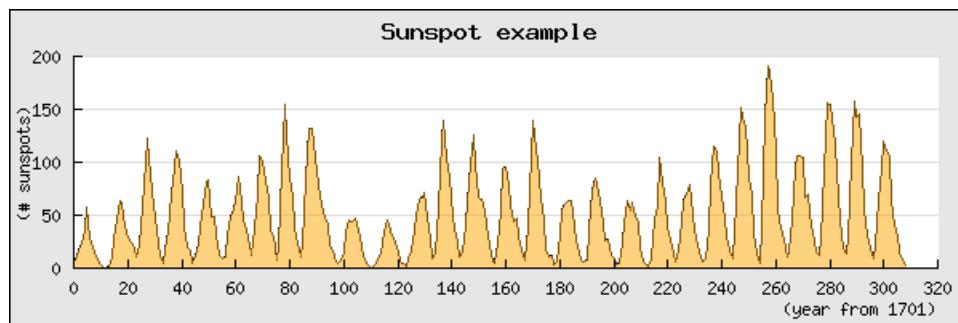
This is something that can be conceptually difficult to fully understand at first and that is why we are spending the entire next chapter Chapter 5, *Fundamentals of dynamic graph generation* on further exploring this. But for now please accept that this works and by calling the script above in your browser (it is available in the "Examples/" directory in the distribution as 'sunspotsex1.php') you should get the exact same image as shown above in Figure 4.3, "Line plot showing the number of sun spots since 1700 (sunspotsex1.php)"

Let's now make a small variation of the above line graph. Let's make it a filled line graph instead. In order to do this we only have to add one single line

```
<?php  
$lineplot->SetFillColor( 'orange@0.5' );  
?>
```

The line above actually does two things. First it sets the basic color to 'orange' and then it modifies this color to be 50% opaque (0.5) which makes the grid line shine through the fill color to some extent. The whole color handling in the library is further described both in Appendix D, *Named color list* as well as in Chapter 7, *Color handling*. The result of adding the line above is shown in Figure 4.4, "Displaying sun spots with a semi filled line graph (sunspotsex2.php)"

Figure 4.4. Displaying sun spots with a semi filled line graph (sunspotsex2.php) [example_src/sunspotsex2.html]



Adding tick labels to the X-axis

Note

This section can be skipped at first reading since it contains some slightly more advanced material. The reason why we have included this section already now is that some of the issues discussed here is an often repeated question among newcomers to the library.

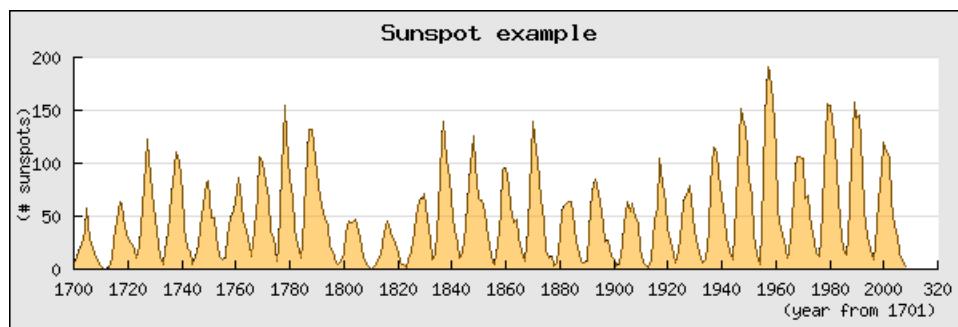
There is one bit of the available data that we haven't used yet and that is the actual years. In the example above we can only see the count from 1700. (If we just want to look at the cyclic behaviour of the number of sunspots this is fine since what year a specific number of sunspots appeared is not relevant.) To make it easier to see what year corresponds to the different sunspot numbers we must change the label on the X-axis scale to show the years instead.

There is actually a couple of ways to do this. The easiest way is to just add the labels we have (`$years`) on the X-axis and instruct the library to use them instead. This is done with a call to the method `SetTickLabels()` on the X-axis. This method call takes an array as argument and uses the values in that array to populate all labels on major tick marks. In order to make adequate room for the scale the library automatically selects a suitable distance between each major tick mark.

```
<?php  
$graph->xaxis->SetTickLabels($year);  
?>
```

Adding this line to our previous graph will generate the graph shown in Figure 4.5, "Adding tick labels to the graph (sunspotsex3.php)"

Figure 4.5. Adding tick labels to the graph (sunspotsex3.php) [example_src/sunspotsex3.html]



However, there is a problem in the graph above. There are valid years on the X-axis up to "2000" but then there is a single label "320".

What is going on here? Have we already discovered a bug in the library?

No, not really. In the way we have setup the graph we have not provided the library with enough labels. What has happened is that the integer scale has chosen a suitable interval between each tick label to have enough space to be able to show the labels. As can be seen from the figure the distance chosen with this particular graph seems to be 20 years between each tick. The way the default labeling works is that the end tick should be labelled and hence be an even multiple of 20 years (in this case).

Since the library needs to have tick labels for all ticks it uses the labels we supplied as far as they go (up to 2008) but since we didn't supply data more than up to "2008" (in the `$year` array) the library does what it can do and continues with the ordinal numbers where we failed to provide enough labels.

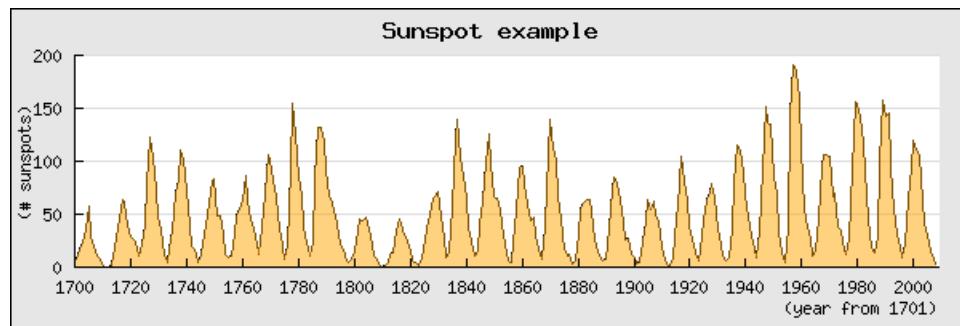
Now, there are some standard ways of correcting this abomination.

1. We can set a manual scale to make sure the scale ends exactly at 2008, i.e. the scale is exactly as long as our data. This is done by submitting the wanted scale min/max as additional argument in the `SetScale()` method. First the min/max for the Y-axis and then the min/max for the X-scale. Since we still want the Y-scale to be fully automatically determined we just put a "0" for both min and max on the Y-scale and specify 0 for the min x-value and then the exact number of sunspots we have measured as the maximum x-value.

```
<?php  
$graph->SetScale('intint',0,0,0,max($year)-min($year)+1);  
?>
```

The result of this is shown in Figure 4.6, “Manually specifying the X scale to use just the supplied X values (sunspotsex4.php)”

Figure 4.6. Manually specifying the X scale to use just the supplied X values (sunspotsex4.php) [example_src/sunspotsex4.html]



2. An alternative way to get labels is to use a callback function to specify the labels. This works so that the library calls the user specified function and as argument passes the label (or the value of the label) that the library intends to put on a tick. The library will then use as the actual label whatever string value we return from our function. Since we know that the integer label 0 (the first tick) corresponds to the first value, i.e. "1700" we can simply take whatever label we get as argument, add "1700" and return that value. This way all labels will be properly named and even if the scale extends far beyond where we have data a sensible tick label will be shown.

A suitable callback function together with the method to instruct the library to use this callback would be

```
<?php
// ...

// Label callback
function year_callback($aLabel) {
    return 1700+(int)$aLabel;
}

// ...

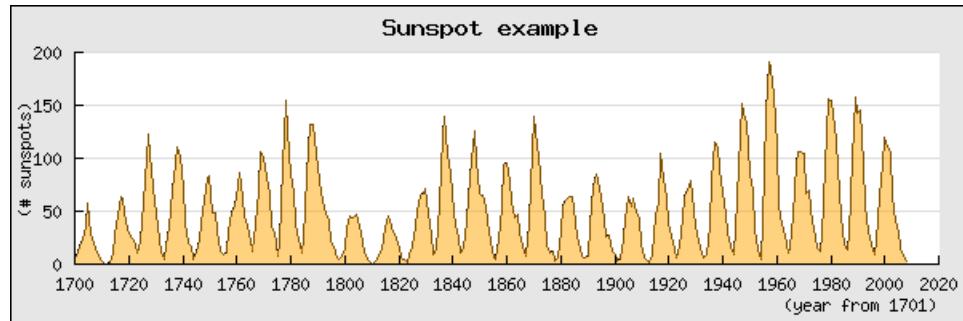
$graph->xaxis->SetLabelFormatCallback( 'year_callback' );

// ...

?>
```

and the result can be seen in Figure 4.7, “Using a callback to get correct values on the x axis (sunspotsex5.php)”

Figure 4.7. Using a callback to get correct values on the x axis (sunspotsex5.php) [example_src/sunspotsex5.html]



3. There is one more way to handle this issue which we will not cover in detail yet. This is to use a "text" scale. The "text" scale can be used when there is no need to show numeric values on the axis. A typical use for text scale would be to add labels to mark bar graphs. Of course a text can contain numeric strings that would make it visually indistinguishable from a "real" numeric value.

For text scales every label counts. So by default the library will assign a tick mark for each ordinal so that every label is used. In some cases this will just be two dense and then the tick and the labelling can be adjusted by calling the two methods `Axis::SetTextTickInterval()` and `Axis::SetTextLabelInterval()` to get to a wanted result. But for now we do not discuss this technique further here since it would bring too far.

4.2.4. A basic bar graph

As a final illustration we will show how easy it is to change the plot type. We will make a small modification of the previous script and display the sun spots as a bar graph instead. In order to do this we take the code from Figure 4.4, “Displaying sun spots with a semi filled line graph (sunspotsex2.php)” and just change the creation of an instance of the `LinePlot()` class to instead be an instance of the `BarPlot()` class (to make the code more readable we also change the name of the variable where we store the instance so it makes more sense). In order to use this class we must also change the include statement so that the bar module is included by adding the statement `require_once('jpgraph/jpgraph_barplot.php')`. The changed code would now look like this

```
<?php
// ...

// Create the bar plot
$barplot=new BarPlot($ydata);

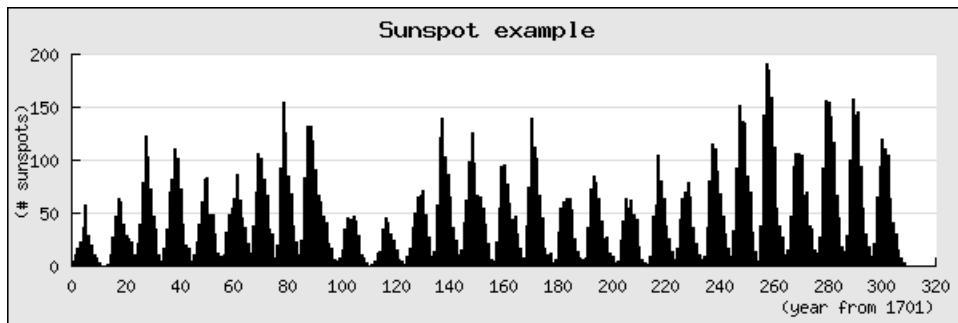
// Add the plot to the graph
$graph->Add($barplot);

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

?>
```

and would result in the graph displayed in Figure 4.8, “Changing the plot type to a bar plot instead (sunspotsex6.php)”

Figure 4.8. Changing the plot type to a bar plot instead (sunspotsex6.php) [example_src/sunspotsex6.html]



Since there are so many bars in small space we cannot see the individual bars in the example in Figure 4.8, "Changing the plot type to a bar plot instead (sunspotsex6.php)". So lets modify the script so that it only shows the last 20 years of measurements so that we can see the individual bars. To set this up there are two things we must do

1. Change the scale of a text scale since we want to make sure each value is displayed. In addition the text scale actually changes one more thing that we haven't mentioned. Using a text scale also changes the alignment of the labels. For linear, integer, logarithmic scales the labels are placed centered below the tick marks. For text labels they are instead placed in between the tick marks. This is the most common way of displaying bar graphs. This is one more reason to think of text scales as a special scale mostly suitable for bar graphs.
2. Add two lines of code to "chop off" the extra not wanted data in the input data arrays.

The following code snippet shows the necessary modifications to the previous script

```
<?php
// Just keep the last 20 values in the arrays
$year = array_slice($year, -20);
$ydata = array_slice($ydata, -20);

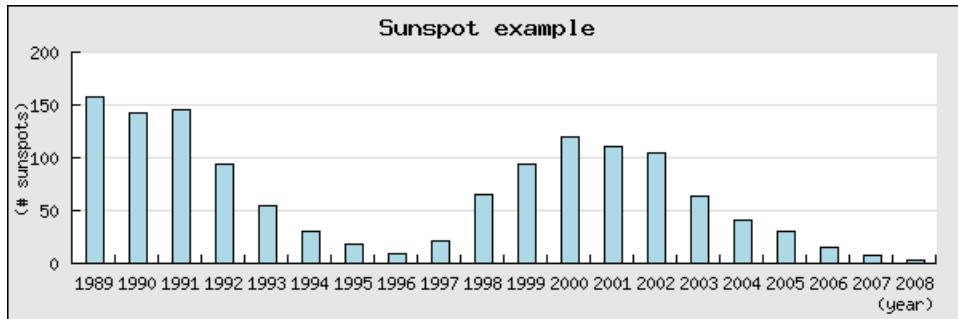
// ...

// Specify what scale we want to use,
// text = txt scale for the X-axis
// int = integer scale for the Y-axis
$graph->SetScale('textint');

// ...
?>
```

The final graph with the "zoomed" last 20 years can now be seen in

**Figure 4.9. Sunspots zoomed to only show the last 20 years (`sunspotsex7.php`)
[[example_src/sunspotsex7.html](#)]**



Since the scale is larger we can now actually see the individual bars. By default the library chooses a light blue color to fill the interior. (Try to see what happens if you add the method call "`$barplot->SetFillColor('orange@0.5');`" just after the "`$barplot`" variable has been assigned the new BarPlot object.)

Chapter 5. Fundamentals of dynamic graph generation

What you will learn in this chapter. The purpose of this chapter is to put dynamic image generation in perspective and illustrate how HTML tags are used to call image generating scripts. Even if You are familiar with PHP it is strongly recommended to quickly browse through this chapter to make sure all concepts are known. If You fully understand and can explain the concept of MIME types, HTTP streams and why the "*Headers already sent error*" error is a very common error when generating dynamic images with PHP it is probably safe to skip this chapter. Otherwise it might be wise to read through this chapter at least once.

5.1. Making sense of HTTP streams and MIME types

The following explanation is slightly simplified since a full description of the HTTP protocol would bring us a bit too far in this manual

1. A client (e.g. browser) requests data from the server by issuing a GET (or possibly a POST) command to the server. This is what happens when you enter a URI in the address bar in the browser.
2. The server replies with a data stream (or an error if the requested data wasn't available). This data stream is prepended with header (MIME header) that tells the client (e.g. the browser) how to interpret the data that follows. The most common type (and the default type if no header is sent by a faulty server) is "**text/html**". This tells the client to interpret the data as plain text with embedded HTML encoding.

When the data is to be interpreted as an image the header will instead be one of the image headers, for example "**image/png**" or "**image/jpeg**". When the client receives this header it will interpret all the following data as an image encoded in the indicated format.

The important thing to keep in mind here is that each server reply (initiated by a call from the client) can only have one and only one MIME type. This is the key to further understanding the specific issues with dynamic image generation. This explains why if a PHP script running on the server sends a header first indicating that the following data it sends should be interpreted by the client as an image it cannot also include text since only one header can be used for one HTTP stream.

What happens on a WEB-page with, for example, multiple `` tags is that the browser issues a separate GET command for each of these images. So even though it can look like a fetching a single WEB-page can have different content each individual request to the server only have one single MIME type.

5.2. What is an image?

This might be a strange question and you might already know the answer but having a true understanding will help. An image is simply a data stream of octets (bytes) specifying the exact color of a number of pixels. A typical image uses 32 bits to specify the color which means it can encode 2^{32} different colors (Homework: How many gray levels are possible with 32 bits?). It would be perfectly possible to send this data stream as it is and indeed this is how the bitmap (BMP) format specifies the data. However, image data usually have a lot of redundancies that can be used to reduce the size of the data that needs to be transmitted to specify any given image. These compression formats are what is more commonly known as for example PNG, JPEG or GIF type of images. So it is actually more correct, in our view, to look upon these formats as different compression techniques rather than as true image formats.

Contrary to popular belief it is not the suffix on a file per se that identifies a particular file as an image on a HTTP server but rather the mapping on the server that instructs the server to send back a particular header for a particular file type (as identified by the file suffix), the MIME type. This might not seem like an important distinction to make but keeping this in mind will help understanding how images are created with a PHP script.

A normal PHP script sends back character data (or more likely HTML encoded character data) that is displayed by the browser. However it is nothing that stops the PHP script from sending back data that instead should be interpreted as an image assuming of course we know how to create such data..

How the client (i.e. the browser) should interpret the data sent is all controlled by the header that is sent before the data. So this now leads us to the idea that if our PHP script by some means constructs a valid image, compressed in a known format (say PNG), we can tell the client to interpret the data we send back as an image by just making sure we first send a header indicating that the data is an image.

This could now lead us to the perhaps surprising but crucial insight that if we call our script that sends back valid image data "myimage.php" and open it in a browser, for example by opening the local URI "`http://localhost/myimage.php`" for the browser this is no different than opening an image directly (for example "`http://localhost/myimage.png`")

The only difference is that in the first case (our PHP script) we create the header and data ourselves and in the second case the HTTP server looks up the file suffix (*.png) to find out what kind of header to send before reading the file and just passing on the data to the client.

Armed with this new knowledge we realize that our PHP script *is* an image for all practical purposes and intent from a client's point of view. So we could safely name our PHP script as a target for an `` tag. This gives us the final and most common way to call a graph script in a HTML page.

```

```

in exactly the same way as we would do with the old type of images. The `".php"` suffix is no problem since the client (e.g. the browser) doesn't care what the file ending is. It only cares about what header the server returns. If that header tells the browser that the data received should be interpreted as a PNG image it will do so regardless of the name of the source file on the server.

5.3. Static vs dynamic images

Having understood how we can generate images using a PHP script a new idea might be forming. There is nothing that says that we have to create the same image using the same data every time the graph script is called. We could for example randomly select a pre-stored image in a directory and every time we call (the same) graph script a new image is returned. Hence, with PHP scripts we can create dynamic images that generate different images depending on some yet to be specified parameters even though the script name stays the same. This is a new behaviour compared with our normal images.

This could also lead to a new, and very real, problem which has to do with the local caching most browsers do in order to avoid re-fetching already fetched objects such as images. Normally a browser will check the time stamp on the file on the server and compare with the latest fetched version it has stored locally. If the browser sees that the file on the server is older than what it already has it will not re-fetch the file.

Since a dynamic image is produced by a script and the script will stay the same even if the produced image is changing (due to new data) the browser might still think that the image has not changed since the script is the same and will not bother re-fetching it again. The exact behavior of the browser cache is usually adjustable by the end user and hence out of control for us who writes the script. A simple way to always force the client to re-fetch the dynamic image is described in Section 5.5.6, "Forcing the browser to update your graph"

5.4. Dynamic images on the command line

It is also possible to generate images directly using the command line version of PHP. This works the same way as the normal "through the browser" generation with the exception that no HTTP headers will be generated. Only the binary image data.

This could be an effective way to make an efficient web-site by having some automatic creation of images at regular interval (perhaps using some kind of cron-job) that are referenced as usual (with an `` tag) in the scripts. This will avoid having to re-generate the image every time a visitor hits the site.

Please make sure that you run the command line version of PHP (cli). Using the CGI SAPI version of PHP will not work since then the HTTP headers will be generated.

Note

If the CGI version is used the generation of headers may be suppressed by adding the '-q' option.

You can easily check the version installed by the command line

```
$ /> php --version
```

this should give a reply with something like

```
PHP 4.3.8 (cli) (built: Aug 29 2004 18:48:13) Copyright (c) 1997-2004 The PHP Group  
Copyright (c) 1998-2004 Zend Technologies
```

The important thing here is the (cli) marker. The JpGraph library check from what SAPI API it is invoked from and adjusts the header generation accordingly.

If all the above requirements are met then images can be generated directly on the command line and stored in a suitable file. For example by

```
$ /> php myimage.php > image.png
```

Please note that the file extension on the image file should match the format in which the image is generated.

5.5. How to generate images with JpGraph library

The two common steps for creating and using a Graph on your Web-page are

1. Create a script that constructs the graph, by getting the data and specifying how the graph should look, the size, what colors to use, what fonts to use and specifying other augmentations on the graph.
2. On the HTML page where the graph(s) should be displayed include add one or more `` tags which links to the PHP graphs script. Of course it is perfectly possible to call the image script directly in the browser to just display the generated image in the browser. This way it is possible to include any number of graphs on the Web-page.

Tip

One further thing to keep in mind is that it is also possible to pass arguments to the image script via the normal HTTP GET/POST arguments.

For example

```

```

This could be used to control the appearance of the image or perhaps send data to the image which will be displayed. Note that this is probably not the best way to send large amount of data to plot. Instead the only practical way, for large data sizes, is to get all the data in the image script directly, perhaps from a DB. Another alternative for large amount of data to be sent to the image script is by creating a POST request to the image script. This is further discussed in ?? (Getting hold of the data to be displayed)

5.5.1. The standard steps of setting up a graph

When it comes to the structure of your imaging script they will generally have the following structure

```
<?php
// ... Include necessary headers
require_once 'jpgraph.php';
require_once '....';

// Create the graph instance
$graph = new Graph($width,$height, ...);

// Specify what scale should be used in the graph
$graph->SetScale('...');

// ... code to construct the graph details and plot objects

// Add one or many plot objects to the graph
$graph->Add(...);

// ... and send back the graph to the client
$graph->Stroke();
?>
```

JpGraph is completely Object oriented so all calls will be action on specific instances of classes. One of the fundamental classes is the `Graph()` class which represents the entire graph.

After the creation of the `Graph()` object all the code lines to construct the details of the graph are added. The final method called in an image script will most likely be the `Graph::Stroke()` method. This will send the constructed image back to the browser. A variation of this is used if the graph are supposed to have image maps (CSIM). In that case the final method will be `Graph::StrokeCSIM()`

Caution

As discussed in Section 5.1, “Making sense of HTTP streams and MIME types” no text can be returned from an image script. Beware!

In addition to this standard usage pattern you can also choose to:

1. ... send the graph directly to a file. This is done by specifying a filename as parameter to the final `Stroke()` method call. See Section 5.5.4, “Writing the image directly to a file” for more detailed information.
2. ... access the GD image handler for further image processing (also needed to include the image in an PDF file, see Appendix C, *FAQ*)

3. ... make use of the built-in cache system to send back a previously generated image. The cache system, which lessens the burden of the PHP server, works by avoiding running all the code that follows the initial Graph() call by checking if the image has already been created and in that case directly send back the previously created (and stored in a file) image file to the browser. The filename used for the image can be either manually selected or automatically created based on the script name. In addition it is also possible to specify a timeout value in the initial call to the Graph() constructor to indicate how long the image in the cache directory should be considered valid before a new image is generated. A full description of the JpGraph cache system is available in Section 5.6, "Efficient graph generation using the built-in cache subsystem".

Note

The cache system by default is disabled and must be enabled by setting the proper define in the file "jpg-config.inc"

4. ... combine several graphs in the same image using the MGraph() class (Multi-Graph). This is an advanced technique described in ??.

5.5.2. Choosing the image compression format for Jp-Graph

By default JpGraph automatically chooses the image format to use in the order PNG, JPEG and GIF. The exact format depends on what is available on the system the library is installed on. There are two ways you can influence the way the graphic format is chosen:

1. Change the default graphic format by changing the DEFINE (in jpg-config.inc.php)

```
DEFINE('DEFAULT_GFORMAT', 'auto')
```

For example; if you by default want all your images to be generated with JPG encodation the define should be changed to

```
DEFINE('DEFAULT_GFORMAT', 'jpg')
```

2. By dynamically (in your script) select the wanted compression format with a call to

```
Image::SetImgFormat()
```

For example; if you want your image to use the JPEG format

```
$graph->img->SetImgFormat('jpeg')
```

(The above line assume that you have called your variable that holds the instantiated Graph() object "\$graph"

5.5.3. Sending back the image to the browser

The very last statement in almost all graph scripts is the line

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

Note

Actually there are some valid exceptions to this when you do some more advanced graph generation involving caching together with the CSIM functionality.

This line starts the actual graph creation. All method calls up to this stage has just been to set the scene for the library and specify all necessary parameters. It is first when you make the call to the `Stroke()` method the library actually starts to build the image. Assuming there are no errors detected when the image is generated the library will now take the following steps in principle:

1. Start building the image in memory. This is done by analyzing the specified parameters and making use of the supplied data in order to create the various plots that have been specified.
2. Check what headers are needed, i.e. what image compression are used for the graph, and send that header back to the client.

The library also have to check how the library was called since if it was called from the command line no MIME headers should be sent back at all, just the raw image data. Running the command line version of PHP will allow you to dynamically create images without using a HTTP server.

3. Send the actual image data representing the built up image back to the client

The dreaded: Headers has already been sent error

This is an error that everyone, and we really mean everyone, will see one time or the other when producing dynamic images with PHP.

First, this is not a problem with JpGraph per se. What has happened is that your PHP script which produces the image has already returned some data to the client before the image header has been sent.

This is most often caused by one or more spaces before the first "`<?php`" statement. What happens is that the server normally sends back all data it finds in the files it reads. Since the server sees a space, a perfectly valid character, it will send that data back to the client. However, before it does that it will automatically generate a header. Since it has seen a normal character data it will generate a header telling the client to expect a data stream of characters.

When later JpGraph tries to send its image header the server will detect that a header has already been sent and since each HTTP data stream can only have one type (and hence only one header) it will generate an error message which is sent back to the client.

To correct this error check your files for any output (even a single space) before the call to `Graph::Graph()` (or `Graph::Stroke()`). If you are running on older version of a Windows server this problem could also be caused by blank line at the end of the files. On some older Windows versions together with PHP4 it might also be caused by a file ending in a newline (which all the JpGraph library files does). Remove the newline so that the file ends just after the final "`?>`". Also remember that when you include external file using `include/include_once` and so on PHP includes the whole content of the file; this content of the file also includes any potential carriage return/line feed or "blank" space before "`<?php`" and after "`?>`". These "dirty characters" will cause the problem just described.

5.5.4. Writing the image directly to a file

In addition to just streaming the file back to the browser it is also possible to write the file directly to a named file. The file name is given as an argument to the `Graph::Stroke()` method. For example as

```
$graph->Stroke('tmp/myimage.png');
```

There are three important things to note here

1. The PHP process must have write permission on the directory you are trying to write the image file on. If you are running PHP through your browser this means that the HTTP server process must have write permission on that directory.
2. The file suffix (e.g. '.png') should match the image compression type used.
3. If the image is streamed directly to a file and not back to the browser the script can of course return ordinary text.

Writing the image to both a file and stream it back to the browser

In this case you should instead use the method `Graph::StrokeStore($aFileName)` which was introduced in version 2.5 of the library. If you are on a previous version and for various reasons cannot upgrade then you can use the following "trick" to achieve this.

The idea is to use the `_IMG_HANDLER` option that forces the `Graph::Stroke()` to just return the image handler and then stop. We can then manually first send the image to the chosen file and then stream it back to the browser using some internal methods in the library. The following code snippet shows how this is done.

```
<?php
// ... necessary includes ...

$graph = new Graph(400,300);

// ... code to generate a graph ...

// Get the handler to prevent the library from sending the
// image to the browser
$gdImgHandler = $graph->Stroke(_IMG_HANDLER);

// Stroke image to a file and browser

// Default is PNG so use ".png" as suffix
$fileName = "/tmp/imagefile.png";
$graph->img->Stream($fileName);

// Send it back to browser
$graph->img->Headers();
$graph->img->Stream();
?>
```

5.5.5. Alternatives to streaming or storing the image

There are also two predefined filenames which have special meaning when supplied as argument of the `Stroke()` method.

- | | |
|---------------------------|---|
| <code>_IMG_AUTO</code> | This will create a file in the same directory as the script with the same name as the script but with the correct image extension. |
| <code>_IMG_HANDLER</code> | Specifying this filename will not create a an image to file nor stream it back to the browser. Instead it will instruct the <code>Stroke()</code> method to just return the handle for the GD image. This is useful if you later want to manipulate the image in ways that are not yet supported by JpGraph. For example include the image in a dynamically |

generated PDF file. See Appendix C, *FAQ* for a detailed example how to include an image in a PDF generated with the "fpdf" library.

5.5.6. Forcing the browser to update your graph

Some browser may not send back a request to the HTTP server unless the user presses "Refresh" (F5 - in most browsers). This can lead to problems that the user is seeing old data since the file stamp of the script might not change but the data the script is using to create the image/graph is. A simple trick is to add a dummy time argument which is not used in the script.

For example

```

```

Since the dummy argument will be a new number whenever the browser checks it the browser understands that it must re-fetch the script and force the image to be reloaded and redisplayed.

It is also important to be aware of any internal caching the browser might do. The general problem with dynamically generated images is that the image generating script (file) remains the same. This makes the browser believe that the data hasn't changed (since the script is the same) and if the browser already has issued a previous GET request and has the data cached it will not send a new GET if the time stamp on the file is the same since it then believes it should and can use the old browser cached version.

5.5.7. Printing the generated image

Some browsers, most notable IE (< v7) can have issues printing a dynamic image. This is because the designers of IE assumed that all images are traditional images that are available as static image files. Not that they could be dynamically generated. This unfortunately have some implications.

1. IE will often (always?) re-fetch the page when preparing to print. This means that a new image will be generated and is perhaps very different from what the user thinks he is printing (if the data is changing rapidly).
2. Some older versions of IE simply refuses to print dynamic images if they are not available as a static "*.png", "*.jpg" etc. file. The only known workaround is to make sure to use static images.

There is one final reported problem to be aware of. Normally most browsers will support "right-clicking" on an image to download the image locally. However, some older versions of IE will become very confused when dynamic images are used. This could manifest itself as that the file type is not the wanted, for example, trying to download a "*.png" image could cause the file to be saved as a "*.bmp" file instead.

Newer versions of IE seems to be able to handle dynamic images much better.

It should also be mentioned that some older versions of FireFox (< v3) could in some circumstances fetch a dynamic image twice causing unnecessary load on the server (See FireFox Bugzilla [https://bugzilla.mozilla.org/show_bug.cgi?id=331126]). However, there are no known issues with dynamic images in current versions of FireFox and IE (i.e. IE v8).

5.6. Efficient graph generation using the built-in cache subsystem

The full explanation of how to use the JpGraph cache system is deferred to Chapter 9, *Using the JpGraph cache system* so this early section is just to explain the principles and why you probably want to read through the full chapter later on.

The core problem is that generating images can require a lot of CPU time. If we combine this with the interpreted nature of PHP we can get a potential lethal performance mixture. For example, if a web page uses one dynamically generated graph and at its peak the site has 50 hits per second this means that the PHP graph script also has to be called 50 times a second. On a high end server a typical graph takes in the range of 0.02s up to 0.2s (and some complex graph, for example an anti-aliased Contour-plot, can take up to 1s) to generate using a plain non-accelerated installation (see more on supported PHP Accelerators in Chapter 11, *NuSphere PHP accelerator*).

If we furthermore makes the realistic assumption that the site needs to run some database services and generate other content as well the graph script itself might not be allowed to use more than 50% of the available CPU bandwidth. It is now easy to see that we are in trouble.

Continuing with the example. Assuming our graph is medium complex and takes a whole 0.04s to generate. This means that if the server does not do anything more than just creating images then we can crank out 1/0.04 or 25 graphs per second per core at maximum. If we then take the 50% load into account it means that we could only be allowed to generate ~12 graphs per second per core and remember that this was on a high end server. If we now have more than 12 hits per second or have a more average server with a less powerful CPU we are heading directly for a seriously under dimensioned server.

If we assume we are running our server on a dual-core CPU this gives us an upper practical limit of 24 graphs per second per server.

There are a couple of ways to counteract this problem but none is a 100% solution.

1. Introduce load balancing among several servers. This is a common way to dynamically adjust to varying loads but without some serious investments into a server farm it is still possible to overload the system.
2. Reduce the complexity of the graphs. However there is still a limit, even very simple and basic graphs takes in the order of 0.02s on a high end server due partly to the cost PHP parsing of all the library files. This means that the upper limit is still valid.
3. Make use of a PHP accelerator. If you are running a professional server this is a must. We would go so far as to say that running a professional PHP server without using any of the available PHP accelerators borders on gross misconduct on the behalf of the information architect that did the logical server design. Using a PHP accelerator we can normally expect to get an almost 100% improvement. Taking the previous example as a case we can probably expect an upper limit of ~24 graphs per second per core instead of the normal ~12.
4. Make use of the built in cache system in JpGraph. This system if set up correctly will avoid the problem by not having to generate the same (or almost the same) image over and over again. The core idea of the system is the observation that the majority of the data presented on a WEB page is changing only very slowly compared with the hits a server gets.

For example, if a server has a graphical overview of defects by showing inflow/outflow there is probably a good case to state that the data change sufficiently slowly that it is probably not necessary to re-generate the graph more than 6 times per day (or perhaps even just once per day).

The JpGraph cache system allows you to do just that. What you do is specify a timeout. When the server calls the graph script the script first checks if a previous image has been generated. If that is the case it then checks how old the image is and if it is newer than the timeout limit then the existing image is sent back. It is just in the case that there isn't already any image (the script has never been run) or that the existing image is too old (older than the specified timeout limit) that the image gets generated again.

You could of course do all of this manually but the library has built in support for this that will allow you to use the system without changing a single line in your existing scripts. It is all taken care of in the background.

For a production server the recommended setup is to use a combination of load balancing, a suitable PHP accelerator and enabling the JpGraph cache system.

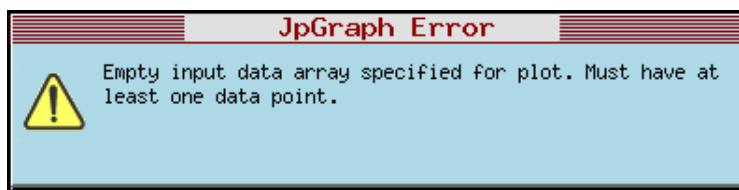
Chapter 6. Error handling

6.1. The problem with error messages and images

Why a whole chapter on error handling? The answer is that error handling with scripts that produces images requires a slightly different approach than normal error messages. In order to appreciate the added complexity one needs to consider how a graph script is called. Usually a graph script is called from an `` tag. This means that the client (e.g. the browser) expect image data to be sent from the script. If we instead send a textual error message the browser will not be able to display anything. It was expecting a stream with image data and the data received was a representation of an error message which of course cannot be interpreted as an image.

To handle this problem the error messages produced by the library are normally generated as an image. A typical error message can be seen in Figure 6.1, "Typical image error message"

Figure 6.1. Typical image error message



There is one exception to this rule and that is in the case of the "Header already sent" error message. This means that the client has already received an header and with probability bordering on certainty this is a text header. Hence in this case it does not make sense to send an image so this, and only this, error message is sent as a normal "text/html" type message. A typical example how this message is shown in the browser can be seen in Figure 6.2, "The "Header already sent error message""

Figure 6.2. The "Header already sent error message"

JpGraph Error: HTTP headers have already been sent.
Caused by output from file test_lonestep.php at line 2.

Explanation:
HTTP headers have already been sent back to the browser indicating the data as text before the library got a chance to send its image HTTP header to this browser. This makes it impossible for the library to send back image data to the browser (since that would be interpreted as text by the browser and show up as junk text).

Most likely you have some text in your script before the call to `Graph::Stroke()`. If this texts gets sent back to the browser the browser will assume that all data is plain text. Look for any text, even spaces and newlines, that might have been sent back to the browser.

For example it is a common mistake to leave a blank line before the opening "`<?php`".

When you get this error (and we mean when - not if) take a very close look at the text in red. This will tell you in what file and what line the erroneous output started. In most cases this will be an innocent looking space or a tab character. It could also be caused by multiple newlines at the end of a file. Since by definition these mistakes can be hard to spot since spaces and tabs are not normally visible in an editor so take care!

6.1.1. Catching errors in other parts of the system while creating images

The problem described in the previous section will also apply to error messages that are generated directly from PHP as well. In order to be able to see any potential error messages during development it is possible to instruct the library to intercept PHP error messages and convert them to image messages instead. This is controlled by the two defines `INSTALL_PHP_ERR_HANDLER` and `CATCH_PHPERR` define in `jpg-config.inc.php`

`INSTALL_PHP_ERR_HANDLER` Setting this define to true will make the library install a custom error handler which will catch all PHP error messages and convert them into an image like what can be seen in Figure 6.1, “Typical image error message”.

By default this is disabled.

`CATCH_PHPERR` This defines control whether the library shall check for any generated error message that are stored in the global PHP pre-defined variable `php_errmsg`. This can be very useful during development if an error has occurred prior to calling the graph script. This error will then be displayed as an image.

By default this is enabled.

6.2. Available error messages

All error messages that can be generated by the library are listed in Appendix H, *Error messages*

6.2.1. Localizing error messages

The library includes two (actually two and one pseudo) localizations of the error messages. As of version 2.5 the library supports English and German error messages. The choice of which localization of error messages that should be used can be done in two ways; either by specifying a global define in `jpg-config.inc.php` or

As of version 2.5 there are three localization options

1. English error messages ("en")
2. German error messages ("de")
3. Production error messages. This is not really a localization but a different set of error messages which does not give detailed error messages but a generic message suitable for a production server where the end user is not helped by detailed graph script errors. Instead a generic message is shown together with an error code that corresponds to the detailed error. ("prod")

In order to specify the error message localization the following define in `jpg-config.inc.php` must be set the

```
define('DEFAULT_ERR_LOCALE', 'en');
```

The possible (string) options are

1. "en", English locale
2. "de", German locale

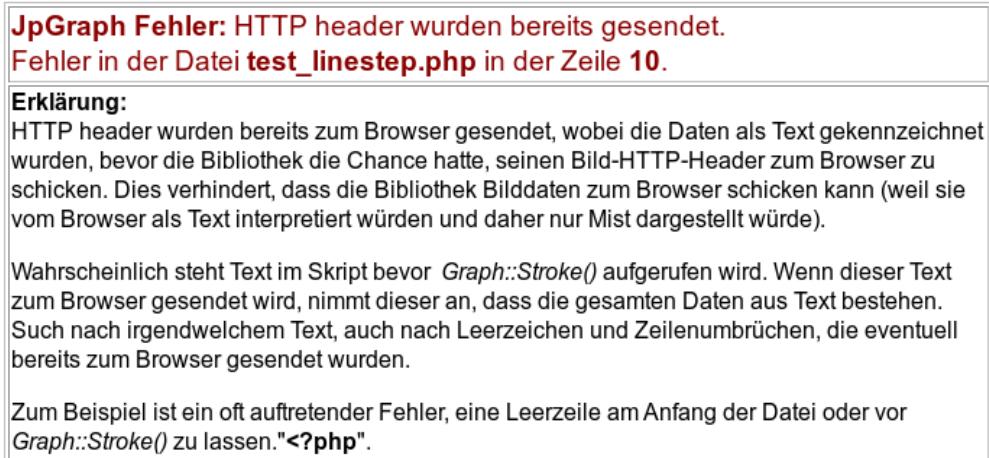
3. "prod", The production version of the error messages.

It is also possible to dynamically set the localization of error messages by calling the static method `JpGraphErr::SetErrLocale($aLocaleString)`. For example, in order to dynamically change to German locale the following method call would be used:

```
JpGraphErr::SetErrLocale('de');
```

As an example of the German locale the "*Header Already Sent*" error message localized in German are shown in Figure 6.3, "The "Header already sent" error message using German locale"

Figure 6.3. The "Header already sent" error message using German locale



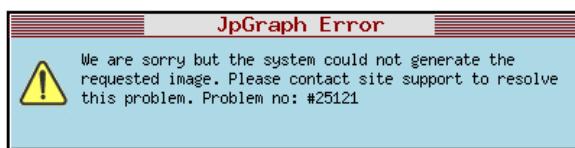
Note

There is one potential issue with specifying the locale dynamically. Since the call to set the locale happens in the script at a certain point this means that if the error occurs before that point the locale change has not yet happened and the default locale specified in `jpg-config.inc.php` will be used.

6.2.2. Error messages on a production server

On a production server it is common not to show detailed specific error messages to the public which could potentially give away security vulnerabilities. For this purpose there is a "pseudo" locale of the error messages. This is specified by setting the locale (as described above) to the string "prod". If this is done a generic error message with an error code will be shown instead. The custom support receiving information on this error can then pass on the detailed error code to the development team for further investigation. In Figure 6.4, "Using the production "pseudo" locale" the same error as is shown in Figure 6.1, "Typical image error message" is displayed using the "prod" locale. As you can see in this case only a more end-user appropriate message together with the error code is displayed.

Figure 6.4. Using the production "pseudo" locale



The problem number corresponds to the error codes that are listed in Appendix H, *Error messages*

Tip

It is possible to customize this "production" error message by changing the actual text in the file "lang/prod.inc.php"

6.3. Using PHP Exceptions

Starting with version v2.5 the library now have full support for PHP5 style exceptions. The library provides an exception class named `JpGraphException` which is slightly different compared with traditional exception classes in that this class can not only return an text string as error but also an image. This is necessary in order to handle the case where a script has an error and is called from within an `` tag. The only data that can then be displayed in a browser is image data and hence it is necessary for the error to be formatted as an image.

In addition to providing an exception class the library also installs its own default exception handler to properly display an image. This default exception handler will be automatically called whenever an "un-caught" exception would otherwise be generated. This means that it is strictly not necessary to use "`try { } catch() { }`" blocks around the library scripts.

When an exception is generated the default exception handler first validates that the exception is a proper descendant of `JpGraphException` and if so, generates the image by calling the `JpgraphException::Stroke()` method. If the exception is not a `JpGraphException` based exception then the handler re-raises the error. This means that in script that you create that is meant to be called from within an `` tag all exception should be a derivate of `JpGraphException` in order to properly generate an error image.

A typical example on how to raise an exception in your own code is shown in Example 6.1, "Throwing a JpGraph exception"

Example 6.1. Throwing a JpGraph exception

```
<?php  
throw new JpGraphException(' ... some error message ... ');\n?>
```

In case you need to handle the exception in a "`try { } catch() { }`" block (perhaps in order to do necessary cleanup) it is important to remember to call the `Stroke()` method which will create and stream the error message back to the browser. An example of this is shown in Example 6.2, "Catching a JpGraph exception and sending it back as an image to the client"

Example 6.2. Catching a JpGraph exception and sending it back as an image to the client

```
<?php
try {

    $graph = new Graph($width,$height);

    // ... Code to setup the graph

    if( /* some error condition */ ) {
        throw new JpGraphException('... some error message ...');
    }

} catch ( JpGraphException $e ) {
    // .. do necessary cleanup

    // Send back error message
    $e->Stroke();
}
?>
```

Tip

Another typical augmentation of the exception handling might be to also log an error to some logging server or plain log file. This should be done before the call to `Stroke()` since that call will never return.

An example of real life error handling with exception is shown in listing Section 4.2.2, “Preparing the data” in the introductory example with Sun spots.

6.3.1. Selecting between text and image based error handling

By default an exception that occurs in the library will create an error image as shown in the previous section. However there might be circumstance where a text based error handling is preferred (usually when graphs are created from the command line).

This could be accomplished in two ways. By catching the exception in the script and handle it accordingly or we could slightly modify the default behavior of the default exception handler in the library. How this is done will now be described.

In order to enable a text based error handler we just need to disable the image based error handler. This is done with a call to the method

- `JpGraphError::SetImageFlag($aFlag)`

Since the error handling have global scope this is a static function which can be called as the following example shows

```
JpGraphError::SetImageFlag(false); // Enable text based error handling
```

Adding the line above to a graph script will cause any error to be printed to STDERR when the script is called from the command line. This is a very convenient way to show errors when command line constructions like

```
$> php mygraph.php > mygraph.png
```

is used since writing the error to STDOUT will cause the error message to be sent back to the console since the call above only redirected STDOUT and not STDERR.

When the script is called from PHP embedded in a HTTP server (e.g. Apache) there is no concept of a STDERR and the error message will just be sent back as normal text to the browser.

6.3.2. Writing error message to a log file (or system logger)

In addition to the option of having the error sent back as a string to the client it can instead be written to a named log file. The log file name is specified with a call to the (static) method

- `JpGraphError::SetLogFile($aFileName)`

The file needs to be writable by the PHP process. All error messages are appended to the end of the file and each error message is prepended by the date and time (in RFC 2822 [<http://www.faqs.org/rfcs/rfc2822>] formatted date).

If the filename is given as the string 'syslog' then the error message will be written to the default system logger instead. When a script is run from, for example, Apache this is normally on Unix '/var/log/apache2/error_log'

6.4. Adding a new locale

All error messages are stored in an array with a specific index that cannot be changed. This index is used in the library to reference a specific error message.

Since error messages can also have additional parameters the error messages also includes count that specifies how many arguments must be handled to enable some error checking when using the error message in the code (to make sure we have enough arguments).

The arguments are formatted in the error text in the same way as parameters in the `printf()` family of functions.

To create a localized error resource file You should first copy the "en.inc.php" to a temporary file, rename it according to the locale you are creating and then translate each error message according to the locale. The make sure that the file is stored under the "jpgraph/lang/" catalogue and the new localized error messages can be used.

For example. To create a French version of the error messages the `lang/en.inc.php` should be copied to `lang/fr.inc.php` and the error messages translated. The French locale can then be used by using the string "fr" as identified in `SetErrLocale()` as described above.

Chapter 7. Color handling

An important part of creating visually clear graphs is the use of appropriate colors. In order to simplify color handling JpGraph supports several ways to adjust and manipulate color both by value and by name.

Almost all color setting methods (with basically only one exception) for all objects in the graph has one of two names

- `SetColor()`, Sets the outline color or if the object only have one color (e.g. a font) it sets this color
- `SetFillColor()`, Specifies the area fill color for objects which has a concept of an area.

Both variants of methods take one argument which identifies the color by one of the available color specification methods as described below.

7.1. Specifying colors by name

There are a number of "standard colors" known by (more or less) illustrative names. A list of all color names that can be used as well as actual color can be found in Appendix D, *Named color list*.

For example:

- `SetColor('white');`
- `SetFillColor('orange');`

Tip

Always use single quotes for strings when you do not need variable substitution since this is faster.

7.2. Specifying colors by RGB triples

Using the named colors will of course be restricted to the colors known by the library. If some other colors are needed then they must be specified manually.

The first variant uses the RGB values for the color, i.e. the Red, Green and Blue component of a color. Each component is specified as an integer in the range [0,255] and the triple is specified as an array.

For example:

- `SetColor(array(255,255,255));`
- `SetColor(array(0xff,0xff,0xff));`
- `SetFillColor(array(0x44,0x54,0xa4));`

As usual a value of `array(0,0,0)` specifies black and a value of `array(255,255,255)` specifies white.

7.3. Using HTML color specifications

The second variant is more or less the same as the RGB triples but instead of an array the RGB triple is specified as a string with the hexadecimal RGB values. This is commonly known as the "HTML color specification" since this form often used when creating WEB pages (for example in CSS sheets).

For example:

- `SetColor('#333')`, note that this is a short form for `SetColor('#333333')`
- `SetColor('#12be7a')`
- `SetFillColor('#99eff5')`

7.4. Fine tuning the color

7.4.1. Specifying the alpha channel (color transparency)

Colors can also be made semi-transparent by specifying a transparency value (or as it is also known an *alpha channel value*). This will instruct the library to mix the foreground color with a certain amount of the background colors creating a "shine-through" effect.

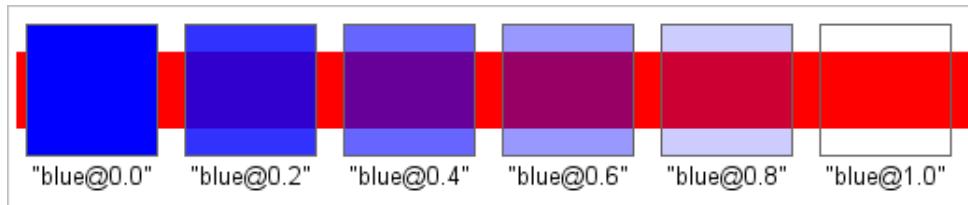
The alpha channel is specified as a real number in the range [0.0, 1.0] where 0.0 means no transparency and 1.0 means full transparency (i.e. only the background color is shown). The transparency is added as a postfix to the color specification separated by a '@' (at) character. It is often most used with area fills (i.e. when using `SetFillColor()`)

For example the following are valid alpha channel modifiers

- `SetFillColor('red@0.2')`, A slightly transparent red color
- `SetFillColor('red@0.8')`, An almost completely transparent color

The result of using different alpha modifiers are shown in Figure 7.1, “Using alpha channel modifiers” where the blue color is made more and more transparent to allow the red bar in the background to become more and more visible.

Figure 7.1. Using alpha channel modifiers



As a final example of how to use transparency we show a graph in Figure 7.2, “Making use of transparency to combine two plots (barlinealphaex1.php)” that uses transparency to allow an area plot to be mixed with a bar graph.

Figure 7.2. Making use of transparency to combine two plots (barlinealphaex1.php) [example_src/barlinealphaex1.html]



Tip

For all graph examples shown in this manual you can always click on the file name in the title of the graph to view the actual source that created the graph.

7.4.2. Adjusting the brightness

Since all colors that can be displayed on a computer can be modeled with the appropriate RGB triple the above ways will allow you to specify any colors you might need. However, conceptually it might be easier to think of colors in terms of "a light blue" instead of a modified RGB triple so the library allows you to take a basic color and apply a brightness correction factor.

The brightness factor is a postfix to the color string separated from the core color with a ':' (colon) character. The brightness factor is a real number in the range of [0.0, 2.0]. Using a brightness factor of 0.0 means that the color will be black (no brightness at all) and a factor of 2.0 means that the color is white (maximum brightness). Using a factor of 1 will leave the original color untouched.

The brightness factor can be specified for named and HTML colors, see Figure 7.3, “Adjusting brightness of named color specifier” and Figure 7.4, “Adjusting brightness of a HTML color specifier” for more examples how the factor adjusts the color.

For example, the following code snippets show valid color brightness modifiers

- `SetColor('red:0.8');`, A slightly darker red color
- `SetColor('red:1.5');`, A brighter red color
- `SetColor('#3485a9:1.8');`, A bright blue-greenish color

Figure 7.3. Adjusting brightness of named color specifier

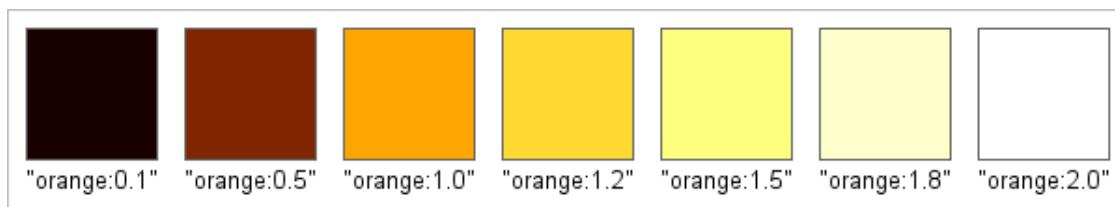
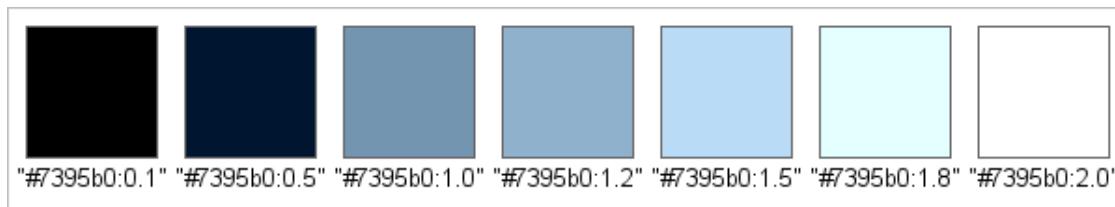


Figure 7.4. Adjusting brightness of a HTML color specifier



Note

The brightness factor is a purely linear factor but due to the human eyes non-linear sensitivity for colors the perceived difference between (for example) "`red:1.1`" and "`red:1.2`" than between "`red:0.2`" and "`red:0.3`" even though the relative difference is the same.

7.4.3. Combining brightness and transparency adjustment

It is also possible to combine the two previous discussed modifiers. This is done by first adding the transparency specifier and then the brightness adjustment as the following examples demonstrates.

- `SetColor('red@0.7:1.2')`, A highly transparent slightly bright red color
- `SetFillColor('#4545aa@0.3:1.5')`, A bright blueish semi transparent color

7.5. Additional color handling

Some plot types have additional color handling to facilitate easier handling and differentiation between data sets.

Pie plots Pie plots support the concept of color-themes. A color theme is simple a set of colors used for consecutive slices within the Pie. How to specify color-themes is further discussed in Section 16.1.9, “Specifying slice colors and using themes”.

Color maps Matrix (Only available in the Pro-version) and contour plots also have the concept of color maps. This is a range of colors used to indicate the data in the plot. Examples of color maps can be found in Section 22.8, “Built in color maps”. (Color maps are defined and implemented in "jpgraph_colormap.inc.php").

Chapter 8. Text and font handling

Before reading this chapter please make sure that your installation supports TTF fonts if you intend to use them (See Section 3.2, “Necessary system requirements for the library”). The rest of this chapter assumes that a working installation with TTF fonts is available when needed.

Note

It is highly recommended to create a setup where TTF fonts work since the visual quality is significantly better than for the built-in bitmap fonts. In addition the bitmap fonts are restricted to only display plain 7-bit ASCII characters which means that no accented characters can be displayed, for example the Scandinavian characters "äö".

Note

All files in the library are encoded in utf-8.

8.1. Different types of fonts

The library supports two fundamental types of fonts.

1. Bitmap fonts

There are three built in bitmap fonts. They are available as font families FF_FONT0, FF_FONT1 and FF_FONT2. The advantage with bitmap fonts is that they are always available in all installations of GD. However, bitmap fonts only supports 7-bit ASCII so if you need to display any character from the extended character set it is not possible to use bitmap fonts. The available sizes of the bitmap fonts are also limited, the three available size corresponds to the three families where FF_FONT0 is the smallest and FF_FONT2 the largest available bitmap font.

Bitmap fonts also has a more "rugged" look since they do not use anti-aliasing.

Example: *The following script lines shows a typical use of bit map font specified for the title of a graph*

```
<?php  
$graph = new Graph(....);  
  
// Adjust the title to use the largest built-in bitmap font in bold face  
$graph->title->SetFont(FF_FONT2,FS_BOLD);  
?>
```

The remainder of this chapter will describe this in more details.

2. TTF Fonts (True Type Fonts)

True Type Fonts (TTF) give a much better visual quality of the text. They are available in all sizes and there are many more font family to choose from. All font family specification apart from the three bitmap fonts are TTF fonts. In order to use these fonts the installation must be configured in the right way which is described in Section 3.2, “Necessary system requirements for the library”. It is necessary to use TTF fonts in order to display extended character sets (outside the traditional 7-bit ASCII). TTF fonts must also be used when inserting unicode entities as described in Section 8.7, “Inserting Unicode entities”

In order to be able to use TTF fonts it is necessary to check if the installations supports this. See Section 3.2, “Necessary system requirements for the library”

Example: *The following script lines shows a typical use of the Arial TTF fonts for the title of a graph*

```
<?php
$graph = new Graph(....);

// Adjust the title to use 14pt Arial bold face
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);
?>
```

8.2. Font families and font styles

All graph objects that uses text allows you to specify the font to be used by calling a `SetFont()` method and specifying three parameters.

1. The font family. This could for example be "Arial", "Times roman" for TTF fonts or "Bitmap font medium size" when using bit-mapped fonts. The font family is specified by using a symbolic constant that starts with the prefix "FF_" (for Font Family). A list of all supported font families are shown in Table 8.2, “Supported Latin Font family specifiers”.
2. The font style. This specifies if the font should be emphasized as italic, bold, or bold-italic. By default all fonts are rendered with the "normal" style. The fonts style is specified with a symbolic constant that starts with the prefix "FS_" (for Font Style). The supported font styles are:

Table 8.1. Supported Font Styles

Font style specifier	Description
FS_NORMAL	Normal font style
FS_BOLD	Bold font style
FS_ITALIC	Italic font style
FS_BOLDITALIC	Bold + Italic style for font families that support this

Please note that not all font families support all available styles.

3. The font size. This is specified as an integer and depicts the size of the font given in typographic points (pt).

Table 8.2. Supported Latin Font family specifiers

Font family name	Type	Comment	Bold	Italic	BI
FF_FONT0	Bitmap	A very small font			
FF_FONT1	Bitmap	A medium sized font	✓	✓	✓
FF_FONT2	Bitmap	The largest bit mapped font	✓	✓	✓
FF_ARIAL	TTF	Arial font	✓	✓	✓

Font family name	Type	Comment	Bold	Italic	BI
FF_VERDANA	TTF	Verdana font	✓	✓	✓
FF_COURIER	TTF	Fixed pitched Courier new font	✓	✓	✓
FF_BOOK	TTF	Bookman	✓	✓	✓
FF_COMIC	TTF	Comic sans	✓		✓
FF_TIMES	TTF	Times New Roman	✓	✓	✓
FF_GEORGIA	TTF	Georgia	✓	✓	✓
FF_TREBUCHE	TTF	Trebuchet	✓	✓	✓
FF_VERA	TTF	Gnome Vera font. All Vera family fonts are available from http://www.gnome.org/fonts/	✓	✓	✓
FF_VERAMONO	TTF	Gnome Vera Mono font	✓	✓	✓
FF_VERASERIF	TTF	Gnome Vera Serif font	✓		✓
FF_DV_SANSSERIF	TTF	DejaVu Font family. The DejaVu fonts are modifications of the Bitstream Vera fonts designed to extend this original for greater coverage of Unicode, as well as providing more styles. These fonts are available from http://dejavu-fonts.org/	✓	✓	✓
FF_DV_SANSSEIFMONO	TTF	DejaVu Font family.	✓	✓	✓
FF_DV_SANSSEIFCOND	TTF	DejaVu Font family.	✓	✓	✓

Font family name	Type	Comment	Bold	Italic	BI
FF_DV_SERIF	TTF	DejaVu Font family.	✓	✓	✓
FF_DV_SERIFC	TTF	DejaVu Font family.	✓	✓	✓
FF_CHINESE, FF_BIG5	TTF	<p>Chinese font (both symbolic names can be used).</p> <p>The actual font file name used for the Chinese font is "bkai00mp.ttf". The name of this file can be changed in the define CHINESE_TTF FONT available in the library file jgraph_ttf.inc.php</p>			
FF_SIMSUN	TTF	<p>Chinese font.</p> <p>The actual font file name is "simsun.ttc"</p>			
FF_MINCHO	TTF	<p>Japanese font.</p> <p>The actual font file name is "ipam.ttf"</p>			
FF_PMINCHO	TTF	<p>Japanese font.</p> <p>The actual font file name is "ipamp.ttf"</p>			
FF_GOTHIC	TTF	<p>Japanese font.</p> <p>The actual font file name is "ipag.ttf"</p>			
FF_PGOTHIC	TTF	<p>Japanese font.</p> <p>The actual font file name is "ipagp.ttf"</p>			
FF_DAVID	TTF	Hebrew font.			

Font family name	Type	Comment	Bold	Italic	BI
		The actual font file name is "DAVIDNEW.TTF"			
FF_MIRIAM	TTF	Hebrew font. The actual font file name is "MRIAMY.TTF"			
FF_AHRON	TTF	Hebrew font. The actual font file name is "ahronbd.ttf"			

Note

Depending on what versions of the DejaVu fonts are installed on the system there are two naming conventions for the font files in common practice. The library will try both naming convention to see if it can find the font files.

In Figure 8.1, “List of all latin TTF fonts. (listfontsex1.php) ” all latin fonts are shown.

Figure 8.1. List of all latin TTF fonts. (`listfontsex1.php`) [`example_src/listfontsex1.html`]

Normal	Italic style	Bold style
Font 0		
Font 1		Font 1 bold
Font 2		Font 2 bold
Bitmap Fonts		
Arial	<i>Arial italic</i>	Arial bold
Verdana	<i>Verdana italic</i>	Verdana bold
Trebuchet	<i>Trebuchet italic</i>	Trebuchet bold
Georgia	<i>Georgia italic</i>	Georgia bold
Comic		Comic bold
Courier	<i>Courier italic</i>	Courier bold
Times normal	<i>Times italic</i>	Times bold
Vera normal	<i>Vera italic</i>	Vera bold
Vera mono normal	<i>Vera mono italic</i>	Vera mono bold
Vera serif normal		Vera serif bold
DejaVu sans serif	<i>DejaVu sans serif</i>	DejaVu sans serif
DejaVu serif	<i>DejaVu serif</i>	DejaVu serif
DejaVuMono sans serif	<i>DejaVuMono sans serif</i>	DejaVuMono sans serif
DejaVuCond serif	<i>DejaVuCond serif</i>	DejaVuCond serif
DejaVuCond sans serif	<i>DejaVuCond sans serif</i>	DejaVuCond sans serif

8.3. Understanding text alignment and anchor point

When a text string position is specified that screen (or scale) position by default gets aligned with the top left corner of the strings bounding box. We say that the top left corner is the *anchor point* of the text string. The alignment of the anchor point can be adjusted with a call to `Text::SetAlignment($aHorAlign, $vertAlign)`. The two arguments are given as text strings and the admissible values for each argument are:

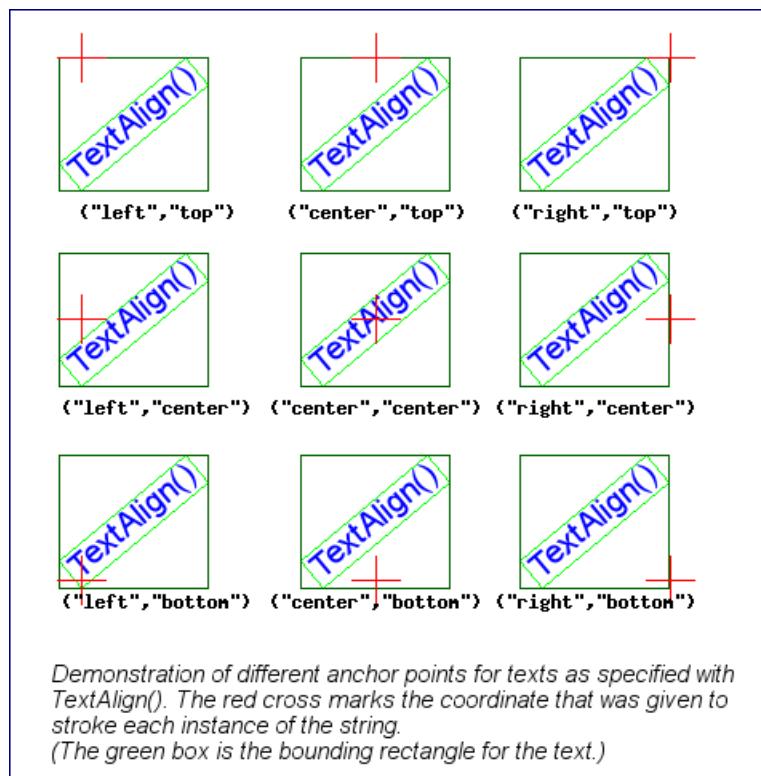
Horizontal alignment	'left', 'center', 'right'
Vertical alignment	'bottom', 'center' (or 'middle'), 'top'

Caution

Even though from an API perspective both bitmap fonts and TTF fonts share the same user API the implementation is vastly different. There are a number of subtle differences in the way built-in bit-map fonts and TrueType fonts are rendered to the screen. This means for example that the alignment of, say the bottom of the text string, is not pixel-perfect between bitmap and TrueType fonts. However, JpGraph abstracts away 99.9% of the differences so it will be, for all practical uses of the library completely transparent to switch between the different font types.

Manually setting the alignment for the anchor point is mostly useful when adding Text object to the graph to get the wanted alignment. You can see an example of how the anchor point changes depending on how the combination of alignments are used in Figure 8.2, “Illustration of anchor point alignment (`textalignex1.php`)”

Figure 8.2. Illustration of anchor point alignment (`textalignex1.php`) [[example_src/textalignex1.html](#)]



Note

It might seem strange to have the method name "SetAlignment()" when it really should make more sense to use the name "SetAnchor()". We agree. This naming scheme is due to historical reasons.

Note

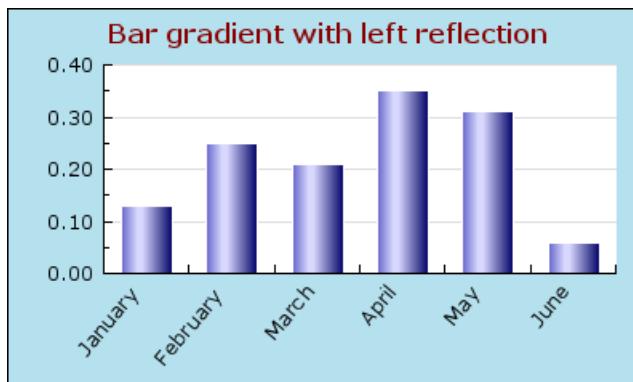
The graph legend box also have an anchor point that is specified as the 3:rd and 4:th argument to `Legend::SetPos()`

8.4. Rotating text

Both bit map and TTF fonts supports rotating text to different extent. With bit map fonts it is only possible to use horizontal or vertical text, i.e. 0 or 90 degree rotation. TTF fonts supports arbitrary angles. If you are using a bit map font and specifies an angle other than 0 or 90 then an error will be displayed.

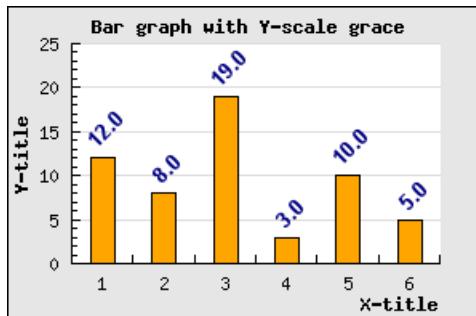
The most common usage for rotating text is probably to adjust the labels on the x-axis so they are at 45 degrees angle. To rotate the label of an axis the method `Axis::SetLabelAngle()` should be used. Figure 8.3, “Example of how to use rotated labels (`bargradex1.php`)” shows an example of this (click on the filename to view the actual code).

Figure 8.3. Example of how to use rotated labels (`bargradex1.php`) [[example_src/bargradex1.html](#)]



In addition to axis label it is also possible to rotate almost every other text object with the exception of graph titles which is always horizontal. For text objects (the class `Text`) that can be added to arbitrary positions on the graph the method `Text::SetAngle()` can be used to specify the wanted text angle. Another common place where text labels are rotated is when individual data points are marked with labels. This could be done for most plot types and in Figure 8.4, “Example of using rotated data point values (`example20.3.php`)” we show an example of using this for adding labels to a basic bar plot (click on the filename to view source).

Figure 8.4. Example of using rotated data point values (`example20.3.php`) [[example_src/example20.3.html](#)]



Caution

When rotating text paragraph the alignment (within the paragraph) will always be reset to "left". It is not possible to use "center" or "right" paragraph alignment in rotated texts.

8.5. Formatting text paragraphs

The text rendering engine within the library offers some basic text paragraph formatting.

- It is possible to use multi line text combined as a paragraph
- It is possible to adjust the text alignment within the paragraph to be left, right or center adjusted. This works in the same way as you would expect in a word-processor.
- If enabled the text rendering will support automatic line breaks at a certain column. The line breaks are intelligent not to break words in the middle.

All text handling is centralized to the class `Text` (defined in file `jgraph_text.inc.php`) which is used both to add arbitrary text to the graph as well as internally within the library to manipulate text on labels and titles. All such texts are an instance of the `Text` class.

The paragraph alignment is controlled by the method `Text::SetParagraphAlign($aAlignment)`. The argument is a text string that should be one of '`left`', '`right`' or '`center`'.

Note

Unfortunately the library does not support an "even" paragraph alignment which in word processors adjust the kerning between individual characters to make the text have even left and right sides. Implementing this would require a much higher complexity than can be motivated for the type of text needed in a graph library.

In the Figure 8.5, "The different types of paragraph alignments (`textalignex1.php`)" the same text paragraph is rendered with the possible paragraph alignments.

Figure 8.5. The different types of paragraph alignments (`textalignex1.php`)
[[example_src/textalignex1.html](#)]

The day was rapidly becoming more and more strange.

Not only had he managed to get by the first pass without so much as a hint of questions but now when he could feel that the second pass wouldn't long be noone had yet seen him.

"left" paragraph align

The day was rapidly becoming more and more strange.

Not only had he managed to get by the first pass without so much as a hint of questions but now when he could feel that the second pass wouldn't long be noone had yet seen him.

"center" paragraph align

The day was rapidly becoming more and more strange.

Not only had he managed to get by the first pass without so much as a hint of questions but now when he could feel that the second pass wouldn't long be noone had yet seen him.

"right" paragraph align

Inserting a newline character "`\n`" in text will cause the text line to break and start on the next row. Note that the newline must be surrounded with double-quotes and not single quotes.

Note

Bitmap fonts that are rotated, i.e. vertical, does not support automatic line breaking. If line breaking is needed with vertical text then one of the TTF fonts must be used.

8.6. Adding custom TTF fonts

In addition to the predefined fonts it is possible to easily use up to three custom fonts. This is done by first specifying the name of the font file that should be used and then specifying the font family as either FF_USERFONT1, FF_USERFONT2 or FF_USERFONT3. A new font is installed by calling one or more of the methods

- Graph::SetUserFont1(\$aNormal,\$aBold,\$aItalic,\$aBoldIt) (or the synonym SetUserFont())
- Graph::SetUserFont2(\$aNormal,\$aBold,\$aItalic,\$aBoldIt)
- Graph::SetUserFont3(\$aNormal,\$aBold,\$aItalic,\$aBoldIt)

The argument to these methods should be the full font file name (including full path) for the normal, bold, italic and/or bolditalic variant of the font family. All arguments apart from "\$aNormal" are optional.

An example on how this can be used to use a special font for the title of a graph is shown in Example 8.1, "Specifying and installing a user specified font"

Example 8.1. Specifying and installing a user specified font

```
<?php
// ...
$graph->SetUserFont('/usr/share/fonts/ttf/digital.ttf');
$graph->title->SetFont(FF_USERFONT,FS_NORMAL,12);
$graph->title->Set('Test title '.$pi);
// ...
?>
```

8.7. Inserting Unicode entities

One reason for using TTF fonts is the possibility to insert unicode character/entities. With this we mean characters from the extended range that are not normally available on West European keyboards. This could for example be classical Greek characters often used in mathematics, for example the symbol for "pi". In order to make these and other commonly used "special" characters more accessible the library provides a utility class that makes it easy to use these characters without having to look up the corresponding unicode entities every time.

In order to specify characters not available on the keyboard the normal way is to specify their unicode code and include it in the text string with the prefix "&#". For example the unicode for the character "pi" is "03C0" in hex so to include this character in a text string you would have to write "This is pi π" note that the code should be given in decimal encoding in the string and always use 4 digits (pre-padded with 0:s as necessary). Since it is very tedious to lookup and encode special characters the library offers a simpler way. The SymChar class.

Note

This section can be skipped at first reading without loss of continuity.

8.7.1. The utility class "SymChar"

As described above it is tedious to have to lookup all the character codes. To simplify this the SymChar class allows you to find these characters with the english common name of these symbols instead. For example to create a string with the Greek character "pi" one would have to write the code as shown in.

Example 8.2. Using the SymChar class to display the Greek letter "pi"

```
<?php
// ...
$pi = SymChar::Get('pi');
$graph->title->Set('Test is pi' . $pi);
// ...
?>
```

All supported entities with there symbolic names are listed in Table 8.3, “Supported character entities in class SymChar”. The first argument to the static method `SymChar::Get()` should be the name of the entity and if the second optional argument is true then the capital version (if it exists) of the symbol should be returned.

Table 8.3. Supported character entities in class SymChar

Name	Unicode	Unicode capital	Comment
"alpha"	03B1	0391	Greek character
"beta"	03B2	0392	Greek character
"gamma"	03B3	0393	Greek character
"delta"	03B4	0394	Greek character
"epsilon"	03B5	0395	Greek character
"zeta"	03B6	0396	Greek character
"ny"	03B7	0397	Greek character
"eta"	03B8	0398	Greek character
"theta"	03B8	0398	Greek character
"iota"	03B9	0399	Greek character
"kappa"	03BA	039A	Greek character
"lambda"	03BB	039B	Greek character
"mu"	03BC	039C	Greek character
"nu"	03BD	039D	Greek character
"xi"	03BE	039E	Greek character
"omicron"	03BF	039F	Greek character
"pi"	03C0	03A0	Greek character
"rho"	03C1	03A1	Greek character
"sigma"	03C3	03A3	Greek character
"tau"	03C4	03A4	Greek character
"upsilon"	03C5	03A5	Greek character
"phi"	03C6	03A6	Greek character

Name	Unicode	Unicode capital	Comment
"chi"	03C7	03A7	Greek character
"psi"	03C8	03A8	Greek character
"omega"	03C9	03A9	Greek character
"euro"	20AC		Monetary symbol
"yen"	00A5		Monetary symbol
"pound"	20A4		Monetary symbol
"approx"	2248		Mathematical
"neq"	2260		Mathematical
"not"	2310		Mathematical
"def"	2261		Mathematical
"inf"	221E		Mathematical
"sqrt"	221A		Mathematical
"int"	222B		Mathematical
"copy"	00A9		Misc symbols
"para"	00A7		Misc symbols
"tm"	2122		Misc symbols
"rtm"	00AE		Misc symbols

Figure 8.6, “Rendered symbol characters corresponding to ” and Figure 8.7, “Rendered capital symbol characters corresponding to ???.” shows how these symbol look when they are rendered with the TTF font *Times Roman 14pt*. The symbols are in the same order as in the table above and are visually grouped by there category as specified in the column “Comment” in the table above.

Figure 8.6. Rendered symbol characters corresponding to

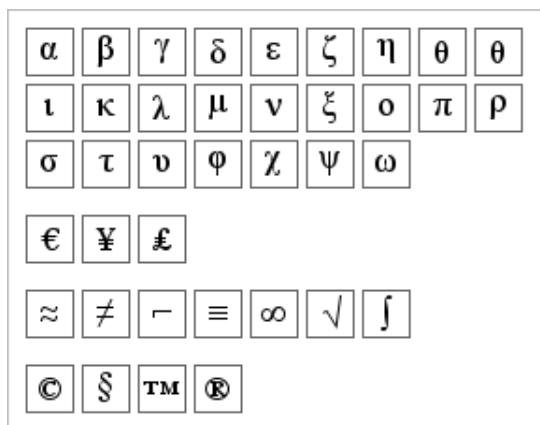
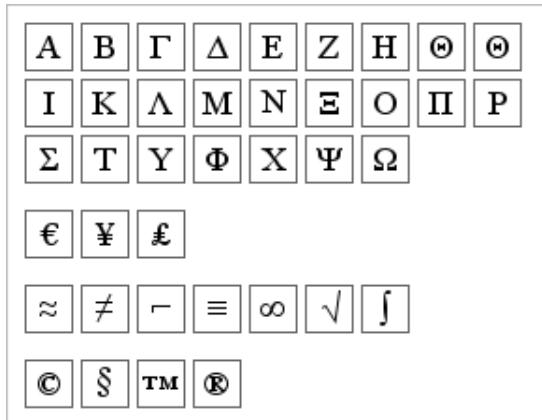


Figure 8.7. Rendered capital symbol characters corresponding to Table 8.3, “Supported character entities in class SymChar”.

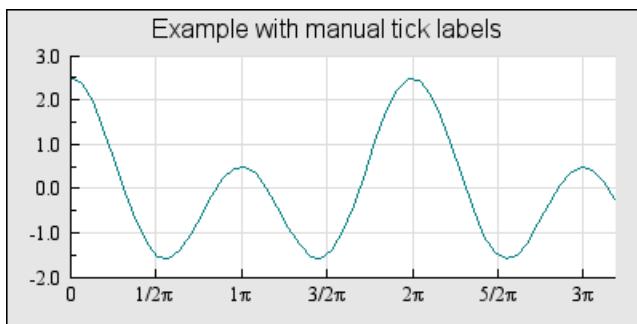


8.7.2. Graph example with Greek labels

In Figure 8.8, “Specifying manual ticks as fraction of Pi. (`manuallickex2.php`) ” we have used the previous discussed `SymChar` class to more readily insert “#” characters in the X-axis label. We highlight some functionality that we haven’t yet discussed in detail

1. We use a utility class (`FuncGenerator` available in "`jpgraph_utils.inc.php`") to help create plot values from a mathematical expression. This will help create a set of (x,y) points that can later on be used as the base for a data series.
2. The x-axis labels and the position of the tick marks on the x-axis are manually positioned and specified to be positioned at "even" fractions of #.

Figure 8.8. Specifying manual ticks as fraction of Pi. (`manuallickex2.php`) [[example_src/manuallickex2.html](#)]



8.8. Character encoding

Note

If you are not using Japanese, Chinese, Cyrillic , Greek or Hebrew languages then this section can be safely skipped.

The core problem for the library is that it has no way of knowing in what input encoding the string given to the library is using. Hence it is necessary to, sometime, tell the library what input encoding is being used

in order for the library to do necessary character encoding conversion to generate UTF-8 (or UTF-16) as needed to properly render the TTF fonts. The specific encoding options for each major supported locale are explained below.

By default all JpGraph library files and examples are encoded in UTF-8

All defines mentioned below can be found in the file "jpgraph_ttf.inc.php"

8.8.1. Japanese encoding options

There is only one possible option that can be specified.

Table 8.4. Japanese encoding options

Symbolic define	Possible values	Description
ASSUME_EUCJP_ENCODING	true/false	Assumes that Japanese text have been entered in EUC-JP encoding. If this define is true then conversion from EUC-JP to UTF8 is done automatically in the library using the mbstring module in PHP. Note that the multibyte extension in PHP is not normally enabled.

Otherwise it is assumed that the input characters are encoded in UTF-8. Remember that to show the Japanese character sets (Kanji, Hiragana and Katakana) one of the Japanese font families (FF_MINCHO, FF_PMINCHO, FF_GOTHIC or FF_PGOTHIC) must be specified.

An example of using Japanese locale together with Windrose plots can be seen in Section 21.3.9, “Localizing the default names for the compass directions”.

8.8.2. Chinese encoding options

There are no specific settings that control the encoding. The following rules are used depending on the font is specified.

1. If the font is specified as FF_SIMSUN the built-in library conversion from GB2312 to UTF-8 will be used. This translation table is stored in the file jpgraph_gb2312.inc.php.
2. If the font is specified as FF_CHINESE then no conversion is made since it is assumed that the input character string is already in UTF-8. This only has the effect of changing the font to the default Chinese font family.
3. If the font is specified as FF_BIG5 then it is assumed that the input character string is encoded in BIG5 and the internal translation to UTF-8 is done by the iconv() function. This means that PHP must be built with iconv() support. By default this is not compiled into PHP (needs the "--width-iconv" when configured). For more on building PHP with the right options see Appendix I, *Compiling PHP*. If this method is not present the library will generate the following an error message.

An example of using Chinese encoding with Windrose plots can be seen in Figure 21.11, “Using chinese fonts (windrose_ex6.1.php)”

8.8.3. Cyrillic encoding options

In order to do proper translation to unicode from cyrillic the `LANGUAGE_CYRILLIC` define should be set to true. If you are running the library in multiuser environment it might be necessary to also adjust the `LANGUAGE_CHARSET` define as described below.

Table 8.5. Cyrillic encoding options

Symbolic define	Possible values	Description
<code>LANGUAGE_CYRILLIC</code>	true/false	Special unicode cyrillic language support
<code>CYRILLIC_FROM_WINDOWS</code>	true/false	If you are setting this config to true the conversion will assume that the input text is encoded in windows 1251, if false it will assume koi8-r
<code>LANGUAGE_CHARSET</code>	string	<p>This constant is used to auto-detect whether cyrillic conversion is really necessary if enabled. Just specify the encoding used, e.g. 'windows-1251', with a variable containing the input character encoding string of your application calling JpGraph.</p> <p>A typical such string would be 'UTF-8' or 'utf-8'. The comparison is case-insensitive. If this charset is not a 'koi8-r' or 'windows-1251' derivate then no conversion is done. This constant can be very important in multi-user multi-language environments where a cyrillic conversion could be needed for some cyrillic people and resulting in just erroneous conversions for non cyrillic language based people.</p> <p>Example: In the free project management software dotproject.net <code>\$locale_charset</code> is dynamically set by the language environment the user has chosen.</p> <p>Usage: <code>define('LANGUAGE_CHARSET', \$locale_charset);</code> where <code>\$locale_charset</code> is a GLOBAL (string) variable from the application including JpGraph.</p>

8.8.4. Hebrew encoding options

There are no user adjustable settings. The conversion is made from iso to unicode with the help of the PHP method "hebrev()" which is used to convert logical Hebrew text to visual text. This conversion is done automatically when the font is one of FF_DAVID, FF_MIRIAM or FF_AHRON

8.8.5. Greek encoding options

In order to do proper translation to unicode from greek the LANGUAGE_GREEK define should be specified to true.

Table 8.6. Greek encoding options

Symbolic define	Possible values	Description
LANGUAGE_GREEK	true/false	Special unicode greek language support
GREEK_FROM_WINDOWS	true/false	If you are setting this define to true the conversion of greek characters will assume that the input text is windows 1251

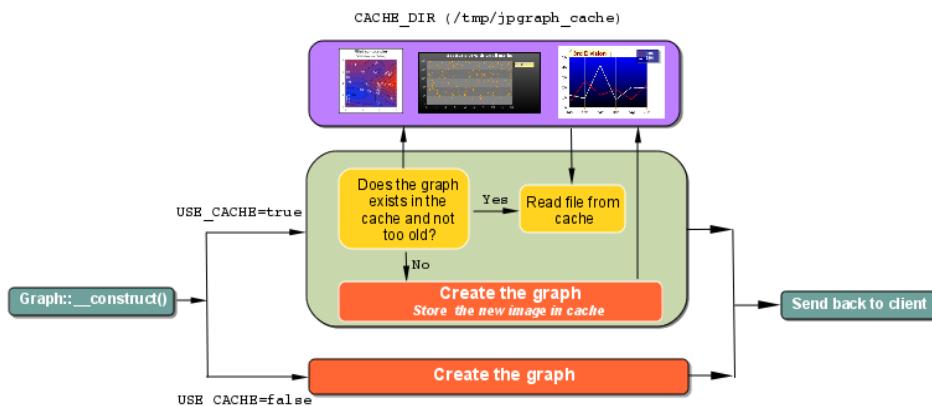
Chapter 9. Using the JpGraph cache system

To reduce load on the web server JpGraph implements an advanced caching system which avoids the burden of always having to run the full image script. The library supports two primary ways of significantly increase performance, using the built in cache system described in this section and the use of a PHP accelerator described in Chapter 11, *NuSphere PHP accelerator*

Depending on the complexity of the image script, for example if it is doing several DB lookups, the use of the library cache system (which will avoid running the graph scripts completely if possible) can make a for very drastic performance increase.

Figure 9.1, “Library cache principle” shows an overview of the cache system in the library.

Figure 9.1. Library cache principle



The rationale behind this is the observation that very few graphs are really real-time, i.e. needs to be updated absolutely every time the graphing script is called. For many graphs in a WEB-environment one can often get good precision by restricting the graphs to only be updated, say, a few times each day. Of course, if truly live data is what is needed then the cache system can not be used since then, by the nature of live data, the graph script must be called at each instance to get the latest available data, most probably from a database.

9.1. Enabling the library cache system

The enabling and disabling of the cache system is controlled by three defines in `jpc-config.php`

1. `DEFINE("USE_CACHE" , true)`
2. `DEFINE("READ_CACHE" , true)`
3. `DEFINE("CACHE_DIR" , "/tmp/jpgraph_cache/")`

The first of these, `USE_CACHE`, is the master-switch which must be set to true to enable the caching system.

The second switch, `READ_CACHE` very seldom needs to be changed. This second switch basically tells whether or not JpGraph should ever look in the cache. Setting this to false and the master-switch to true would then always generate a new updated image file in the cache and this new image would be send back

to the browser. The main use for this (admittedly) strange setting is if you like to have the side effect of the script that a fresh image is always stored in the cache directory.

The third define is not really a switch but a directory specification that tells the library what directory to use as the cache directory (where the cached images are stored).

The cache directory (CACHE_DIR) can be set to an arbitrary directory but the important thing to keep in mind is that the cache directory must be read and writable for the process running PHP.

Note

The directory name given should be an absolute directory path and **not** a file path relative to the document root.

9.2. Permission settings for the cache files

Note: This section is only applicable to a Unix derivate system which have the concepts of group and file ownership.

There are two additional settings that will allow the control of the group and file permission settings

1. `DEFINE('CACHE_FILE_GROUP' , 'www')`
2. `DEFINE('CACHE_FILE_MOD' , 0664)`

The CACHE_FILE_GROUP specifies what group should be set on the cached image file. If this is left empty then the group will be the same as the process running PHP.

The CACHE_FILE_MOD specifies the file permissions for the image file. If this is left empty the default permissions used by PHP will be set on the file.

When PHP is run from the command line (using the PHP CLI version) then the file permission and group will normally be set to the one of the user running PHP. Keep in mind that normally ordinary users are not allowed to change the group to 'www' (the default Apache2 group).

9.3. Using the cache in your script

The principle of the library cache is as follows when it is enabled.

1. The first time the graph script is called everything will be as usual, the script will run and in the end the script sends back the image to the browser. However if the caching is enabled JpGraph will automatically have stored a copy of the generated image in the cache directory.
2. When the graph script is executed the next time it checks to see if an image corresponding to this graph script has already been generated and is available in the cache directory.
3. If the image is available in the cache directory the library check to see how old the image is. If the image is older than a specified limit than it assumes that the image is out dated and runs the graph script as usual and makes sure the newly generated image is stored in the cache directory. Hence replacing the outdated image.
4. If the image in the cache directory was current (i.e. not too old) it is read and send back to the clients (e.g. Web-browser) without the rest of the graph script being executed.

From the above description there are a couple of parameters that should be specified, the name to use when the image is stored and the timeout value when the image is considered too old, i.e. how long was it since the image was generated.

The first parameter, the filename, can be either manually specified or the library can create a filename based on the name of the graph script.

Both these parameters are specified in the initial `Graph()` call where a new graph instance is created. A basic example of this is shown in Example 9.1, “Using an automatic cache filename and a 60min timeout of the cached images.”.

Example 9.1. Using an automatic cache filename and a 60min timeout of the cached images.

```
<?php  
// ... includes  
  
$graph = new Graph($width, $height, 'auto', 60);  
  
// ... rest of the graph script  
  
$graph->Stroke();  
?>
```

The code in Example 9.1, “Using an automatic cache filename and a 60min timeout of the cached images.”, will use an automatic filename for the cached image and make the image valid for 60 minutes. This means that if the script is called again, within 60minutes, it will return the image just after the initial `Graph()` call and not execute any more lines of code in the script.

For basic usage this is all that is necessary, enable the cache in the settings and supply a filename and a timeout value. The rest of the logic is handled by the library.

Tip

If you want the timeout value to be "forever" then you can specify a "0" as the timeout value (or leave the parameter blank). To regenerate the image you will have to manually remove the image files from the cache. This removal could for example be handled by a nightly cron-job.

There is however one caveat which must be understood when using the above construction. The image/graph store in the cached file will be returned to the browser as a side effect of the initial `$graph = new Graph()`. This also means that:

No lines after the initial `Graph()` call will be executed in the image script in case the image exists in the cache directory.

This is the expected behaviour since this means that no unnecessary code will be executed in the graph script in case the image has been found in the image cache.

However, for the case where some more control of exactly how a cached image is sent back it is necessary to add some complexity by doing things less automatically. This gives greater control but also is slightly more complex and is described in the next section.

9.3.1. Manually controlling the cached image

Note

These utility functions were added in **v3.0.5** of the library. It is still possible to do this in previous versions but then some more code is needed to duplicate what these methods does. If this feature is wanted then it is strongly advised to upgrade to this or later version.

There are two parts to doing this manually.

1. Check if the cached image exists in the cache and is valid (i.e. not too old)
2. Stream the cached image file back to the browser in that case

If the cached image is not valid then we just need to construct the graph as usual and it will be stored in the cache automatically.

The following code example shows how this is done in principle in a graph script where we use automatic naming i.e. the cached file name will get a name based on the script name. This is done with the library utility function `GenImgName()` which constructs a suitable image name from the script name and with a proper image compression format (i.e png, jpg or gif).

```
<?php
$width = ...;
$height = ...;
$cachefilename = GenImgName();
$graph = new Graph($width,$height);

// Check if the cache file exists and is valid
$valid = $graph->cache->IsValid($cachefilename);

if( $valid ) {
    // The cached file is valid and we can now do any necessary
    // processing and then send it back to the client
    doSomeProcessingIfNecessary();

    $graph->cache->StreamImgFile($graph->img,$cachefilename);

} else {

    // The cache file is not valid or does not exists so we
    // must construct the graph as normal

    // Tell the graph that we want to cache this image
    $timeout = ...;
    $graph->SetupCache($cachefilename,$timeout);

    // The remainder of a normal graph script

    ...

    // .. and send back the image as usual (this will also store
    // a copy of the image in the cache directory)
    $graph->Stroke();
}
```

9.4. Using the cache with Client Side Image Maps (CSIM)

(See Chapter 10, *Using CSIM (Client side image maps)* for a full description on the usage of CSIM together with the library)

You can also use the cache system for images that uses image maps (CSIM) as well. The cache system interface is slightly different in this case since the cache needs to store both the cached image and the cached image-map. It also needs to change due to the added complexity with CSIM scripts not sending back an image but rather a pre-formatted HTML page. The two major differences from the non-CSIM cache is:

1. The cached version will **not** be stored in the previous defined cache directory. It uses a different cache directory.
2. The script that want to make use of the cache system together with CSIM must call an extra method, `CheckCSIMCache()`, to check the cache.

Note

Please remember that when using CSIM you must end your script with a call to `Graph::StrokeCSIM()` method instead of the `Graph::Stroke()` used for non-CSIM.

To use the cache with CSIM you have to call the `Graph::CheckCSIMCache()`. As with the caching for non-CSIM you have to supply a name to be used for the cached version (it can be manually or automatic) as well as an optional timeout value. The default timeout value if nothing else is specified is 60 minutes.

The name argument requires some more explanations. The file name must be specified as a relative name.

For example "myimage" or perhaps "firstpage/image3".

The reason for this is that a script that uses CSIM does not send back an image. Instead it sends back a HTML page which includes the image map coordinates and has an included "" tag which references the image script recursively in normal cases but if the image is found in the cache this reference will be to a static image instead.

Depending on the installation of the library this will now end up in the directory specified in the `CSIMCACHE_DIR` define. This must also be a directory accessible by the normal HTTP/PHP process. By default, if nothing else is specified, a directory called "csimcache" will be created in the same directory as the image script itself.

The default value has the drawback that the directory from where the script is executed must also be writable by the process running PHP. Best practice here is to keep the number of writable directory for PHP down to a minimum and re-use the same directory as is used for the standard cache.

This however, requires that the PHP and HTTP setup is such that the cache directory is also accessible by the HTTP server from the htdocs root.

The `CheckCSIMCache()` method checks the cache for an existing cached version and if found it returns it and halts execution of the script. So, this call should be the first call after the creation of the core graph instance, i.e. `$graph = Graph($width,$height)` and before any heavy work is done to create the image so that no unnecessary work is done in case a cached image + image map is found.

The general structure of a script that uses CSIM and cache should follow the template shown in

Example 9.2. General structure for a CSIM script that uses CSIM

```
<?php
require_once( 'jpgraph.php' );

// ... any necessary includes

$width = ...
$height = ...
$graph = new Graph($width,$height);

// Check cache and use a 10 min timeout
$graph->CheckCSIMCache('csim_image1',10);

// If a cached version exists, execution halts here
// and the cached HTML image map is sent back

// ... construct and format the graph

// ... finally send back the image map HTML script
$graph->StrokeCSIM();
?>
```

Please note that it is not necessary to pass any argument to the final call of StrokeCSIM()

Note

The CSIM caching works by storing two files in the cache directory. One file being the image and the other file being the corresponding image map as a pure HTML file. The HTML file will reference the static image.

Chapter 10. Using CSIM (Client side image maps)

Image maps makes it possible to create images with "active" areas that will react for a mouse-click. It is then up to the designer to decide what actions should be taken. Image map is often used to create drill-down charts where it is possible to dynamically zoom into an image. Image maps is part of the HTML standard.

There are actually two types of images maps, client and server-side. This refers to where the actual processing of the image click happens. Without doubt the best (and most commonly used) type is the client side. This is also what the library supports. In the remainder of this manual this will be referred to as **CSIM, Client Side Image Maps**.

10.1. The principles

Image maps works so that each hotspot area in the graph that should be used must have an associated URL. When the user clicks somewhere in that particular hotspot area the browser will open the specified URL. Typical hotspot areas in the graphs are

1. Texts, for example titles
2. Markers in line graphs
3. Slices in pie graphs
4. Legends
5. Bars in barplots
6. etc.

The way the CSIM HTML standard works is that the HTML page must have a section with coordinates that defines the various hotspots together with the associated URL that should be called. Each section of coordinates are connected to a specific image that is included with a standard by a common id. This will now add some complexity since the library must return an HTML page for the coordinates and not image data as normal for the library. How this is done is the topic of the next section.

A number of examples of CSIM graphs are included in the Examples/ directory. Some of the available examples are listed in Table 10.1, “CSIM Examples (in Examples/ directory)”

Table 10.1. CSIM Examples (in Examples/ directory)

CSIM Examples (in Examples/ directory)

bar_csimex1.php	bar_csimex2.php
bar_csimex3.php	barline_csimex1.php
barlinefreq_csimex1.php	boxstockcsimex1.php
ganttcsimex01.php	ganttcsimex02.php
imgmarkercsimex1.php	pie3d_csimex1.php
piec_csimex1.php	pie_csimex1.php
scatter_csimex1.php	titlecsimex01.php

In order to easily access all of these examples it is possible to call the `testsuit.php` example with an additional argument "`t=2`". By following the link "`testsuit.php?t=2`" a separate window will open with all the possible CSIM examples.

10.2. The basic structure of an image map script

The basic structure for a HTML page using client side image maps will have the following layout

```
// Image map specification with name "mapname"
<MAP NAME=...>
... specification ...
</MAP>

// Image tag

```

This poses an interesting question. Since we normally call the graphing script directly in the `` tag how do we get hold of the image map (which is available only in the image script) in this "HTML wrapper" script?

In JpGraph there is actually two ways of solving this.

1. Use the preferred "builtin" way using the modified `Stroke()` method `StrokeCSIM()` instead of the standard `Graph::Stroke()` method.
2. Directly use the `Graph::GetHTMLImageMap()` which gives you fine control at the expense of more complex coding. This is necessary if several image map graphs are needed on the same page.

The first (and preferred) way modifies the stroke method so that instead of returning an image (like the standard `Stroke()` method) `StrokeCSIM()` actually returns an HTML page containing both the image map specification and the correct `` tag.

This means that it is necessary to treat an image map returning image script differently from a non-CSIM image script, for example it is not possible to use it directly as the target for the "`src`" attribute of the `` tag since it sends back a HTML page containing both an image tag together with an image map.

10.3. Specifying targets for image map plots

To turn a standard image script into a CSIM script the first thing needed to do is to supply the appropriate URL targets for the hotspots in the image. What the hotspots represent depends on the type of plot. CSIM is supported by all graph types.

To specify a URI link for each hotspot the `SetCSIMTargets()` method is used on the graph object that should be given a hotspot.

There are two arguments to this method

1. `$aTargets`, an array of valid URL targets. One URL per hot spot, for example if you have a 10 values bar plot you need 10 URLs. If the `SetCSIMTarget()` is applied to, for example, a text then only one URL target should be specified.
2. `$aAlts`, an array of valid alt-texts. Many browsers (but not all) will show this text string if the mouse hovers over a hotspot.

10.3.1. Creating popup-windows as targets for CSIM

URL targets specified will be opened by the browser as usual, i.e. it will replace the current HTML page. Another common usage pattern is to open a popup window, perhaps with a zoomed version of a graph. This can be done by adding some javascript in the target URL. If the URL, for example, is specified as

```
$target = "...";
$targetURL = "javascript:window.open('$target?id=%d', '_new', 'width=500,height=300'"
```

clicking on a target will now open a separate window with the specified width and height. The \$target variable must be set to the wanted URL.

10.4. Using StrokeCSIM()

The simplest way of creating a creating a CSIM image is with the `StrokeCSIM()` method. As mentioned before this method actually returns a (small) HTML page containing both the image-tag as well as the image map specification. Hence it is not possible to use a script that ends with this method in a standard image-tags src property.

There are two ways to create CSIM (or get hold of) the image maps

1. Use the CSIM image script as the target in a standard anchor reference, for example

```
<a href="mycsimscript.html">
```

This has the drawback that the image page will only contain the image and nothing else.

2. The other way will allow the image script to be included in an arbitrary HTML page by just including the image script at the wanted place in the HTML page using any of the standard "include" php statement. For example

```
<h2> This is an CSIM image </h2>
<?php include "mycsimscript.php" ?>
```

Note: If there are several CSIM images on the same page it is necessary to use "include_once" in the scripts for the inclusion of "jpgraph.php" and the other jpgraph library files since the files will be included multiple times on the same page and one or more "*Already defined error*" will be displayed.

The process to replace `Stroke()` with `StrokeCSIM()` is strait forward. Replace all existing calls to `Stroke()` with the equivalent calls to `StrokeCSIM()`.

10.4.1. Optional argument to StrokeCSIM()

Once difference compared with `Stroke()` is that if a filename is supplied to the `StrokeCSIM()` method it does not specify a file to write an image to as is the case with `Stroke()`. The signature for the method is

```
StrokeCSIM($aScriptName='auto', $aCSIMName='', $aBorder=0)
```

The first (optional) argument is the name of the actual image script file including the extension. So for example if the image script is called "mycsimscript.php" the call should be

```
$graph -> StrokeCSIM( 'mycsimscript.php' )
```

By using the predefined name 'auto'. This will be done automatically.

Note

Why does the script name need to be used as the first parameter? The reason is that in the creation of the HTML page which is sent back we need to refer to the script in the image tag. In older versions of PHP there was no 100% way of getting hold of the actual script name in all circumstances and it was then necessary to specify it here. In PHP5 (and newer version of PHP4) this is no longer a problem but this parameter is still kept for backward compatibility.

The other arguments to `StrokeCSIM()` are optional as well. The second argument specifies the id that connects the map with the corresponding image. Please note that if several CSIM images are used in the same HTML page it is necessary to specify the image map name as the second parameter since all image maps must be unique to properly match each image map against each image. Please consult the class reference `StrokeCSIM()` for more details.

The final argument specifies whether the image should have border or not.

10.4.2. How does `StrokeCSIM()` work?

Knowledge of the exact technical details of the way `StrokeCSIM()` works is probably not needed by many people but for completeness we outline those details in this short section.

The fundamental issue we have to solve is that we must be able to call the image script in two modes. When the user includes the image script the `StrokeCSIM()` method should return the HTML page but when the image script is later called directly in the image tag it must not return an HTML page but rather the actual image.

The way this is solved is by using one GET argument which is passed on automatically when we use the image script name in the ``-tag.

A look at the generated HTML from `StrokeCSIM()` will make this clear. In Figure 10.1, “Example of generated HTML code for `StrokeCSIM()`” the resulting HTML code after running the example script `bar_csimex1.php` is shown. The argument to the `src`-property of the image tag is not simply the script name but the script name with an additional argument, `?_jpg_csimd=1`. This argument is passed on to the library which then will run the script again but this time only generate the image and not the HTML. (In the JpGraph internal code this pre-defined argument is checked for and if it exists the image is sent back and not the HTML page.)

Note

The actual string name of this parameter is defined by a `DEFINE` statement in `JpGraph,_CSIM_DISPLAY`.

Figure 10.1. Example of generated HTML code for `StrokeCSIM()`

```
<map name="__mapname1828__" id="__mapname1828__" >
<area shape="poly" coords="74, 210, 74, 165, 92, 165, 92, 210" href="bar_clsmex1.php?_jpg_csimd=1" ismap="ismap" usemap="#__mapname1828__" bo
<area shape="poly" coords="118, 210, 118, 112, 136, 112, 136, 210" href="bar_clsmex1.php?_jpg_csimd=1" ismap="ismap" usemap="#__mapname1828__" bo
<area shape="poly" coords="162, 210, 162, 176, 180, 176, 180, 210" href="bar_clsmex1.php?_jpg_csimd=1" ismap="ismap" usemap="#__mapname1828__" bo
<area shape="poly" coords="206, 210, 206, 146, 224, 146, 224, 210" href="bar_clsmex1.php?_jpg_csimd=1" ismap="ismap" usemap="#__mapname1828__" bo
<area shape="poly" coords="250, 210, 250, 93, 268, 93, 268, 210" href="bar_clsmex1.php?_jpg_csimd=1" ismap="ismap" usemap="#__mapname1828__" bo
</map>

```

Note

As the astute reader has come to realize CSIM scripts has a performance impact in that each script has to be run twice. Once to get hold of all the coordinates and generate the image and a second time via the `` tag in generated HTML. The library is intelligent enough to minimize the code to run so that it only runs what is absolutely necessary. This means that roughly 75% has to be run which yields a total overhead of around 1.75 times when generating a CSIM image.

10.4.3. Image maps and the cache system

For version 1.9 and later the cache system has been extended to include the CSIM maps as well. For each CSIM graph two files are stored in the cache, the image file itself as well as the wrapper HTML with the actual image map.

10.5. Getting hold of the image map

There are at least two cases where the basic `StrokeCSIM()` method will not work. Basically this is limited to only showing the graph in one HTML page and nothing more. So the cases where this needs to be handled differently are

1. In the case where you want to store the image on disk and later use it in an img-tag you need to get hold of the image map.
2. In order to include multiple CSIM images on a WEB-page. (This is not entirely true though, it is possible to include several CSIM graph images with the use of the `<iframe>` tag. This in effect creates its own WEB page within the WEB page but we will not discuss this further here.)

To get hold of the image map the function `Graph::GetHTMLImageMap()` should be used. This returns the coordinates for the hotspots

An example of the use of this is shown below. With these lines the image will be written to a file. The script then returns a HTML page which contains the Client side image map and an img-tag which will retrieve the previously stored file.

```
$graph -> Stroke ( "/usr/local/httpd/htdocs/img/image001.png" );
echo $graph -> GetHTMLImageMap ( "myimagemap001" );
echo "<img src=\"img/image001.png\" ISMAP USEMAP=\"#myimagemap001\" border=0>" ;
```

10.6. Mixing several CSIM images in an HTML page with text

As was mentioned above using `StrokeCSIM()` is very simple but it has the drawback that it returns a single HTML page without any possibilities to add additional text which makes its use fairly limited in real life situation where the graph is part of a complex WEB-page. In this section we will discuss some best practice to do this.

In principle there are two ways to do this

1. Store both the image map and the image in files which are later read back in the HTML page. This has the advantage of being simple but the drawback that it increases the processing time since writing and reading from a file takes time.

2. Avoiding the use of temporary file by mimicking the way `StrokeCSIM()` works and do the steps `StrokeCSIM()` does internally but in the script directly. In the remainder of this section we will show how this can be setup. That part will also introduce the method `StrokeCSIMImage()` which is key to include CSIM graphs in HTML pages.

Caution

Some of the described methods in this section was added in version 2.5 of the library. In earlier versions more of this has to be done manually.

10.6.1. Adding one CSIM graph in a HTML page

The library has been designed to make this as painless as possible but due to the way CSIM works there are some manual work that cannot be avoided. We will start slowly and in detail walk through an example where we include one CSIM graph in an arbitrary HTML page. There are a few things to keep in mind, the rest will be taken care of automatically by the library

1. In the HTML page the CSIM map must be semi-manually included. (It doesn't matter where)
2. The `` tag for rendering the image must be semi-manually created
3. The original graph script needs a minor augmentation since it should no longer end by calling `StrokeCSIM()`
4. Some care needs to be taken to specify what the URL:s to be called should be

The library provides suitable functions for step 1 & 2 above so the only thing that needs to be done in the HTML page is calling these functions at suitable places. In principle the HTML page should have the structure shown in Example 10.1, “Principles of including CSIM graph in a HTML page”

Example 10.1. Principles of including CSIM graph in a HTML page

```

<html>
<body>
<?php
// The name of the graph script file (change as needed!)
$_graphfilename = 'mycsimgraph.php';

// This is the filename of this HTML file
global $_wrapperfilename;
$_wrapperfilename = basename (__FILE__);

// Create a random mapname used to connect the image map with the image
$_mapname = '__mapname'.rand(0,1000000). '__';

// This is the actual graph script
require_once ($_graphfilename);

// Get hold of the image map to include it in the HTML page
$imgmap = $graph->GetHTMLImageMap($_mapname);
echo $imgmap;
?>

<p>Some arbitrary HTML text .... </p>

<?php
// We now create the <img> tag
$imgtag = $graph->GetCSIMImgHTML($_mapname,$_graphfilename);
echo $imgtag;
?>

</body>
</html>

```

Before we discuss the HTML code in detail lets first show what augmentation is needed in the graph script.

Normally the graph script will end with a call to either `Graph::Stroke()` or `Graph::StrokeCSIM()`. However, with CSIM style graph the complication is that we need to call the script twice. Once to get the image map and once (in the final `` tag) to actually generate the image. In order to separate these two cases we make use of a URL argument which will be used as a flag so that we know how to behave in the graph script. In conjunction with HTML skeleton shown in Example 10.1, “Principles of including CSIM graph in a HTML page” we need to change the graph script so that it instead uses the method `Graph::StrokeCSIMImage()` so that the last line will be changed to

```

...
$graph->StrokeCSIMImage();
?>

```

This method will only stroke the image when the “secret” flag is passed as a URL argument (which will be added automatically when the `` tag is constructed in the call to `GetCSIMImgHTML()`). This means that the first time this function gets called when we do the initial `require_once()` in the top of the HTML page this method will do nothing, which is exactly what we want since we only want to include the graph script in order to be able to do the call to `GetHTMLImageMap()` later down in the script.

We are now in position to discuss the HTML script above.

Line 1-2 This is the standard HTML opening tags (by choice we keep this very simple and sloppy in these example.)

Line 3-20 This is where we include the graph script. In addition we must also create a map name that will be used to connect an image map with the `` tag. We have chosen to create a random name since the actual name is not significant. The only criteria is that the name must be unique if there are multiple maps in the same HTML page.

In addition we also record the name of the actual HTML page in the variable "`$wrap-perfilename`". This is so we can potentially use it as a target in the image script. We could then have targets that redisplays the same page but with potential additional or changed URL argument.

Line 21-22 This is just an illustration that it is possible to add arbitrary HTML markups and text on the page.

Line 23-27 This is where we generate the needed `` tag that should be included in the page. It is illustrative to view how the `` tag actually looks.

```
SetCSIMTargets($targets,$altnames,$wintargets)
```

Case 2: Opening the same page but with some different URL arguments.

To do this we make use of the name of the HTML wrapper file we stored in `$_wrapperfilename`. We can then construct the targets as shown in Example 10.2, “Creating CSIM URL targets to open in same browser window”

Example 10.2. Creating CSIM URL targets to open in same browser window

```
<?php
...
global $_wrapperfilename;
$n = ... ; // Number of bars
$targ = array(), $alt = array(); $wtarg = array();
for( $i=0; $i < $n; ++$i ) {
    $urlarg = urlencode( ... );
    $targ[] = $_wrapperfilename.'?'.$urlarg;
    $alt[] = 'val=%d';
    $wtarg[] = '';
}
$bplot->SetCSIMTargets($targ,$alt,$wtarg);
...
?>
```

Caution

Remember to not include the "&" or the "=" used when constructing the URL argument in the call to `urlencode()`. Otherwise they will become part of the data and not separators in the URL argument string.

Case 3: Open a fresh page in the browser

The following example opens the plain browser script in a fresh window. In order to get hold of the script name we use the predefined PHP constant `__FILE__`. The target can of course be changed to any URL.

Example 10.3. Creating CSIM URL targets to open in a fresh window

```
<?php
...
$n = ... ; // Number of bars
$targ = array(), $alt = array(); $wtarg = array();
for( $i=0; $i < $n; ++$i ) {
    $urlarg = urlencode( ... );
    $targ[] = __FILE__.'?'.$urlarg;
    $alt[] = 'val=%d';
    $wtarg[] = '_blank';
}
$bplot->SetCSIMTargets($targ,$alt,$wtarg);
...
?>
```

Case 4: Open in an existing window/frame

By modifying the \$wtarg[] line in the example above to

```
<?php
$wtarg[ ] = '';
?>
```

The target will open in the existing window.

In the "Example/" directory you can find the above a fully working script as "csim_in_html_ex1.php" (HTML script) and "csim_in_html_graph_ex1.php" (Graph script).

10.6.2. Adding multiple CSIM graphs in a HTML page

Having laid the foundation for inclusion of CSIM graphs in Section 10.6.1, “Adding one CSIM graph in a HTML page” it is now a simple exercise to extend this to include multiple CSIM graphs in the same HTML page. The only modifications we have to do is to make sure that:

1. Each image map has a unique name
2. The graph scripts must create unique instances of the main Graph class, i.e. they cannot both have an instance called "\$graph"
3. Include each graph script in turn and get the corresponding HTML map
4. Get the proper image tag for each graph

In Example 10.4, “Example of HTML page that includes two Graph CSIM scripts (“Examples/csim_in_html_ex2.html”)” we show a complete HTML script that includes two graphs, one bar (the same as in the previous example) and one Pie graph. For illustrative purposes we use class PieGraphC variant which is a Pie graph with a circular middle. The result of calling this HTML page is shown in ??

```

}
else {
    echo '<b style="color:darkred;">Clicked on pie slice: '.$_GET['pie_clickedon'];
}
echo '<p />';
?>

<p>First we need to get hold of the image maps and include them in the HTML
page.</p>
<p>For these graphs the maps are:</p>
<?php
// The we display the image map as well
echo '<small><pre>'.htmlentities($imgmap1).'</pre></small>';
?>
<p>
and
</p>
<?php
// The we display the image map as well
echo '<small><pre>'.htmlentities($imgmap2).'</pre></small>';
?>

<?php
// Construct the <img> tags for Figure 1 & 2 and rebuild the URL arguments
$imgtag1 = $graph->GetCSIMImgHTML($_mapname1,$_graphfilename1);
$imgtag2 = $piegraph->GetCSIMImgHTML($_mapname2,$_graphfilename2);
?>
<p>The graphs are then displayed as shown in figure 1 & 2. With the following
created <img> tags:</p>
<small><pre>
<?php
echo htmlentities($imgtag1);
echo htmlentities($imgtag2);
?>
</pre></small>

<p>
Note: For the Pie the center is counted as the first slice.
</p>

<p>
<table border=0>
<tr><td valign="bottom">
<?php
echo $imgtag1;
?>
<br><b>Figure 1. </b>The included Bar CSIM graph.
</p>
</td>
<td valign="bottom">
<?php
echo $imgtag2;
?>
<br><b>Figure 2. </b>The included Pie CSIM graph.
</p>
</td>
</tr>
</table>
</body>
</html>

```

Figure 10.2. Browser window after calling HTML page in Example 10.4, “Example of HTML page that includes two Graph CSIM scripts (“Examples/csime_in_html_ex2.html”)” (Note: The image has been scaled down to better fit this manual.)

This is an example page with CSIM graphs with arbitrary HTML text

Clicked on bar: <none>

Clicked on pie slice: <none>

First we need to get hold of the image maps and include them in the HTML page.

For these graphs the maps are:

```
<map name="__mapname986680__" id="__mapname986680__">
<area shape="poly" coords="135,16,135,3,268,3,268,16" href="#45" alt="Title for Bar" title="Title for Bar" target="_blank" />
<area shape="poly" coords="67, 210, 67, 145, 89, 145, 89, 210" href="csim_in_html_ex2.php?clickedon=1" title="val=12" alt="val=12" />
<area shape="poly" coords="121, 210, 121, 70, 143, 70, 143, 210" href="csim_in_html_ex2.php?clickedon=2" title="val=26" alt="val=26" />
<area shape="poly" coords="175, 210, 175, 161, 197, 161, 197, 210" href="csim_in_html_ex2.php?clickedon=3" title="val=9" alt="val=9" />
<area shape="poly" coords="229, 210, 229, 118, 251, 118, 251, 210" href="csim_in_html_ex2.php?clickedon=4" title="val=17" alt="val=17" />
<area shape="poly" coords="283, 210, 283, 43, 305, 43, 305, 210" href="csim_in_html_ex2.php?clickedon=5" title="val=31" alt="val=31" />
</map>
```

and

```
<map name="__mapname157617__" id="__mapname157617__">
<area shape="poly" coords="253, 197, 260, 187, 264, 174, 265, 165, 316, 165, 312, 190, 303, 214, 292, 230, 253, 197" href="csim_in_html_ex2.php?pie_clickedon=1" alt="Clicked on pie slice" />
<area shape="poly" coords="194, 224, 207, 225, 220, 222, 231, 217, 241, 209, 244, 206, 283, 239, 264, 256, 242, 269, 217, 275, 192, 275, 184, 274, 194, 208, 212, 171, 219, 180, 221, 170, 271, 146, 263, 125, 249, 107, 231, 98, 214, 144, 193" href="csim_in_html_ex2.php?pie_clickedon=2" alt="Clicked on pie slice" />
<area shape="poly" coords="138, 138, 134, 150, 134, 163, 136, 175, 138, 181, 92, 202, 84, 178, 83, 153, 88, 128, 92, 117, 138, 138" href="csim_in_html_ex2.php?pie_clickedon=3" alt="Clicked on pie slice" />
<area shape="poly" coords="182, 97, 170, 101, 159, 107, 150, 116, 144, 126, 98, 105, 111, 83, 130, 66, 152, 53, 174, 47, 182, 97" href="csim_in_html_ex2.php?pie_clickedon=4" alt="Clicked on pie slice" />
<area shape="poly" coords="260, 148, 259, 135, 254, 124, 247, 113, 237, 105, 225, 100, 213, 97, 201, 98, 192, 47, 218, 46, 243, 52, 266, 63, 285, 80, 2" href="csim_in_html_ex2.php?pie_clickedon=5" alt="Clicked on pie slice" />
<area shape="circle" coords="200,160,51" href="csim_in_html_ex2.php?pie_clickedon=6" title="val=$d" alt="val=$d" />
</map>
```

The graphs are then displayed as shown in figure 1 & 2. With the following created tags:

```


```

Note: For the Pie the center is counted as the first slice.

CSIM Center Pie plot



Figure 2. The included Pie CSIM graph.

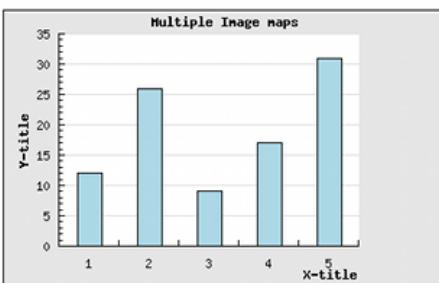


Figure 1. The included Bar CSIM graph.

Chapter 11. NuSphere PHP accelerator

Caution

The NuSphere encoded variant of the library is only included in the JpGraph pro version.

11.1. Introduction and purpose

One of the best way to increase the performance of large and complex PHP scripts is to install one of the available PHP accelerators. JpGraph supports *NuSphere PhPEXpress* accelerator. In order to take advantage of this accelerator you must install the version of the library that is encoded for use with the accelerator. This version is included in the Pro-version of the library under the directory `src-encoded/`.

If you have the Pro-version of the library there is really no good reason not to use the accelerated version of the library. It will reduce the load on your server as well as significantly decrease the run time for graph scripts. In addition it will also reduce some of the memory requirements needed since the parsing process (translating PHP to bytecode) for the library can be avoided. Furthermore *NuSphere PhPEXpress* also implements a caching mechanism which means that often executed scripts will be kept in memory to avoid re-reading them from disk each time they are needed. This applies to both encoded and non-encoded script files.

11.2. Installing PhPEXpress

NuSphere PhPEXpress is a regular PHP extension, which makes it easy to install and deploy. You can download PhPEXpress freely from

<http://www.nusphere.com/products/phpxpress.htm>

To install *NuSphere PhPEXpress* do the following:

1. Open your `php.ini` file for editing
2. Add line `zend_extension_ts=c:\full\path\to\phpxpress-php-x.x.dll` if you are deploying on Windows, or `zend_extension=/full/path/to/phpxpress-php-x.x.so` if you are deploying on Unix, Linux or Mac OS operating systems
3. Copy `phpxpress-php-x.x.dll` or `phpxpress-php-x.x.so` in the PHP extensions directory specified in `php.ini` file.
4. Stop and Start Apache if you are running PHP as Apache module.
5. [Optional] Execute call to `phpinfo()` function and make sure that PhPEXpress is properly installed. Once PhPEXpress is installed on the server, you can execute PHP Scripts encoded with Nu-Coder as well as regular, not encoded PHP scripts. In both cases you will gain an improved performance in the execution of the scripts.

Part III. Common features

This part will describe the common features for all graphs as well as explaining the structure of a typical graph script. After reading through this part the user should have a basic understanding how to create simple graph scripts and understand what core settings and naming conventions are used.

Chapter 12. Commonalities for all graphs

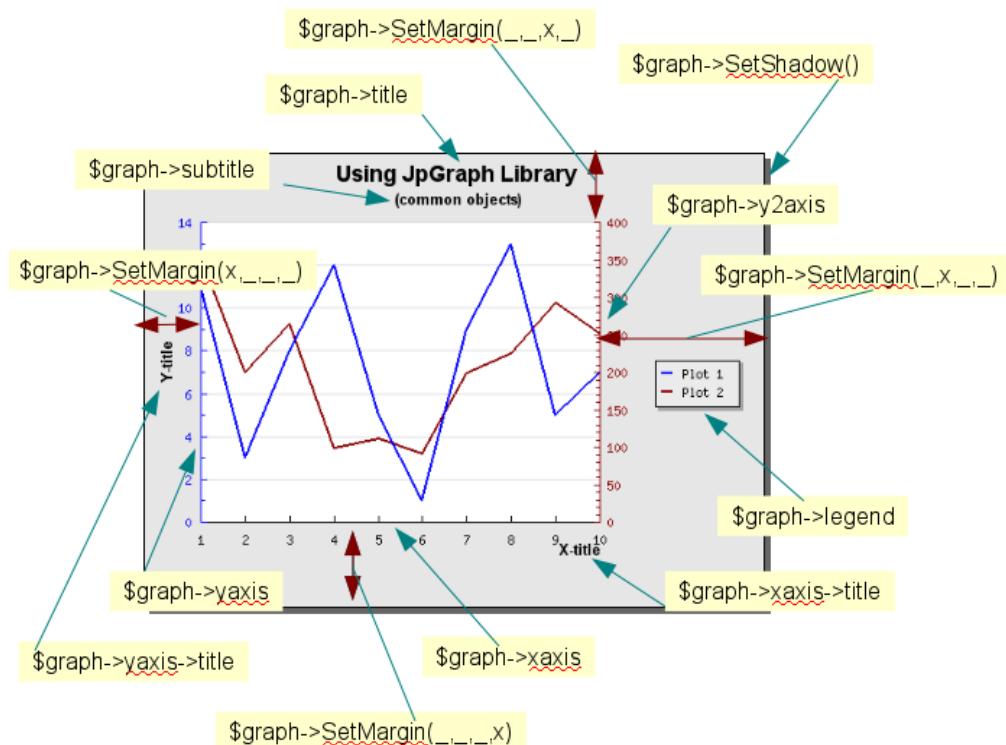
All graph scripts follow, to some extent, the same structure. All scripts must start by creating an instance of the Graph class. The Graph class represent the entire graph and can have one or more plots build up from the data series. Almost all methods that are used to control the appearance of the graph are methods of this class.

In order to give a feel for common objects we show examples of two of the most commonly used graph types, cartesian graphs and pie graphs in the following two sections.

12.1. Common objects for cartesian graphs (x-, y-graphs)

In all documentation of the library the convention is that the instance of the Graph class will be stored in a variable named "\$graph". Most of the parts that make up a graph are available as instance variables of the Graph class. In order to illustrate some of the commonly used instance variables Figure 12.1, "Commonly used objects in a graph (common-obj-graph.php)" shows a basic graph with a number of the objects that can be manipulated in the graph scripts.

Figure 12.1. Commonly used objects in a graph (common-obj-graph.php [example_src/common-obj-graph.html])



We note a few things in Figure 12.1, "Commonly used objects in a graph (common-obj-graph.php)"

- All graphs will start by including the necessary library files via one or more "require_once" PHP statements
- All graph scripts (for 2D linear type graphs) will have the following two method calls

```
$graph = new Graph($width, $height);  
$graph->SetScale('...');
```

These two calls create the necessary instance of the core Graph class that represents the entire graph and specifies what scale should be used for the y- and x-axis. In Figure 12.1, “Commonly used objects in a graph (common-obj-graph.php)” we are also using a second -axis so in this case the script will also contain a call to

```
$graph->SetY2Scale('...');
```

- Most graphs will also adjust the left, right, top and bottom margin. In Figure 12.1, “Commonly used objects in a graph (common-obj-graph.php)” the margins are shown with red arrows. The margins are specified with the method

```
$graph->SetMargin($left,$right,$top,$bottom);
```

- All text objects (e.g. graph and axis titles) are instances of the common Text class. This means that the text, font, color are specified in the same way. For example the code to set the title and subtitle in the graph above is

```
$graph->title->SetFont(FF_ARIAL, FS_BOLD, 14);  
$graph->title->Set("Using JpGraph Library");  
$graph->title->SetMargin(10);  
  
$graph->subtitle->SetFont(FF_ARIAL, FS_BOLD, 10);  
$graph->subtitle->Set('(common objects)');
```

In a similar way the code to set the titles of the axis are

```
$graph->xaxis->title->SetFont(FF_ARIAL, FS_BOLD, 10);  
$graph->xaxis->title->Set("X-title");  
  
$graph->yaxis->title->SetFont(FF_ARIAL, FS_BOLD, 10);  
$graph->yaxis->title->Set("Y-title");
```

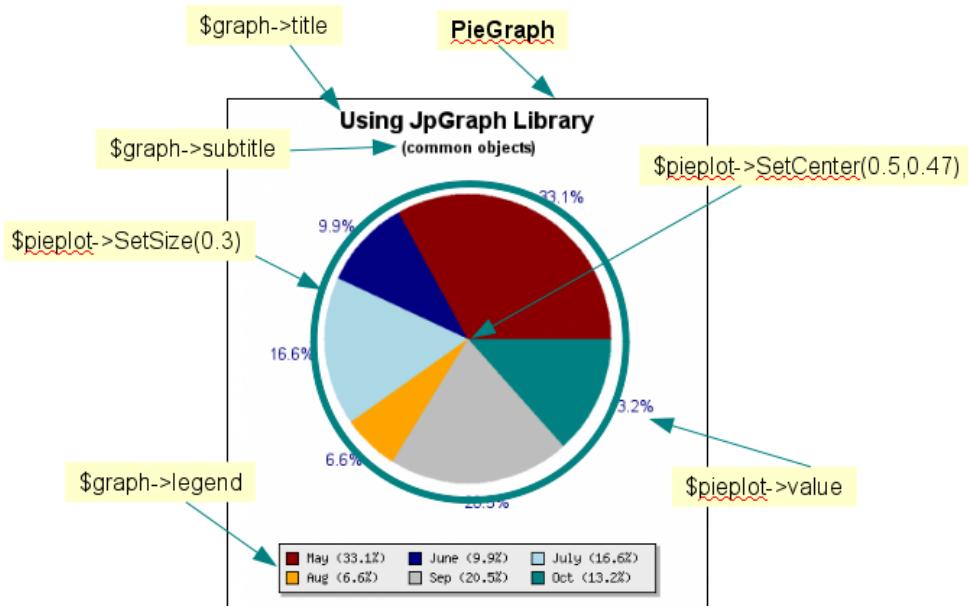
- The graph uses a legend in two rows to name each data series. The position of the legend is controlled by the script and in Figure 12.1, “Commonly used objects in a graph (common-obj-graph.php)” we placed it on the right side. It is also possible to in detail to adjust the layout of the names in the data series. For example we could change the layout of the legend to position all the names of the data series in one row as opposed to one column which is the default. The number of columns and/or rows is user settable.
- Finally all graphs will end with a call to `Stroke()` or one of its variants (`StrokeCSIM()`, `StrokeCSIMImage()`). As was described in Part II this will send back the constructed image to the client, usually a browser.

For other types of graphs (for example Gantt charts) some of the above standards still apply but since they have a very different usage and layout the instance variables will be different.

12.2. Common object for Pie Graphs

For pie plots there are no concepts of axis or scales. Instead Pieplots have a size and position within the graph. Figure 12.2, “Commonly used objects in Piegraphs (common-obj-piegraph.php)” shows the common objects for a typical Pie graph

Figure 12.2. Commonly used objects in Piegraphs (common-obj-piegraph.php [example_src/common-obj-piegraph.html])



The following should be noted.

- Instead of an instance of class Graph and instance of class PieGraph must be used
- ```
$graph = new PieGraph($width,$height);
```
- The title and subtitle has the same semantics as for cartesian graphs
  - The pieplot is an instance of the class PiePlot which is given the data to be used in the pieplot as first and only argument

```
$pieplot = new PiePlot($data);
```

- The size of the pieplot radius is given as fractions of the minimum of the width and height of the graph
- The position of the center of the pieplot is given as fractions (width, height) of the graph
- The legend is controlled in the same way as for cartesian graphs by accessing the instance variable "legend". In the Piegraph in Figure 12.2, “Commonly used objects in Piegraphs ( common-obj-piegraph.php)” we have changed the layout of the legend by the two lines

```
$graph->legend->SetPos(0.5,0.97,'center','bottom');
$graph->legend->SetColumns(3);
```

These two method calls adjust the position and the internal layout of the legend. The first line instructs the library to place the legend at a point (defined in fractions of width and height) (0.5, 0.97) and align

that point with the (center, bottom) anchor point in the graph box. The second lines specifies the number of columns to use int the legend.

---

# Chapter 13. Getting hold of the data to be displayed

The initial obstacle that must be negotiated is to get hold of the data to be displayed since the library itself is completely agnostic to where the data comes from. The library will use data supplied in one or more arrays and it is up to the user of the library to get hold of the data to populate these arrays with proper data. In principle the data can come from one of the following places

1. Hard-coded data in the script. This is the least flexible and can only really be recommended for examples and really static data.
2. Data stored in plain text files.
3. Data stored in binary format in flat files.
4. Data stored in a database
5. Data sent to the script via URI parameter passing (either GET or POST HTTP constructs can be used).

In the following sections we will shortly discuss each of these methods.

## Caution

The library assumes that the data available in an array that starts with index 0

## 13.1. Static data

This is the simplest way and consists of only specifying the data in one or several usual PHP arrays directly in the graph script. For example to specify data for a Pieplot one could use the following constructions

```
$data = array(1,8,5,4,12,18);
$pieplot = new PiePlot ($data);
```

This is the method used for all examples under the "Example/" directory.

## 13.2. Reading data from a file

The second method in order of complexity is to read the data from plain text files. An example on how to use this was shown in Section 4.2.2, “Preparing the data”. The library contains utility methods to ease reading of plain textual data in one of the following formats:

1. One column
2. Two columns
3. Comma separated values, CSV

The utility class to handle this is called Class `ReadFileData` and contains three utility methods corresponding to the list above, they are `ReadFileData::From1Col()`, `ReadFileData::From2Col()`, `ReadFileData::FromCSV()` and `ReadFileData::FromCSV2()`.

These methods are described shortly below

---

Getting hold of the  
data to be displayed

---

`ReadFileData::From1Col($aFileName, $aCol1)`

Reads data from a text file with one column of data and stores in the supplied \$aCol1 vector.

Typical data looks like

```
123
14.5
19.2
```

which would result in

```
$aCol == array(123, 14.5, 19, 2)
```

`ReadFileData::From2Col($aFileName, $aCol1, $aCol2, $aSepChar=' ')`

Reads data from a text file with two columns separated by the specified character.

Typical data looks like

```
12,15
13,34
14,27
```

which would result in

```
$aCol1 == array(12,13,14);
$aCol2 == array(15,34,27);
```

`ReadFileData::FromCSV($aFile, &$aData, $aSepChar=',', $aMaxLineLength=1024)`

This method reads comma separated values from a specified file. The values are all separated by the specified character. This method can be seen as a generalization of From1Col() method.

Typical data looks like

```
12,34,56,18,19.7,55
```

which would result in

```
$aData == array(12,34,56,18,19,7,55)
```

`ReadFileData::FromCSV2($aFile, &$aData, $aOptions = array())`

This method also reads comma separated values from a file but with more advanced options to control how the data is read. This can be seen as a generalization of From2Col() method.

The possible options and their default values for this method are

- 'separator' => ','
- 'enclosure' => ""
- 'readlength' => 1024
- 'ignore\_first' => false
- 'first\_as\_key' => false

Typical data (using the default values)

```
10,55
```

```
12,78
15,98
```

would result in

```
$aData = array(
 0 => array(10,12,15),
 1 => array(55,78,98)
) ;
```

If 'first\_as\_key'=>true and the data looks looks like

```
"key", "value"
10,55
12,78
15,98
```

the data would instead be read as

```
$aData = array(
 "key" => array(10,12,15),
 "value" => array(55,78,98)
) ;
```

```
ReadFileData::FromMatrix($aFile,
$aSepChar=')
```

This method is especially suited to read matrix data from a file for use with the Matrix visualization (described in Chapter 22, *Matrix graphs*). Each line in the file corresponds to one row in the matrix.

Typical data can look like

```
13,87,12
15,99,33
19,86,61
```

Which wold return a matrix looking like

```
array(
 array(13,87,12),
 array(15,99,33),
 array(19,86,61)
) ;
```

### 13.3. Sending data to a graph script with URI arguments (GET and POST)

In order to send data between different HTML pages (or to a specific page) there are two ways to do this. Either by including the parameters in the URI directly, (e.g. `http://localhost/mygraph.php?d1=12`) or by creating a POST request and send to a page. The details of these two methods are discussed below.

#### Note

There might be some confusion when to use the '&' and when to use '&' to separate variables in GET or POST requests.

When writing URL in HTML the string should always be written with a full entity encoding since the '&' is a special character indicating the start of an entity that ends with an ';' . So for example writing a URI in a HTML page should look like.

```

```

the browser will then correctly convert the URI to single '&' which is what should be sent to the server. When typing URI directly in the browser (or in any plain text file) one should of course always just use a single '&'

### Tip

Use the method `http_build_query()` to build http queries from an array with keys and values.

## 13.3.1. Using GET arguments

GET arguments are the arguments that can be added as part of the URI. For example as

```
http://localhost/mygraph.php?id=12&start=20081223&end=20090115
```

PHP automatically places all given arguments in the "super global" array variable `$_GET` as an associative array.

There are a couple of things to note here

- The values in the argument string in the URI must be URL encoded, this can easily be done with the PHP function `urlencode()` (See PHP Manual [<http://php.net/manual/en/function.urlencode.php>])
- When the arguments are read from `$_GET` they must be un-quoted with a call to `urldecode()`
- Some browsers restrict the length of the URI that they will read (typically < 2048bytes) which means that there is a limit on how much data can be sent as URI arguments
- Some browsers will allow the syntax "`a[0]=10&a[1]=11&a[2]=12`" in order to send the array `(10,11,12)`

This way of specifying the argument string is mostly useful when

- The arguments are short
- When the user should be able to bookmark an URI
- When the data is not sensitive (since it can be seen in the URI)
- When the graph should be viewable by clicking on a link (more on this when we compare the GET method with the POST method below)

The "best practice" of using this method is to send a short key (or id) to the graph script and the graph script itself will use this id to extract the real data from a DB (or a plain text file). This way the same core graph script can be used in various contexts to display wanted data.

## 13.3.2. Using a POST request

### Warning

This is a fairly advanced topic and it is recommended to use the other methods of sending data to a script unless the specifications explicitly demands that a POST request is constructed. Fur-

thermore this requires a very good understanding of HTTP request headers and the difference between server side and browser side so if you are not sure that you have the necessary background we strongly recommend to stay away from this method.

Two of the obvious restrictions with the GET method is that **a)** the length of the data is restricted and **b)** the data is visible directly in the URI. The other way to send data as part of the HTTP request is to use the POST method.

Unfortunately this is not as easy as just doing some magic and then we get the same functionality as with the GET method. Even some authors get this wrong in some very prominent PHP text books. Unfortunately it will take us too far to discuss all the details of HTTP request headers (as described in RFC2616 [<http://www.w3.org/Protocols/rfc2616/rfc2616.html>]) but we will explain the very basics.

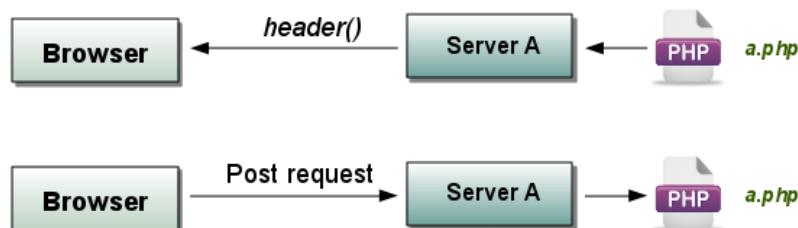
First. let's state what an HTTP request is

- A POST HTTP request is used to send data to a specified URI from a client. A client is normally a WEB-browser but can also (as we will use it) be a script that sends the request to the same or to another server. The data is URL encoded and passed in the body of the request. There is no theoretically limit on the length of the data.

A further common misunderstanding is that it is possible to use the PHP method `header()` in order to create a POST request. This is even given as an example in the notes to the `header()` method in the PHP manual (See PHP Manual [<http://php.net/manual/en/function.header.php>]). This is **wrong**. It is **not** possible to use the `header()` method to send a POST header. Trying to do this reveals a basic misunderstanding of the role of a server and client.

- The `header()` method is used to send one or more headers from the server **to the client** (i.e. WEB-browser)
- The HTTP POST request goes from the client (i.e. WEB-browser) **to the server**

**Figure 13.1. Post vs. header() data direction**



Note that the image is greatly simplified to help illustrate the vital point on data direction. For example the post request shown to originate from a browser could originate from any client, for example another script taking the role of a client.

Hence it is never possible to "simulate" a POST call with the use of the `header()` function. There are basically three (correct) ways to simulate a POST request as described below.

Before we continue lets first recapitulate the most common use of a POST request, i.e. in a HTML form submission. When data is entered in a form on a HTML page and the user presses the "Submit" button the data from the form is packed as a POST request which is sent to the server at the specified URI (the action URI). The server will reply back to the POST request (with the data sent back from the target of the post request) and the browser will show the reply in the window in place of the original HTML form page.

However there is a crucial difference when we do this manually from a script (running on the server) compared with the original form post data from the browser to the server. After issuing a POST request (originating from a HTML form) the browser automatically replaces the current page with the reply from the POST request as a "new" page (by default using the same target window as the request was made from).

This is not possible to do when sending a "fake" post request to a page since we are not the browser. Instead what we will see in the browser is the page *sending the POST request*, and not the target of the post request. The best we can accomplish is to show the reply inline in the calling page which are then shown in the browser.

This means that it is not possible to create a POST request and then somehow directly show the reply as the resulting image. Instead what we can do is to send the data to a image script (via a POST header) and then the graph script can write the image to a file that is accessible from the server.

So to summarize. What we can do with a post request is to send the data to a script b.php from a script a.php. The b.php can then execute some statements, for example creating a graph and store it on the server. This stored image can later be read by the a.php script, for example via a <img> tag.

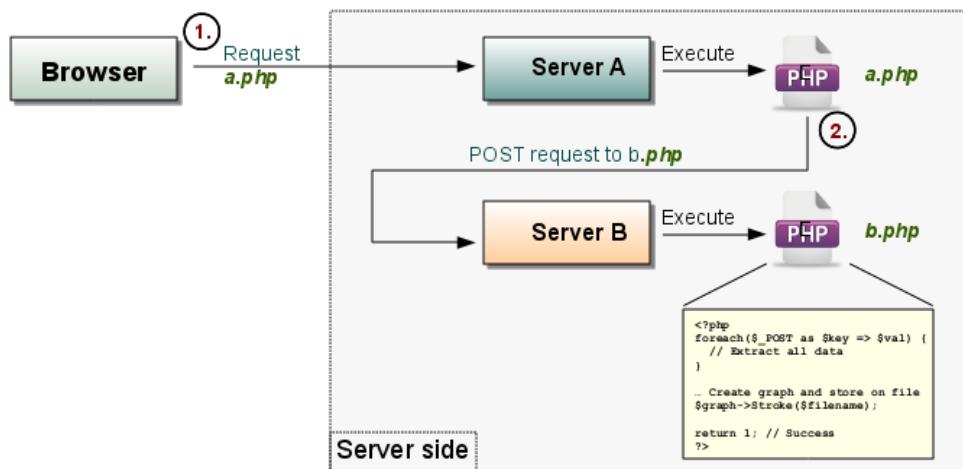
### How to create a POST request

There are in principle three ways of constructing a POST request to send data to a specified URI as shown below.

After we have made the request (with any of the three methods shown below) the server will reply back with the response created by the URI. This response is any output sent by the script we are sending our request to. Normally this should just be a return code indicating if the request was successful or not.

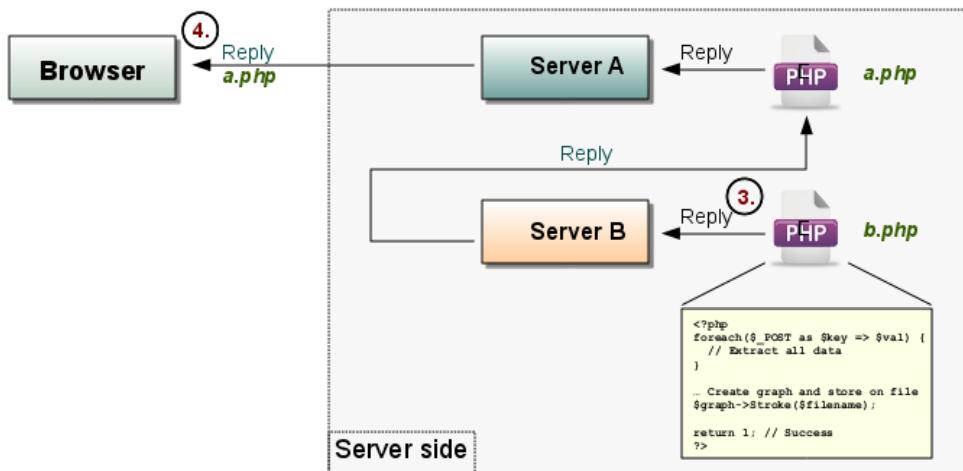
Remember that these are calls made from the script a.php running on server A to a script b.php running on server B. There are no browser involved in these calls apart from the initial request to run a.php. The figure below illustrates the first phase when the request is sent to the B side

**Figure 13.2. The request phase of a POST header**



1. The request is initially made by calling the a.php script, e.g. <http://localhost/a.php> in the browser
2. When the a.php script is executed it will create the POST header and call script b.php on the server B (could possibly be the same server). Since the browser is displaying script a.php we can never change that directly but we can display the reply from b.php in the page displayed by a.php

**Figure 13.3. The reply phase of a POST request**



3. The b.php returns a reply by for example echoing back a reply code
4. The script running the browser receives its final data (which is the reply from the b.php script) and then finish the original request started in step 1.

**Note:** All the `sendpostData_vX()` methods below assumes that the data is already urlencoded.

### 1. Create a stream request.

The advantage with this method is that it is available by default in PHP  $\geq 4.3$  and of course in PHP5,6 without the need to install additional libraries.

```
<?php
function sendData_v1($url, $data) {
 $opts = array('http' => array(
 'method' => 'POST',
 'header' =>
 "Content-type: application/x-www-form-urlencoded\r\n".
 "Content-Length: ".strlen($data)."\r\n".
 'content' => $data,
));
 $stream = stream_context_create($opts);
 $fp = fopen($url, 'rb', false, $stream);
 if (!$fp) { // Some error handling }

 // Find out what the page returns as its body
 $reply = stream_get_contents($fp);
 if ($reply === false) { // Some error handling }

 return $reply;
}
?>
```

### 2. Open a socket directly and write to it

```
<?php
function sendData_v2($url, $data, $port=80) {
```

```
$errno=-1;
$errstr='';
$fs = fsockopen($url,$port,$errno,$errstr);
if($fs === false) { // Some error handling }

$header = "POST $url HTTP/1.0\r\n";
$header .= "Content-Type: application/x-www-form-urlencoded\r\n";
$header .= "Content-Length: " . strlen($data) . "\r\n\r\n";
fputs($fs, $header . $data);

// Find out what the page returns as its body
$reply = '';
while(!feof($fs)) {
 $reply .= fgets($fp,8192);
}
return $reply;
}
?>
```

### 3. Use CURL to handle all the necessary details

**Note 1:** This requires that curl libraries are installed and that curl is enabled in the PHP installations.

**Note 2:** Depending on the application there might be many more options that needs to be tweaked. The one used below are just the bare necessities.

**Note 3:** Using CURL is the most general way to handle POST requests and simplifies the additional complexity if we want to add encryption (i.e. HTTPS) in the connection handling.

```
<?php
function sendpostData_v3($url,$data)
{
 // Initialize and get the curl handle
 $ch = curl_init();

 // Include possible headers in the reply from the server as well
 curl_setopt($ch, CURLOPT_HEADER, true);

 // Return the reply as a string from curl_exec() instead of directly to stdout
 curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

 // The URI we want to send the post data to
 curl_setopt($ch, CURLOPT_URL, $url);

 // Use the POST method
 curl_setopt($ch, CURLOPT_POST, true);

 // All the data to be sent
 curl_setopt($ch, CURLOPT_POSTFIELDS,$data);

 $reply = curl_exec ($ch);

 curl_close($ch);
 return $reply;
}
```

?>

## 13.4. Reading data from a database

Since there are so many databases and so many ways to organize data in tables it is impossible to give generic examples. Instead we point out some common pitfalls when gathering data from a database.

- Make sure the data is valid numeric data. It is a common mistake to read NULL (or empty strings "") values and try to plot them. Beware that the PHP function `empty()` also will return true for 0 and "0" values so it is not enough to just test with `empty()` since this will also remove all values that happens to be valid 0 values. To avoid this use the following modified empty method.

```
function empty0($aValue){
 if (!is_numeric(trim($aValue)))
 return empty($aValue);
 return false;
}
```

- Some databases (e.g. MySQL) returns a column with decimal type values as string in a SELECT statement which normally is handled by automatic conversion in PHP but since PHP has different interpretation of "0.00" and 0.00 in respect to what is interpreted as empty or not this needs to be carefully handled.

## 13.5. Reading binary data from a file

This can be potentially dangerous since the exact format for the binary data can differ depending on convention and system. The advantage is that it usually gives smaller data files and is usually not as suspect to data corruption since it is not easy to read directly into a text editor and change. In PHP a variable can be stored in a binary format using the `serialize()` function and read back from a file with `unserialize()`.

### Tip

It is also possible to store binary data in a text file as a string. One way of doing this is to do a base64 encoding of the binary data. This will guarantee that the encoded data will only be printable characters which can be stored in a normal string. One thing to remember is that this will enlarge the data by roughly 30%. Another example on when base64 encoding is useful is to store binary images in a text file. This is the way for example all built-in images are stored in the library. The image is re-created from the string by first decoding the base64 data and then read in the data with the GD function `imagecreatefromstring()`.

## 13.6. Different types of NULL data handling

In some data there might be data points missing. For example if some experimental data is to be graphed there can be data points where the equipment was faulty and not valid values exist. The library offers to way to handle this.

1. Leave the data point empty and leave a hole in the graph, for example a break in a line plot. This happens if the null value is specified as either one of the following values

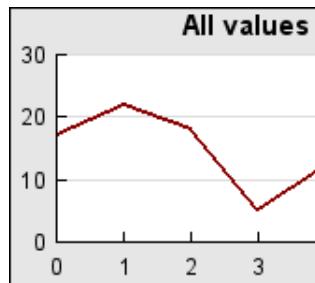
- 'x' - A single character 'x'
- '' - An empty string

- NULL - The NULL value
2. Ignore the data point and connect the previous and next data point with the line in a line graph. This is accomplished by specifying the null value as
    - ' - ' - A single hyphen character

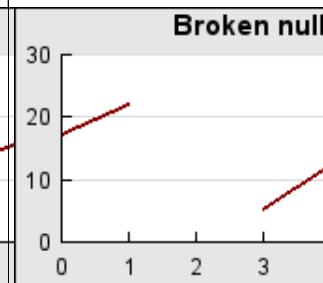
In Example 13.1, “Illustration of different types of NULL values in graphs” an example with the different types of null values are shown. In these graphs the third data point is set to null with the two different null values.

### Example 13.1. Illustration of different types of NULL values in graphs

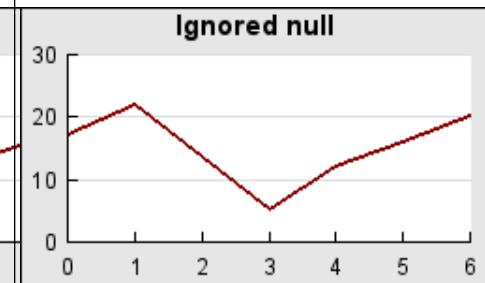
**Figure 13.4. Original graph with all values**



**Figure 13.5. Value at x=2 as ''**



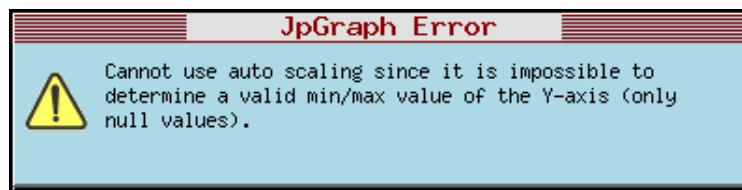
**Figure 13.6. Value at x=2 as '-'**



### Caution

If all values in a line plot are null values an error will be shown since this is not considered to be a valid plot.

**Figure 13.7. Error when only null values are specified**



## 13.7. Troubleshooting input data

There are two common mistakes that causes problems with input data.

1. The indexes in the data array has missing keys. The library assumes that the data array has consecutive values indexed from 0 and upwards. This means that the following data array specification is **illegal**

```
$data = array(1 => 23, 2 => 14, 3 => 17) ;
```

The problem with the array above is that it lacks the 0 key. A similar problem is to have a missing index, such as

```
$data = array(0 => 23, 2 => 14, 3 => 17) ;
```

If `E_NOTICE` is not enabled in `php.ini` then this mistake will give the (slightly misleading) error message shown in Figure 13.7, “Error when only null values are specified”. If full error checking is enabled in `php.ini` then these types of problem will give the standard PHP error message shown in Figure 13.8, “PHP (HTML) error when missing data keys”. This shows a PHP setup which has HTML errors enabled. Note the JpGraph textual error string at the bottom of the second PHP error message box.

**Figure 13.8. PHP (HTML) error when missing data keys**

The figure consists of two vertically stacked screenshots of PHP error messages. Both messages are preceded by a red exclamation mark icon and the text '( ! ) Notice: Undefined offset: 0 in /home/ljp/workspace/jpgraph/src/jpgraph.php on line 5225' or '5190'. Below this is a 'Call Stack' table with columns: #, Time, Memory, Function, and Location. The stack traces are identical for both errors, showing four entries:

#	Time	Memory	Function	Location
1	0.0002	53892	{main}()	./test_linenull.php:0
2	0.0487	4196524	Graph->Stroke()	./test_linenull.php:32
3	0.0502	4199088	Graph->GetPlotsYMinMax()	./jpgraph.php:1623
4	0.0502	4199088	Plot->Max()	./jpgraph.php:2898

Below the second screenshot, there is an additional error message: 'JpGraph Error Cannot use auto scaling since it is impossible to determine a valid min/max value of the Y-axis (only null values)'.

- Another typical mistake caused by "faulty data" is different convention for specifying decimal data. PHP and the library assumes that decimal data used "." (point) to separate the integer from the decimal part of a number. Some locales uses "," (comma) to do this separation. This means that the following code will **not** work.

```
$data = array(0 => '12,0', 1 => '18,0', 2 => '15,0') ;
```

Again, if `E_NOTICE` is not enabled in `php.ini` then this mistake will give the (slightly misleading) error message shown in Figure 13.7, “Error when only null values are specified”

---

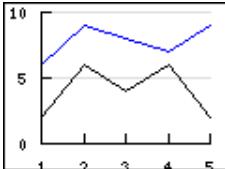
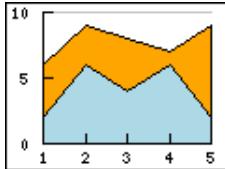
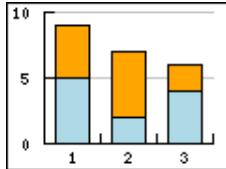
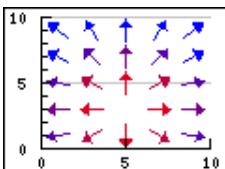
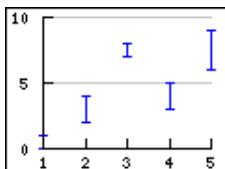
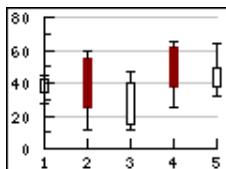
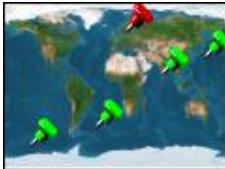
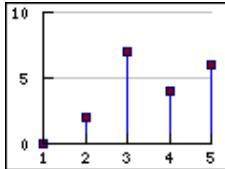
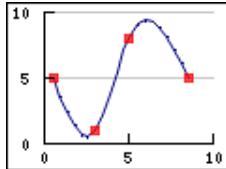
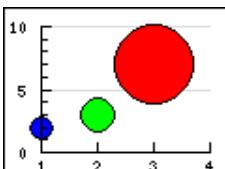
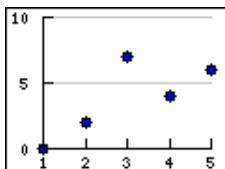
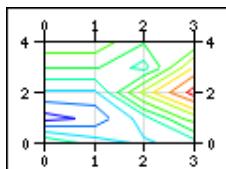
# **Chapter 14. Common features for all Cartesian (x,y) graph types**

## **14.1. The definition of linear graphs**

With cartesian graphs we refer to all plots which have orthogonal x and y axis. The library support the option of multiple y-scales (when applicable) but only on x-axis can be used.

The following principle linear graph types are supported as of v2.5 (note that some graph types may have additional subtypes that are not shown in this overview.)

**Figure 14.1. Supported principle linear graph types in the library**

 a) Line plot (See Section 15.1.1, “Creating a line graph”)	 b) Area plot (See Section 15.1.10, “Creating a filled line graphs (a.k.a. area plots)”)	 c) Bar plot (See Section 15.2, “Bar graphs”)
 a) Field plot (See Section 15.5.3, “Field plots”)	 b) Error plot (See Section 15.3, “Error plot graphs”)	 c) Stock plot (See Section 15.4, “Stock graphs”)
 a) Geo-map plot (See Section 15.5.5, “Creating Geo-maps”)	 b) Impuls (stem) plot (See Section 15.5, “Scatter graphs”)	 c) Spline plot (See Section 15.1.15, “Constructing smooth line plots with Cubic Splines”)
 a) Balloon plot (See Section 15.5.4, “Balloon plots”)	 b) Scatter plot (See Section 15.5, “Scatter graphs”)	 c) Contour plot (See Section 15.6, “Contour graphs”)

Each of these graph types have their own section where more details can be found by following the link under the corresponding graph icon.

## 14.1.1. Axis and coordinate systems

The x and y axis each in a graph has an associated scale, labels, titles, grid lines, colors and position. The axis properties are accessed as objects of the axis instance variables in the main graph class. The axis can be access vi the following instance variables.

- `Graph::xaxis`, The x-axis, (by default on the bottom)
- `Graph::yaxis`, The y-axis, (by default on the left side)
- `Graph::y2axis`, The second y-axis (by default on the right side)

In addition the library also supports the use of multiple y-axis and they are accessed via an instance array

- `Graph::ynaxis[ ]`

All axis in turn are instances of `class Axis` and hence share common properties. The only two property that can be publicly accessed on the axis are

- `Axis::scale`. The scale of the axis. An instance of either `LinearScale`, `LogScale` or `DateScale`. This rarely needs to be accessed directly.
- `Axis::title`. The axis title. On the x-axis this is horizontal by default and on the y-axis the title is vertical by default

On the other hand there are a large amount of methods that can be used on the axis to adjust various properties. Some examples of commonly used methods are given below. The full description of each method is given in the API reference.

- Adjusting the labels
  - `Axis::SetLabelFormatString($aFormStr,$aDateFormat=false)`. Specifies the labels format string assuming `printf()` format if `$aDate` is false and in `date()` format if `$aDate` is true.
  - `Axis::SetLabelFormatCallback($aCallbackFunc)`
  - `Axis::SetLabelAlign($aHorAlign, $aVertAlign='top', $aParagraphAlign='left')`
  - `Axis::HideLabels($aHide=true)`
  - `Axis::SetTicklabels($aLabels, $aLabelColors=null)`
  - `Axis::SetLabelMargin($aMargin)`
  - `Axis::SetLabelSide($aSide)`
  - `Axis::SetFont($aFamily,$aStyle=FS_NORMAL,$aSize=10)`
  - `Axis::SetLabelAngle($aAngle)`. Species the angle of the label. **Note:** It is only possible to use arbitrary angles if the font is a true type font. The built in bit map fonts only supports 0 and 90 degree text strings.
- Adjusting the tick marks
  - `Axis::SetTickSide($aSide)`

- Axis::SetTickPositions(\$aMajPos, \$aMinPos=NULL, \$aLabels=NULL)
- Axis::HideTicks(\$aHide)
- Adjusting the actual axis
  - Axis::HideLine(\$aHide=true), Only hide the axis but show the labels
  - Axis::Hide(\$aHide), Hide both axis and labels
  - Axis::SetWeight(\$aWeight), Set the weight in pixels of the axis
  - Axis::SetPos(\$aPositionOnOtherScale) Specifies the position of the axis on the other scale. The position is given the scale of the other axis. There are two special values (strings) that can be given and those are
    - 'min' - Will position the axis at the minimum value of the other scale
    - 'max' - Will position the axis at the maximum value of the other scale
- Adjusting the title
  - Axis::SetTitle(\$aTxt)
  - Axis::SetTitleMargin(\$aMargin)
  - Axis::SetTitleSide(\$aSide)
  - Axis::SetColor(\$aColor, \$aLabelColor)

The above methods are valid for all possible axis. So for example the following line sets the font for the labels on the x-axis

```
$graph->xaxis->SetFont(FF_ARIAL, FS_NORMAL, 12);
```

and the following code set the font for the y-axis

```
$graph->yaxis->SetFont(FF_ARIAL, FS_NORMAL, 12);
```

## 14.1.2. Adjusting the axis look and feel

The two major ways to adjust the look and feel of the axis are adjustments of the color and the weight (i.e. width) and this can be done with the appropriate methods as described above.

- Axis::SetColor(\$aColor, \$aLabelColor), For example \$graph->xaxis->SetColor('teal'). Please note that by default the color for the labels will be that of the line if the label color is not explicitly specified.
- Axis::SetWeight(\$aWeight). Specify the weight (in pixels of the axis)

## 14.1.3. Adding grid lines in the plot

### Note

The possibility of having different styles, colors and weight for minor and major grid lines was added in 3.0.4 and is not available in earlier releases.

Grid lines will make it easier to see where the data points are in the graph. The grid lines are access by the properties "xgrid" and "ygrid" of the Graph class. By default only the y-axis grid are enabled by default. The following code example enables the major grids for both the x- and y-axis.

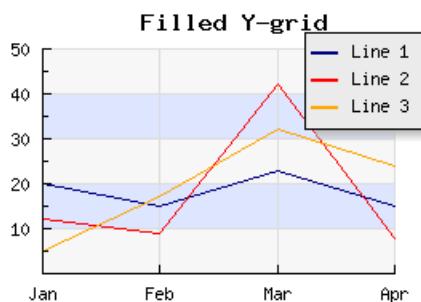
```
$graph->xgrid->Show();
$graph->ygrid->Show();
```

The grid lines are instances of Class Grid and supports the following methods

- Grid::SetColor(\$aMajColor,\$aMinColor=false). Specify the color for the major and minor grid lines
- Grid::SetWeight(\$aMajorWeight,\$aMinorWeight=1). Specify the weight of the line for the major and minor grid line
- Grid::Show(\$aMajGrid=true,\$aMinGrid=false). Determine which grid lines should be shown
- Grid::SetLineStyle(\$aMajorType,\$aMinorType) This method makes it possible to adjust the line style of the grid lines (both major and minor separately). The line style is specified as a string and can have one of the following values
  - "solid"
  - "dotted"
  - "dashed"
  - "longdashed"
- Grid::SetFill(\$aFlg=true,\$aColor1='lightgray',\$aColor2='lightblue')

The last method needs an explanation. The fill refers to the possibility to fill the space between the grid lines with alternating colors as specified in the method call. Figure 14.2, “Using alternating fill colors in the grid (filledgridex1.php)” shows an example on how this can be used to make it easier to read a plot.

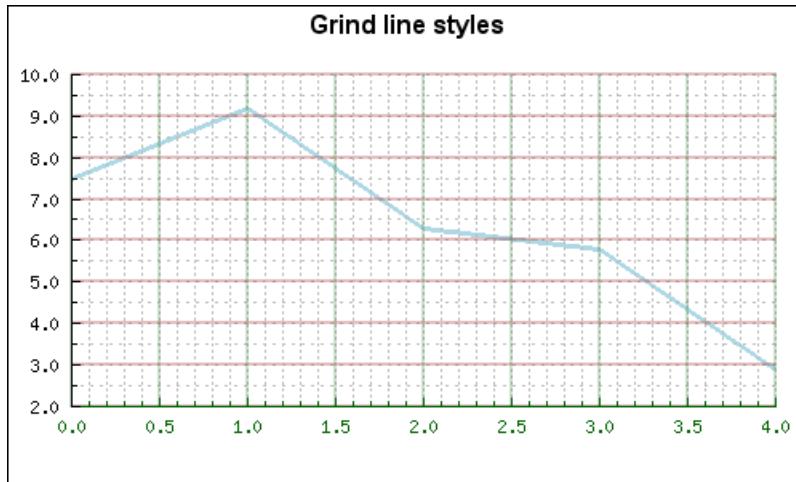
**Figure 14.2. Using alternating fill colors in the grid (filledgridex1.php)  
[example\_src/filledgridex1.html]**



In the above example we have also used the possibility of using alpha-blending (for example on the shadow on the legend box).

The example below shows how to use different styles for the major and minor grid lines

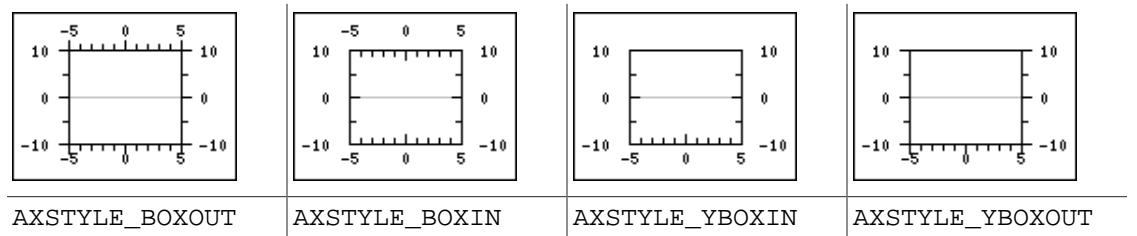
**Figure 14.3. Using different grid styles for major and minor grids (gridstylesex1.php) [example\_src/gridstylesex1.html]**



## 14.1.4. Predefined scientific axis setups

In order to make it easier to setup a couple of typical axis configuration used in science plots there are four predefined configurations as shown in Figure 14.4, “Predefined scientific axis positions”.

**Figure 14.4. Predefined scientific axis positions**

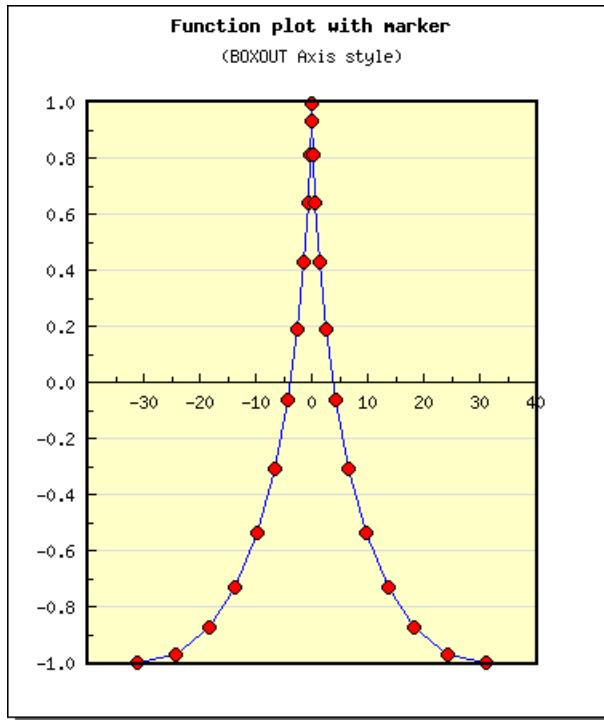


The styles can easily be setup with a call to the method

- `Graph::SetAxisStyle($aStyle)`

An example of using this setup of the axis is shown in Figure 14.5, “Example of AXSTYLE\_BOXIN axis style (funcex2.php)”

**Figure 14.5. Example of AXSTYLE\_BOXIN axis style (funcex2.php)  
[example\_src/funcex2.html]**



### 14.1.5. Other possible ways to position the axis

The axis can be manually positioned with a call to

```
Axis::SetPos($aPos)
```

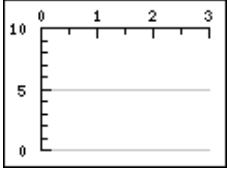
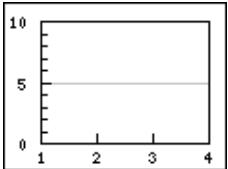
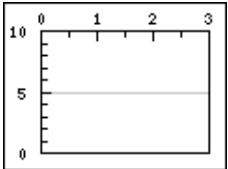
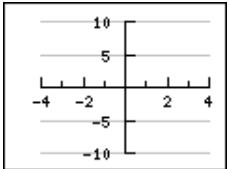
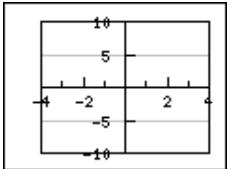
The argument \$aPos is normally the coordinate position on the "other" axis where the crossing of this and the other axis should be. There are also two special positions which are given as strings. They are 'min' and 'max'. Not surprisingly these special positions will always refer to the min and max scale value of the other axis.

The position given is the scale position on the "other" axis, i.e. for the x-axis the position specifies the crossing of the y-axis and vice versa.

Since it is possible to manually specify all aspects of the axis the table below shows some typical common setups and the principle calls needed to achieve the illustrated affect.

**Table 14.1. Axis configurations**

	This is the default setup and not extra configurations are needed and it is the same as  <code>\$graph-&gt;SetAxisStyle(AXSTYLE_SIMPLE);</code>
--	-------------------------------------------------------------------------------------------------------------------------------------------------------

	This setup is configured by moving the x-axis to the top  <pre>\$graph-&gt;xaxis-&gt;SetPos('max'); \$graph-&gt;xaxis-&gt;SetLabelSide(SIDE_UP); \$graph-&gt;xaxis-&gt;SetTickSide(SIDE_DOWN);</pre>
	This is the standard style but with an added box around the plot area.  <pre>\$graph-&gt;SetAxisStyle(AXSTYLE_SIMPLE); \$graph-&gt;SetBox();</pre>
	 <pre>\$graph-&gt;xaxis-&gt;SetPos('max'); \$graph-&gt;xaxis-&gt;SetLabelSide(SIDE_UP); \$graph-&gt;xaxis-&gt;SetTickSide(SIDE_DOWN); \$graph-&gt;SetBox();</pre>
	This configuration locks the x- and y-axis at the origin  <pre>\$graph-&gt;xaxis-&gt;SetPos(0); \$graph-&gt;yaxis-&gt;SetPos(0);</pre>
	With an added box around the plot area  <pre>\$graph-&gt;xaxis-&gt;SetPos(0); \$graph-&gt;yaxis-&gt;SetPos(0); \$graph-&gt;SetBox();</pre>

## 14.2. Specifying and formatting the overall displayed graph

The overall look and feel of the graph can be adjusted in a number of ways.

- color of the plot area
- color of the margin
- size of the margin (and hence the position of the plot area)
- title(s) of the graph, a graph can have both a "title", "subtitle" and a "subsubtitle"
- drop shadow around the graph
- adding how long time it took to generate the graph

### 14.2.1. Adjusting size, margins and frame

The overall size of the graph is specified in the initial creation of the "\$graph" instance. This is mandatory.

```
$graph = new Graph($width,$height);
```

The margin determines the space between the edges of the plotarea and edge of the graph. Please note that the edge does not include the frame around the entire graph area, i.e. whatever weight the frame edge has it doesn't impact on the margin area. The margin is specified in the order left,right,top and bottom margin and the size is specified in pixels.

```
$graph->SetMargin(30,10,40,20);
```

The default margin will be as narrow as possible but still have enough space to show the titles and labels. The final adjustments is wheter or not the graph will have a frame or not. There are two distinctions to be made here. No frame meaning that the margin area will not be colored (i.e. painted) and no frame as in no edge around the graph area. The first case is achieved by calling the `SetFrame()` method on the Graph class

```
$graph->SetFrame(false);
```

and the second case is achieved by setting the frame edge to have weight zero. Since the weight is given as the third argument we wwill have to put in valid "dummy" argument for the on/off flag and the frame color parameter.

```
$graph->SetFrame(true,'black',0);
```

Refer to Figure 12.1, “Commonly used objects in a graph (common-obj-graph.php)” for definitoin of plot area and margin.

## 14.2.2. Adding drop shadow to the graph

A graph can have a drop shadow specified with the method `SetShadow()` which has the signature

```
SetShadow($aShowShadow=true,$aShadowWidth=5,$aShadowColor=array(102,102,102))
```

by default there will be no dropshadow. Note that the drop shadow is a straight simple shadow and has no Gaussian blur.

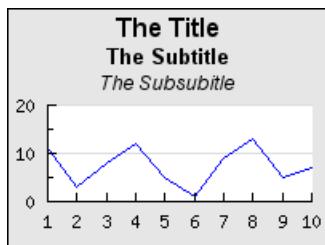
## 14.2.3. Formatting and specifying the titles of the graph

There are three possible standard titles in the graph

1. `$graph->title`
2. `$graph->subtitle`
3. `$graph->subsubtitle`

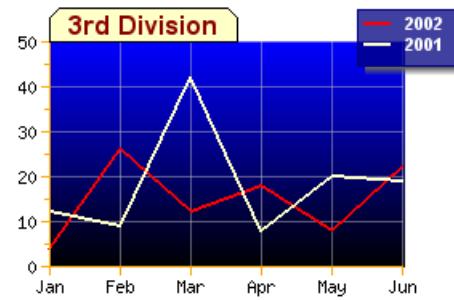
Each title is an instance of the Text class which means it supports the standard text adjusting methods `SetFont()`, `Set()`, `SetColor()` etc. The title s will be centered on the top of the graph area with the title highest up, subtitle below and finally the subsubtitle below that as shown in Figure 14.6, “The various titles in a graph (titleex1.php)”

**Figure 14.6. The various titles in a graph (titleex1.php) [example\_src/titleex1.html]**



In addition to the above standard titles there is also another type of graph titles called the "Tabular title". This is a title that sits directly on the plot area looking like a "tab" in a sorting cabinet. Figure 14.7, "Tabtitle and gradient background (gradbkge1.php)" and Figure 14.8, "Tabtitle and image marker in a line plot (imgmarkerex1.php)" shows examples of this.

**Figure 14.7. Tabtitle and gradient background (gradbkge1.php) [example\_src/gradbkge1.html]**



**Figure 14.8. Tabtitle and image marker in a line plot (imgmarkerex1.php) [example\_src/imgmarkerex1.html]**



These type of titles are created by using the

```
$graph->tabtitle
```

property in the Graph class. For example the tabtitle in Figure 7.2, "Making use of transparency to combine two plots (barlinealphaex1.php)" was generated by adding the lines

```
$graph->tabtitle->Set('Year 2003');
$graph->tabtitle->SetFont(FF_ARIAL,FS_BOLD,10);
```

## Adding special background to the title

There is one more way to make the graph title stand out that. This feature is probably one of the least known ways to adjust the graph title and is used to put a lot of emphasis on the title by having a separate color (and style) for the background of the graph title.

### Note

This feature was added by the request of one of our users who had a company design policy that demanded these type of formatting options for graphs.

The two methods used to control this style are

- `Graph::SetTitleBackground($aBackColor='gray',  
$aStyle=TITLEBKG_STYLE1, $aFrameStyle=TITLEBKG_FRAME_NONE,  
$aFrameColor='black', $aFrameWeight=1, $aBevelHeight=3,  
$aEnable=true)`

This method is the basic method to specify a solid color background with the defined style and type of framing. The first style parameter specifies the extension of the background and how it interacts with the frame around the image.

- `TITLEBKG_STYLE1`, The title frame will be drawn inside the overall graph frame
- `TITLEBKG_STYLE2`, The title frame will extend all the way to the edge of the graph and overwrite any graph frame
- `TITLEBKG_STYLE3`, This is the same as `TITLEBKG_STYLE2` apart from when it is used together with framestyle `TITLEBKG_FRAME_BEVEL` then the bevel frame border is on top of the title frame border
- `Graph::SetTitleBackgroundFillStyle($aStyle,$aColor1='black',  
$aColor2='white')`

The method determines adjusts the way the background of the title is filled by allowing a simple pattern filling to be set. This pattern filling is either horizontal or vertical stripes (lines) with user selectable colors. The possible patterns are

- `TITLEBKG_FILLSTYLE_HSTRIPED`, The background will have horizontal stripes with `$aColor1` on a background with `$aColor2`
- `TITLEBKG_FILLSTYLE_VSTRIPED`, The background will have vertical stripes with `$aColor1` on a background with `$aColor2`
- `TITLEBKG_FILLSTYLE_SOLID`, The background will only have one color specified as `$aColor1`

To better show the effect of the possible combinations the table below shows a matrix of the possible combinations and the resulting styles. To keep the table small only the relevant top area of the graph with the title is shown. In addition to keep the style simple we have only used shades of gray.

Frame/Fill style	<code>TITLEBKG_FILLSTYLE_HSTRIPED</code>	<code>TITLEBKG_FILLSTYLE_VSTRIPED</code>	<code>TITLEBKG_FILLSTYLE_SOLID</code>
<code>TITLEBKG_STYLE1,</code> <code>TITLEBKG_FRAME_NONE</code>			
<code>TITLEBKG_STYLE2,</code> <code>TITLEBKG_FRAME_NONE</code>			
<code>TITLEBKG_STYLE3,</code> <code>TITLEBKG_FRAME_NONE</code>			
<code>TITLEBKG_STYLE1,</code> <code>TITLEBKG_FRAME_FULL</code>			
<code>TITLEBKG_STYLE2,</code> <code>TITLEBKG_FRAME_FULL</code>			

TITLEBKG_STYLE3,				The full graph title
TITLEBKG_FRAME_FULL				
TITLEBKG_STYLE1,				The full graph title
TITLEBKG_FRAME_BOTTOM				
TITLEBKG_STYLE2,				The full graph title
TITLEBKG_FRAME_BOTTOM				
TITLEBKG_STYLE3,				The full graph title
TITLEBKG_FRAME_BOTTOM				
TITLEBKG_STYLE1,				The full graph title
TITLEBKG_FRAME_BEVEL				
TITLEBKG_STYLE2,				The full graph title
TITLEBKG_FRAME_BEVEL				
TITLEBKG_STYLE3,				The full graph title
TITLEBKG_FRAME_BEVEL				

### Note

There is a change in the behaviour of the latest GD library that causes style 2 and style 3 in JpGraph 2.x to be more or less visually identical as can be seen above. Using 1.x will more clearly show the difference. The reason is that with JpGraph 2.x we rely more heavily on the GD built in functions while in 1.x many of the functions were instead implemented in the library (since they weren't earlier available in GD 1.x). The change in JpGraph 2.x was made to increase the performance, unfortunately some minor differences exist in the way GD 2.x handles border line cases as how to handle pixel-by-pixel accuracy (or convention). Over time these changes will become less and less as parts of the library will be adjusted to the new GD 2.x behaviour.

A typical example of changed behaviour is how rectangles with thickness > 1 are positioned. The library assumes that the top left (0,0) corner is the very top left part of the rectangle. This is not so with GD 2.x. Instead (0,0) is the center point on the line. If the line has thickness=1 they are identical but not if the line has a thickness > 1

### 14.2.4. Specifying the image format to use

As described in Section 3.2, “Necessary system requirements for the library” the actual image compression available depends on the system setup. Assuming the needed setup is done then the image format can be specified by using the method `Graph::SetImgFormat()` and specifying the wanted format as a string. For example as

```
$graph->SetImgFormat('jpg', 80);
```

The method call above selects JPEG image compression algorithm and tells the algorithm to set the quality parameter to 80%. This additional parameter is only applicable together with JPG formatting.

### 14.2.5. Generic line formatting

Any lines in the library can have a number of formatting option. Attributes that can be adjusted are

- Lineweight (thickness). The weight is specified in pixels and can be any integer values
- Linestyle , (dashed, dotted, solid etc). The style is specified as a string using the line method `SetLineStyle()`.

The available line styles are

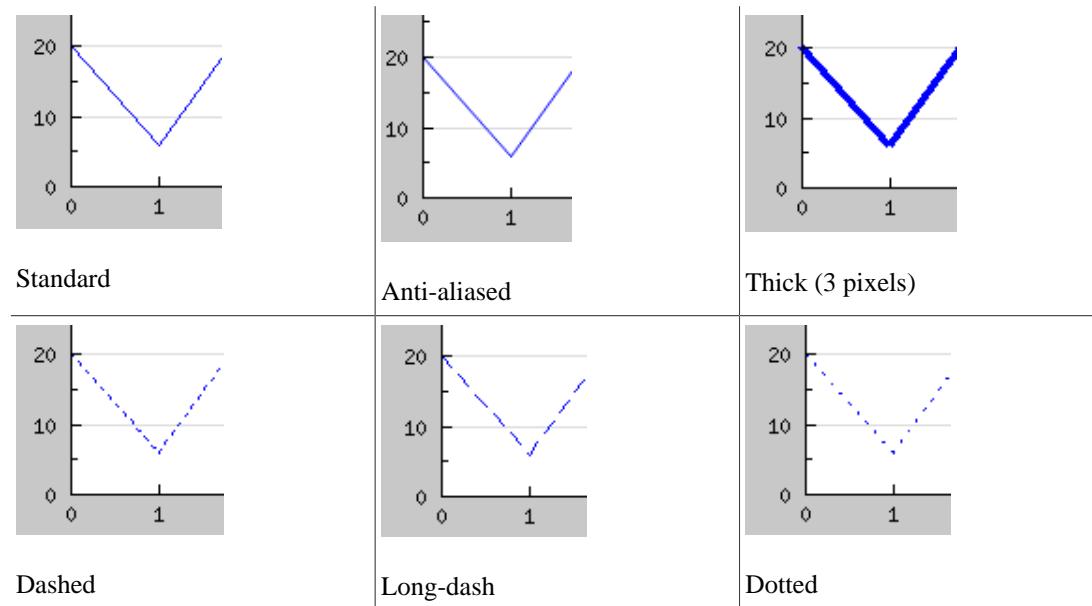
- 'solid', A solid line
- 'dotted', A line with dots
- 'dashed', A line with short hyphens
- 'longdashed', A line with long hyphens

For example to following method call set the y-grid to use the "dotted" style

```
$graph->ygrid->SetLineStyle('dotted');
```

The image in Figure 14.9, “Different line formatting” shows example of different line formatting styles

**Figure 14.9. Different line formatting**



- Color. The line color is (as usual) adjusted with the method `SetColor()`.

For example the following line sets the color of the x-grid to lightblue.

```
$graph->xgrid->SetColor('lightblue');
```

### Note

If anti-aliasing is enabled (see ??.) then all line weights will always be interpreted as having weight 1 regardless.

### Tip

To avoid drawing a line set the line weight to 0

## 14.2.6. Adding a footer to the graph

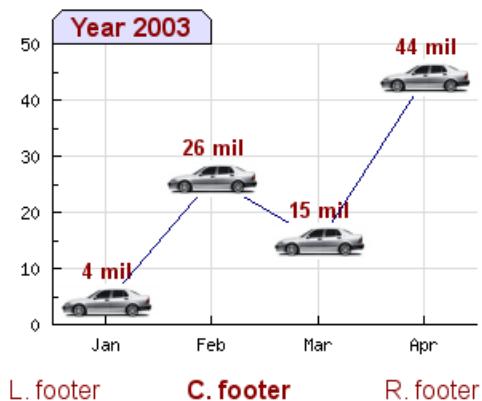
It is possible to add a footer to the graph as well. The footer can have a left, center and right part. The footer is accessed through the footer property of the Graph class and its three parts "left", "right" and "center". These parts are instances of the Text class and hence all normal text formatting can be applied to each part of the footer. The following code snippet shows how the footer parts can be set.

```
$graph->footer->left->Set('Left footer');
$graph->footer->center->Set('Center footer');
$graph->footer->right->Set('Right footer');
```

An example of using a footer is shown in Figure 14.11, “Adding a left,right and center footer (footerex1.php)”

**Figure 14.10. Using a footer in a graph**

**Figure 14.11. Adding a left,right and center footer (footerex1.php)**  
[example\_src/footerex1.html]



## 14.2.7. Adding timing of graphs

The footer in the bottom of the graph can also be used to show an approximative timing of the graph generation. This is done with the help of class `JpgTimer`. This timer class offers two methods

1. `JpgTimer::Push()`, Starts a new timer and puts it at the top of the timer stack
2. `JpgTimer::Pop()`, Pops the timer at the top of the stack and returns the time (in ms) since the timer was pushed

(The stack based design makes it possible to have multiple timers running at the same time without interfering each other.)

The footer class (Section 14.2.6, “Adding a footer to the graph”) have a special provision to handle an instance of the timer class

- `Footer::SetTimer($aTimer,$aSuffix='')`
- `$aTimer`, an instance of `JpgTimer`

\$aSuffix, an optional suffix string that will be added at the end of the timing value

At the end of the graph generation the specified timer will be sent the sign pop and the resulting timer value will be stored in the right footer of the graph (appended to any previous value in the footer). For example the following simple lines would time the graph generation.

```
<?php
// Create a new timer instance
$timer = new JpgTimer();

// Start the timer
$timer->Push();

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

// Make the bottom margin large enough to hold the timer value
$graph->SetMargin(40,20,20,60);

$graph->title->Set("Timing a graph");
$graph->footer->right->Set('Timer (ms): ');
$graph->footer->right->SetFont(FF_COURIER,FS_ITALIC);
$graph->footer->SetTimer($timer);

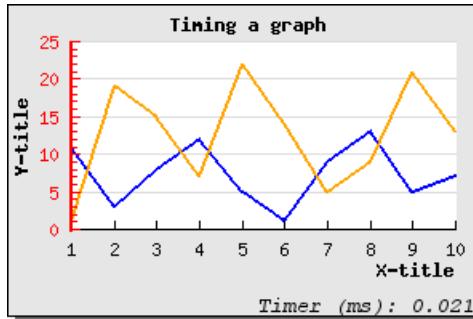
// The rest of the script as normal ..
```

## Note

Remember to make the bottom margin large enough.

Figure 14.12, “Adding a timer to the graph (in the footer) (example11.php) ” shows an example of using a timer to add the time (in ms) it took to generate the graph

**Figure 14.12. Adding a timer to the graph (in the footer) (example11.php)  
[example\_src/example11.html]**



## 14.3. Adjusting the look and feel of the plot area

The graph area can be divided in two parts.

1. The margin area
2. The plot area

These areas can have individual background colors as well as edges of user settable size and color. In addition it is possible to use an image as background with different formatting options.

### 14.3.1. Plot area and margin areas

The plot area is where the plot is drawn and the margin area is everything outside. Methods that affect the format of the plot area are

1. `Graph::SetMargin()` , This will adjust the size (and position) of the plot area
2. `Graph::SetBox($aDrawPlotFrame=true,$aPlotFrameColor=array(0,0,0), $aPlotFrameWeight=1)`, This will adjust the format of the edge around the plot area. By default this is disabled.

In a similar way the margin areas properties can be adjusted with the methods

1. `Graph::SetMargin()` , same as above
2. `Graph::SetMarginColor($aColor)`
3. `Graph::SetFram($aDrawImgFrame=true,$aImgFrameColor=array(0,0,0), $aImgFrameWeight=1)`
4. `Graph::SetFrameBevel($aDepth=3,$aBorder=false,$aBorderColor='black', $aColor1='white@0.4',$aColor2='darkgray@0.4',$aFlg=true)`

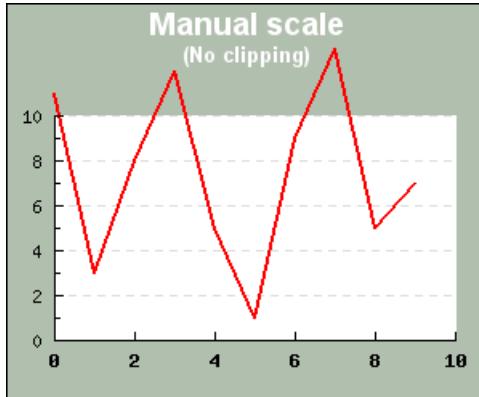
### 14.3.2. Clipping to the plot area

When auto scaling is used no data points will ever be outside the plot area since the scaling will make sure it covers the dynamic range of the data. However, when setting the scale manually there is always the risk that some data points are outside the specified scale. This could then lead to the result shown in Figure 14.13, “Original plot without clipping (clipping\_ex1.php)” To avoid this situation it is possible to enable clipping to the plot region by calling the method

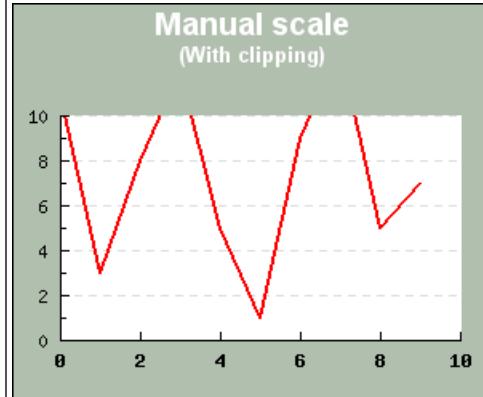
- `Graph::SetClipping($aFlg)`

This will "cut away" all parts of the plot that would otherwise be stroked outside the plot area. Figure 14.14, “Plot with clipping enabled (clipping\_ex2.php)” shows the result after clipping has been enabled.

**Figure 14.13. Original plot without  
clipping (clipping\_ex1.php)  
[example\_src/  
clipping\_ex1.html]**



**Figure 14.14. Plot with clipping enabled  
(clipping\_ex2.php)  
[example\_src/  
clipping\_ex2.html]**



### Note

By default clipping is disabled since it will slightly increase the CPU load and the time to generate the graphs.

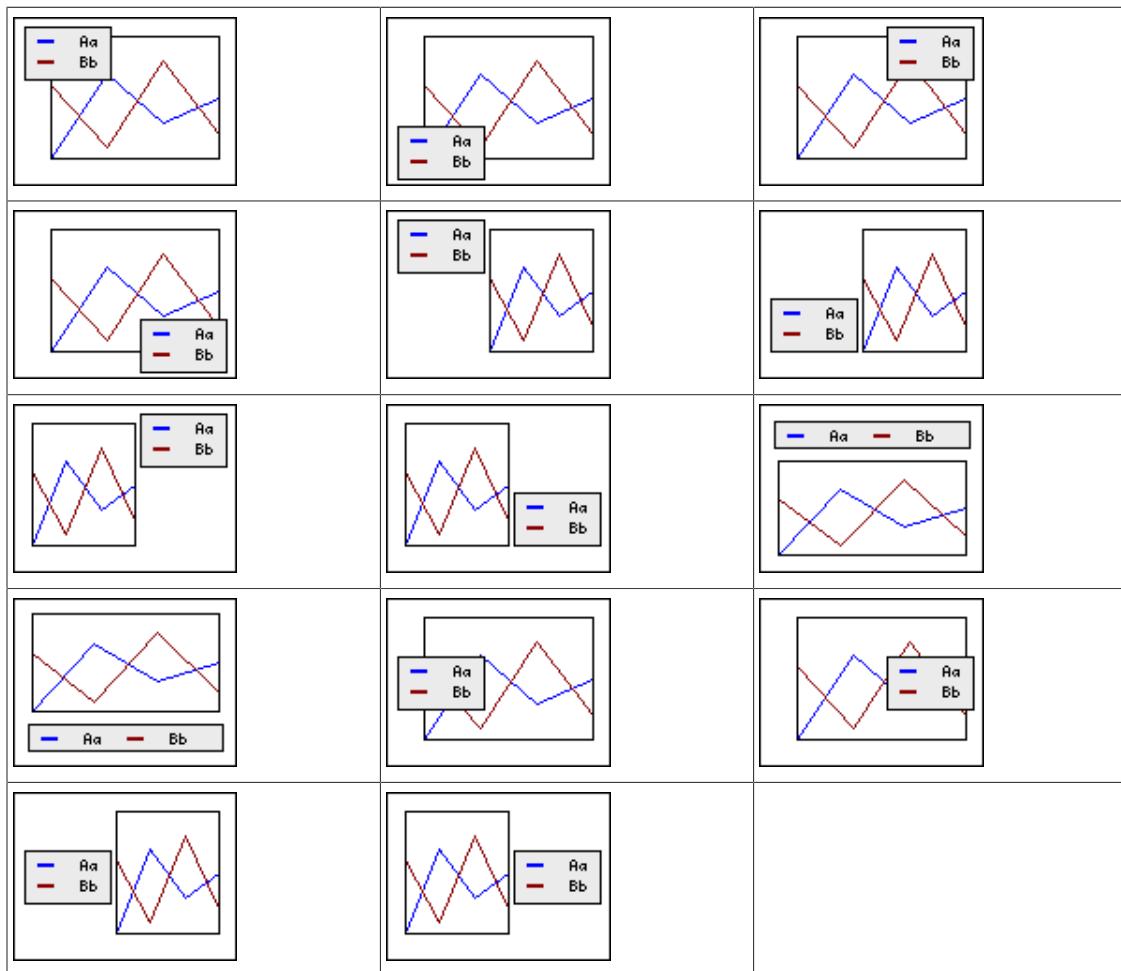
## 14.4. Adjusting the position and layout of the legend

While it is not strictly necessary to use a legend in the graph to describe the various data series plotted it is usually a good idea. With the library there are three basic types of formatting you can do with the legend.

1. Adjusting basic parameters such as color and width of the edges (frame) around the plot box as well as adding/removing a drop shadow on the legend box.
2. Adjusting the position of the legend in the graph
3. Adjusting the number of columns that should be used for the internal layout in the legend box. As default the legend uses one column.

In Figure 14.15, “Some example of ways to position the legend box in the graph” we have shown some ideas on how the legend can be positioned in the graph to give some ideas on the options available.

**Figure 14.15. Some example of ways to position the legend box in the graph**



The way the legend is positioned is by calling one of the two methods

- `Legend::SetPos($aX,$aY,$aHAlign='right',$aVAlign='top')`
- `Legend::SetAbsPos($aX,$aY,$aHAlign='right',$aVAlign='top')`

The first method specifies the position in fraction of the width and height and the second method specifies the position in absolute pixels where (0,0) is the top left corner. The anchor point for the legend box, i.e. the point in the legend box that should be aligned with the specified positoin, is given with the following two parameters. The line below gives an example of specifying the legend position

```
$graph->legend->SetPos(0.5,0.98,'center','bottom');
```

The line above will position the legend centered horizontally and the bottom of the legend just above the bottom edge of the graph.

### Tip

The legend box can also have a drop shadow. By specifying the color to be alpha-blended it is possible create a very nice drop shadow affect onto the graph. See Figure 14.2, “Using alternating fill colors in the grid (filledgridindex.php)” or Figure 14.16, “Some example of different legend layouts” for some examples on how this looks.

## Tip

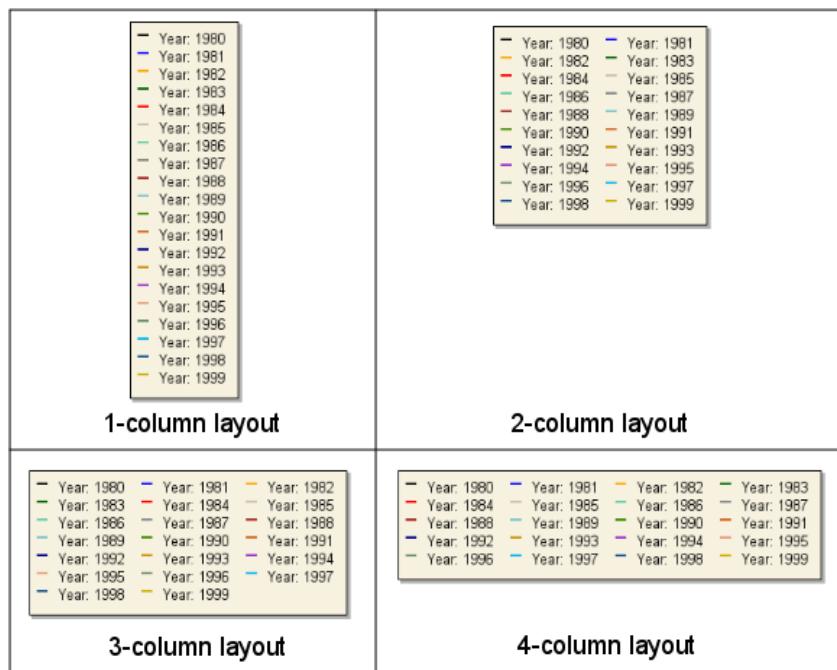
By default the ordering of the texts in the legend will be the same as the order the plots are added to the graph. It is also possible to reverse this order in the legend with a call to the method `Legend:::Revers()`

The layout of the legend box can also be adjusted by specifying how many columns the legends should use. By default one column is used which means that all legend texts will be on top of each other. The number of columns to be used in the legend is specified with a call to the method

- `Legend:::SetColumns($aNbr)`

Figure 14.16, "Some example of different legend layouts" shows four examples how setting different values for the number of columns will change the layout of the legend.

**Figure 14.16. Some example of different legend layouts**



As two special cases the one column layout (a.k.a. vertical layout) which is the default, and the "unlimited number of columns" a.k.a. horizontal layout can also be specified with the method

- `Legend:::SetLayout($aLayout)`

`$aLayout` is either `LEGEND_VERT` or `LEGEND_HOR`

## Caution

Either `SetLayout()` or `SetColumns()` should be used but not both since the method last called will be the one that is used.

## Tip

The vertical margin between two legend texts can be adjusted with a call to the method

- `Legend::SetVColMargin($aSpacing)`

by default the margin is quite small. **Note**. Since the bounding box for TTF fonts and bitmapped texts are slightly different the distance between legend texts when using a bitmap font (the default) will be visually larger than for TTF fonts. The default value is optimized for TTF fonts.

## 14.5. Other formatting options of the axis

### 14.5.1. Adjusting and positioning titles on the axis

Each title of the graph can have a title of there own. By default the x-axis has the title on the right and the y-axis have the title in vertical at the top of the graph. The title of the axis is accessed by the title property of the axis for the x- and y-axis as in

```
Graph::xaxis::title
Graph::yaxis::title
```

As a continence the following there methods are provided

- `Axis::SetTitle($aTitle, $aAdjustAlign)`

This method sets the title text string and the alignnemt of the title. For the x-axis the alignment is pecified as one of the folliwng three strings

- "left"
- "center"
- "right"

For an y-axis the alignment is set with one of the following three strings

- "bottom"
  - "middle"
  - "top"
- `Axis::SetTitleMargin($aMargin)`

The margin is specifies as the number of pixels distance from the axis to the title

- `Axis::SetTitleSide($aSideOfAxis)`

The side specifies for a horizontal axis (x-axis) if the title should be above or below the axis and for a vertical axis (y-axis) if the title should be on the left or the right of the axis.

Y-axis defines:

1. SIDE\_LEFT
2. SIDE\_RIGHT

X-axis defines

1. SIDE\_TOP

2. SIDE\_BOTTOM

### 14.5.2. Adjusting the font and color of the title

Since the title property can be easily accessed all usual Text class property can be used. This includes the color property. To set the font and color of the title for the x-axis the following methods are used

- Graph::xaxis::title::SetFont()
- Graph::xaxis::title::SetColor()

For the y- and y2-axis the fonts and colors are set completely analogues.

## 14.6. Using multiple y-axis

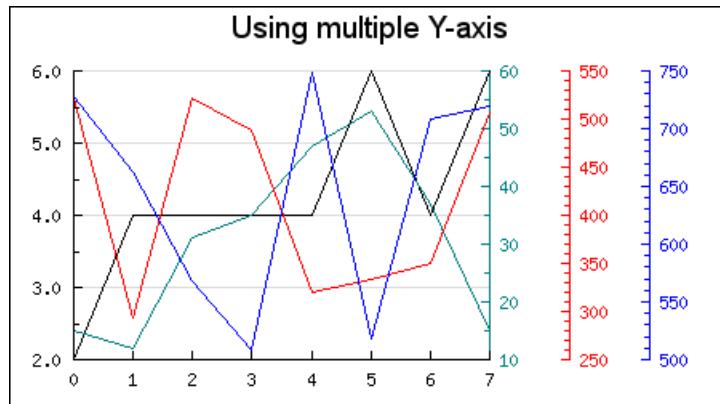
With this feature an arbitrary number of Y-axis can be added to the right side of the graph. The library itself doesn't impose any restrictions on the number of extra Y-axis but from a practical concern it is most likely very difficult to interpret a graph with more than 2-3 extra y-axis.

If there is only need for one more Y axis then the concept of the additional Y2 axis is available as a convenient shortcut for this the most common use of an extra y-axis. The Y2 axis is also a first class citizen in the library meaning it has all the properties available to the "normal" Y-axis.

These extra axis inherits most of the properties of the normal Y-axis (but not all) and the few restrictions imposed are described in Section 14.6.3, "Restrictions with multiple y-axis" below.

Figure 14.17, "Basic example of multiple y-axis (mulyaxisex1.php)" shows a basic example of how to use this feature. The color encoding maps a specific line to the corresponding axis.

**Figure 14.17. Basic example of multiple y-axis (mulyaxisex1.php)**  
[example\_src/mulyaxisex1.html]



### 14.6.1. Adding additional y-axis

Adding additional Y-axis is very similar to the way standard Y axis work. The Y-axis are numbered [0..n] where the 0:th axis is the Y-axis furthest to the left. At the same time as these additional Y-axis are used it is also possible to add a Y2 axis. The difference being that the Y2 axis can have all the same options as the Y axis.

For basic usage only three new methods are needed

1. `Graph::SetYScale($aNbr, $aScaleType, $aMin, $aMax)`

Specifies the type of scale ('lin', 'int' or 'log') to use for the axis number '\$aNbr'

2. `Graph::AddY($aNbr, $aPlot)`

Add a plot to axis number '\$aNbr'

3. `Graph::SetYDeltaDist($aDistance)`

This is an optional method that if used specifies the default number of pixels between each additional Y-axis. This value will be used unless a specific position for the N:th axis has been specified. By default the additional Y-axis are separated with 50 pixels (which is what is used in Figure 14.17, "Basic example of multiple y-axis (mulyaxisex1.php)" )

In order to initialize the extra y-axis the method `SetYScale()` must be called. In Figure 14.17, "Basic example of multiple y-axis (mulyaxisex1.php)" the following lines are used

```
$graph->SetYScale(0, 'lin');
$graph->SetYScale(1, 'lin');
$graph->SetYScale(2, 'lin');
```

Once setup these additional Y-axis are accessed through the array

```
Graph::ynaxis[]
```

The axis are numbered from 0. By accessing the axis through this array most of the same method as for the usual Y and Y2 axis are available. For example, the line below will set the color of axis number 1

```
$graph->ynaxis[1]->SetColor('red');
```

Finally the plots are added to a specific axis with a call to the method `AddY()` (as opposed to the regular `Add()` or `AddY2()` methods). The first argument must be an ordinal that specified the number of the axis that the plot should be added to.

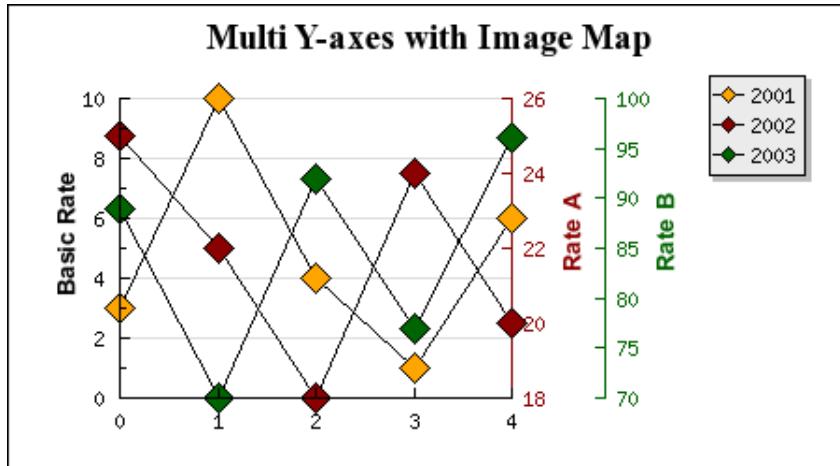
## 14.6.2. Using CSIM together with multiple y-axis

Client Side Image Maps is fully supported and is used in the same way as CSIM for the basic Y-axis. A short example will clarify this. The following code snippet shows a line plot where we have added some oversized markers (in the shape of diamonds) that will act as the image map areas for each data series

```
<?php
...
$lp2 = new LinePlot($datay2);
$lp2->mark->SetType(MARK_DIAMOND);
$lp2->mark->SetWidth(15);
$lp2->SetCSIMTargets($targ2, $alts2);
$graph->AddY(0, $lp2);
...
?>
```

As usual the targets for the image maps are specified with a call to `Plot::SetCSIMTargets()`. An example of CSIM with multiple y-axis is included in the Example directory and the resulting image is reproduced in Figure 14.18, "Illustration of mulyaxiscsimex1.php" (but just as an image not as a CSIM image)

**Figure 14.18. Illustration of mulyaxiscsimex1.php [example\_src/mulycsim\_example\_static.html]**



### 14.6.3. Restrictions with multiple y-axis

- Only standard plots can be added to the extra Y-axis, this means no Text objects, no PlotBand, no PlotLines, no Icons etc
- No Grid lines can not be added to the extra Y-axis

## 14.7. Understanding and using different scales on the axis

### 14.7.1. Different scale types

The scale of the graph axis are controlled by either

- `Graph::SetScale()` which can take one of the following strings as argument  
`"intint", "intlin", "intlog", "linint", "linlin", "linlog", "logint", "loglin", "loglog", "textint", "textlin", "textlog", "datint", "datlin", "datlog"`

The first half of the string argument specifies the x-axis scale and the second half of the string specifies the y-axis scale. When an y2 or multiple y-scales (see ??

### Numerical scale types

The scale for a basic graph is specified with a call to `Graph::SetScale()` and supplying the wanted scales for the x- and y-axis. For numeric data there are three basic scales available

1. Integer scale "int"
2. Linear scale (a.k.a decimal labels)
3. Logarithmic scales

The choice of scale will affect how the autoscaling is done and how the labels will look. Choosing an integer scale will, as the name suggests, restrict the labels to integer and integer interval between the

scale step (at each tick mark). The linear scale has no such restriction and can be considered a superset of the integer scale (even though the implementation is done in rather the opposite way). Labels for linear scale can have fraction intervals and will show the decimal values by default in the labels. Finally the logarithmic scale will create a deca - logarithmic scale.

The style of the labels can be formatted in two ways

1. By specifying a format string (in `printf()` format) to be used by calling the method `Axis::SetLabelFormatString()`. The format string is interpreted according to the second parameter. If that parameter is "true" then the format string will be assumed to give a date format as expected by the `date()` function. This is an easy way to translate timestamp values to proper time/date labels.
2. By specifying a callback function. The callback function/method will be called with the label as the only argument and must return the label that should be printed `Axis::SetLabelFormatCallback()`

For example. An easy way to get 1000' separators for numbers is to add the PHP function `number_format()` as callback to the wanted y-axis as in

```
$graph->yaxis->SetLabelFormatCallback('number_format');
```

Another example of using a callback function is to "revert" the y-axis. By default the y-axis grows from the bottom and up. But what if we want the y-axis to grow from top to bottom, i.e. the 0-value should be at the top and the largest value at the bottom?

The way to accomplish this is to use a negative y-scale which the plot is made against (by negating all the data values in the data serie). This would then give the appearance that we want apart from the fact that all labels will have a minus sign in front of them. By creating a callback function that just returns the absolute value of the label we can adjust that and we get the effect that we want. Figure 14.19, "Inverted y-scale to show a dive profile (inyaxisex2.php)" shows an example of a "dive-curve" where the sea-level is at the top of the graph to give better connection to this particular use-case.

**Figure 14.19. Inverted y-scale to show a dive profile (inyaxisex2.php)**  
[example\_src/inyaxisex2.html]



## Note

The dive profile in Figure 14.19, "Inverted y-scale to show a dive profile (inyaxisex2.php)" is an actual dive of one of the authors dives in the *Gulf of Bothnia*

## Textual scale types

There is only one pure textual scale type

1. Text scale ("text")

This type of scale is exclusively used for the x-axis.

The primary use of this scale is to label bars in a bar graph. However, as will be shown this can also be used at other occasions to achieve various wanted effects. There is no concept of autoscaling for text scales. Instead the whole purpose of this type of scale is for the user to manually supply the wanted labels in an array using the method `Axis::SetTickLabels()`.

The other key difference for a text scale as compared to the numeric scales is that the labels are positioned in between the tick marks and not directly under them. This is the common practice to label bar graphs and hence this is the way text scale works.

## Date scale types

This scale type only has one valid scale

1. Date scale ("dat")

This specific scale type will assume that the data values are timestamps and will properly format them to give "even" steps (in a time/date sense).

## 14.7.2. Manual vs automatic scale handling

Normally the scale is determined automatically in the library by analyzing the input data and making sure that a suitable scale is established that fulfill the following criteria:

1. The full dynamic scope of the data series can be displayed
2. The step size between major tick marks (tick marks that have a label) is a multiple of either (1, 2, 5, 10) \*  $10^n$ , where the size of 'n' is determined by the dynamic range of the data.
3. In addition the number of labels is dependent on the size of the graph. A smaller graph will have fewer labels since there is "physically" not enough room to show too many labels.

### Note

Internally the auto scaling algorithm has fair amount of intelligence to try to make the scale in the same way a human would do it. For example the origin has a specially strong attraction meaning that the autoscaling will try hard to make sure 0 is included in the scale as long as it "makes sense".

Usually this works fine in most cases but there are always exception where exact control over the scale is wanted. For example to be able to compare several graphs that might otherwise get different scales. There are two ways of manually adjusting the scales.

#### 1. Manual min/max values but with the tick marks automatically determined

In this case the exact min/max value of the scale is submitted to the `SetScale()` method call to set either (or both) of the x- and y-scale. Since the first two argument after the scale type determines the y min/max and the fourth and fifth argument specifies the x min/max this means that in order to specify the x

scale the y scale must have some values. If the y axis should be left alone (i.e. to be autoscaled) then just put the "dummy" values (0,0) as placeholder in the method call. The following examples clarifies this.

- `$graph->SetScale('intlin',0,0,-10,20);`  
Y-scale is autoscaled and the x-scale is set to [-10,20]
- `$graph->SetScale('intlin',0,50);`  
Automatic x-scale and manual y-scale with range [0,50]
- `$graph->SetScale('intlin','-10,10,-20,20);`  
Manual x- and y-scale

## 2. Semi automatic (or semi manual)

This is a away to lock either the min or max value and let the auto scaling algorithm determine a suitable corresponding max and min value. This can be very useful to make sure that for example the 0 value is always included even if the minimal value is so high that the autoscaling algorithm have chosen to start at a larger value. These two locked down values are set with the two methods

- `Scale::SetAutoMin()`  
Example: `$graph->yaxis->scale->SetAutoMin(0);` This will lock the y-min to 0
- `Scale::SetAutoMax()`  
Example: `$graph->xaxis->scale->SetAutoMax(1000);` This will lock the x.max to 0

## 14.7.3. Major and minor ticks

The final step in understanding the scaling is to understand tick marks. There are two types of tick marks, major and minor. Major tick marks are tick marks that have a label associated. Up to now we have assumed that the library adjusts where the tick marks are positioned and the inter spacing between tick marks.

The tick object is available as an instance variable of the scale class which means that you can access and adjust the tick mark properties by accessing:

`$Graph::Axis::Scale::ticks`

For further fine control of the scale it is possible to manually adjust the major and minor ticks using one of the three methods

### 1. Adjusting how dense the auto scaling algorithm should put the tick marks.

Even when using a fully automatic scaling it is possible to adjust how many tick marks that the algorithm should try to place on the scale. This is controlled with the method

- `Graph::SetTickDensity($aYDensity, $aXDensity)`

The y- and x-scale density can be set to one of the following (symbolic) values

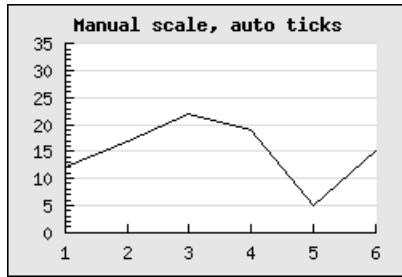
- `TICKD_DENSEK`
- `TICKD_NORMAL`
- `TICKD_SPARSE`

- TICKD\_VERYSPARSE

By default the density for all scales are set to TICKD\_NORMAL

As a comparison Figure 14.21, “Fully automatic not so good scaling (`manscaleex2.php`) ” is the exact same as Figure 14.20, “Setting tick density to TICKD\_DENSE (`manscaleex3.php`) ” with the difference that in this figure the density for the y-scale has been set to TICKD\_DENSE

**Figure 14.20. Setting tick density to TICKD\_DENSE (`manscaleex3.php`)**  
[[example\\_src/manscaleex3.html](#)]



## 2. Manually specifying the step size between each minor and major tick.

This is done using the method

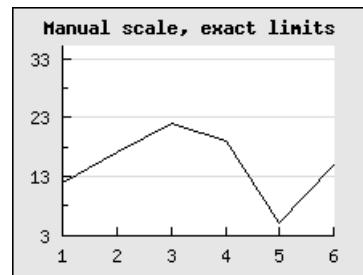
```
LinearTicks::Set($aMajStep,$aMinStep=false)
```

please note that this is a method of the Tick class which is available as an instance variable in the scale class

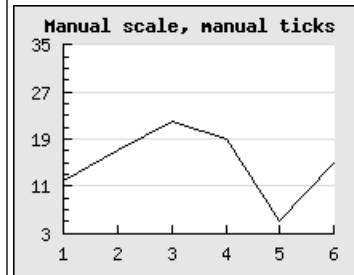
```
Example: $graph->xaxis->scale->ticks->Set(20,5);
```

This type of manual positioning of the tick marks might be useful if the scale has been set to some non-multiples of 1,2,5,10. The following examples will clarify this. In ?? the left graph is what we get after manually specifying the y min/max values to (3,35). In this case the auto scaling algorithm fails to assign "nice" steps to the y-axis so that the tick marks starts and begins at the min/max of the scale. If we manually set the tick distance to be (8,2), i.e. we set the major tick marks 8 units apart and the minor (non-labeled) tick marks 2 steps apart we get the much nicer result as shown in the right graph in ??

**Figure 14.21. Fully automatic not so good scaling (`manscaleex2.php`)**  
[[example\\_src/manscaleex2.html](#)]



**Figure 14.22. Manually specified tick distance which gives a much better appearance (`manscaleex1.php`)**  
[[example\\_src/manscaleex1.html](#)]



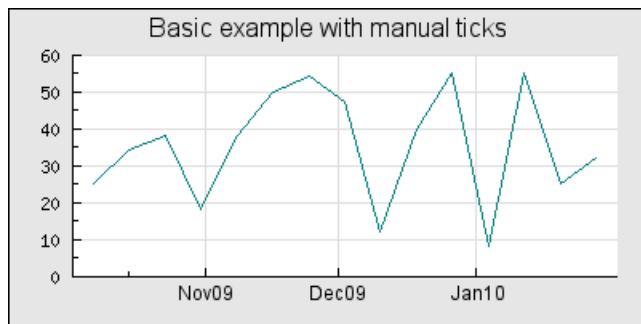
## Caution

The major tick step must be an even multiple of the minor tick step. If this is not the case the library will automatically adjust the major step size to be the closest multiple of the minor step size.

### 3. Specify the exact position for each single tick mark

This is often used to position only the major tick marks to be able to have labels at very specific points in the graph. One example of how to use this was shown in Figure 8.8, “Specifying manual ticks as fraction of Pi. (manualtickex2.php) ” where the tick marks were positioned at fractions of #. Another typical example is to place a label at exactly the beginning of a month. Since months have different lengths there is no other way if 100% precision is wanted. An example of this is shown in Figure 14.23, “Manually specifying the tick position for each month (manualtickex1.php) ”

**Figure 14.23. Manually specifying the tick position for each month (manualtickex1.php) [example\_src/manualtickex1.html ]**



## Caution

When the tick position is manually set there should also be a corresponding array of labels to be put at these positions.

### 4. Adjusting the size and on what side the tick marks should be drawn

The side on the axis which has the tick marks is adjusted with a call to

- Axis::SetTickSide(\$aTickSide)

Possible options for the tick side are

- SIDE\_UP
- SIDE\_DOWN
- SIDE\_LEFT
- SIDE\_RIGHT

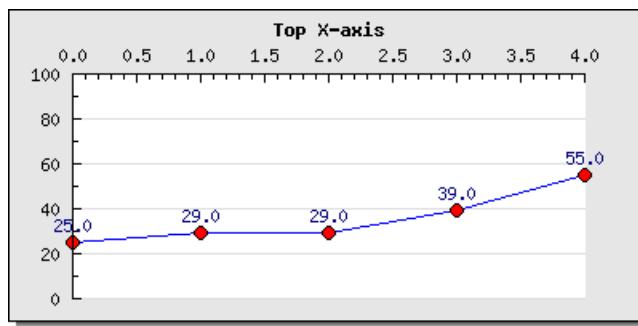
For example, the following lines added to a script would change side of the labels and tick marks for the x-axis.

```
<?php
```

```
$graph->xaxis->SetLabelPos(SIDE_UP);
$graph->xaxis->SetTickSide(SIDE_DOWN);
?>
```

This technique can for example be used to put the x-axis at the top of the graph as the following example shows.

**Figure 14.24. Adjusting the side which have the tick marks and position the x-axis at the top (`topxaxisex1.php`) [`example_src/topxaxisex1.html`]**



In passing we mention that there are additional ways to adjust the look and feel of the tick marks which is discussed in the next section (Section 14.8, “Adjusting the appearance of the scale labels”) in detail.

## 14.8. Adjusting the appearance of the scale labels

Since scale labels play such an important part of producing a visually pleasing graph there are a number of ways to fine tune this appearance. In the section below we have tried to logically group the different options to make it easier to get an overview.

### 14.8.1. Adjusting the position

The exact position of the labels can be adjusted with the methods

- `Axis::SetLabelAlign($aHAlign,$aVAlign='top',$aParagraphAlign='left')`

With this method the anchor point for the labels can be adjusted. The labels are positioned straight below (or on the side for vertical axis) the corresponding tick mark and by adjusting the anchor point in the label different effect can be achieved. By default labels on the x-axis have an anchor point on the top side in the center. For y-axis (on the left) the anchor point is middle of the right edge of the label.

- `Axis::SetLabelMargin($aMargin)`

This method adjusts the distance between the corresponding tick mark and the label itself.

- `Axis::SetLabelSide($aSidePos)`

This method will adjust on what side the labels will be positioned. For a horizontal axis it can be either above or below the axis and for a vertical axis it can be on the left or right side. The position is defined by one of the following self-explaining defines

- `SIDE_LEFT`

- SIDE\_RIGHT
- SIDE\_TOP
- SIDE\_BOTTOM

## 14.8.2. Adjusting font and color

By default the color of the labels will be the same as for the axis but it is possible to use a different color for the labels by supplying a second argument to the `SetColor()` method for the axis.

- `Axis::SetColor($aColor, $aLabelColor=false)`

There is actually a special twist to the color specification here. If the color is specified as a single value this color value will be used for all labels. However, if the color is specified with an array then those colors will be used for the labels in turn. If there are more labels then colors specified then the colors will be used in a wrap around fashion.

The font family for the labels can be specified with

- `Axis::SetFont($aFamily, $aStyle=FS_NORMAL, $aSize=10)`

By default a bitmap font is used for the labels (since those are the only fonts guaranteed to be available everywhere)

### Example 14.1. Adjusting the axis font and color

```
$graph->xaxis->SetFont(FF_ARIAL, FS_NORMAL, 9);
$graph->xaxis->SetColor('black', 'blue');
```

## 14.8.3. Adjusting the background of the labels

As a kind of a very special formatting it is also possible to set a specific background for just the labels on the graph. This is most often used when there is background image that risks of hiding the clarity of the labels. In those cases it can be suitable to have a some lighter background to make it easier to read the labels.

### Tip

This feature can also come in handy when using the more advanced feature of combining several graphs to amke the labels stand out.

The method that controls this is

- `SetAxisLabelBackground($aType, $aXFCOLOR='lightgray', $aXCOLOR='black', $aYFCOLOR='lightgray', $aYCOLOR='black')`

Possible values of the type of background are:

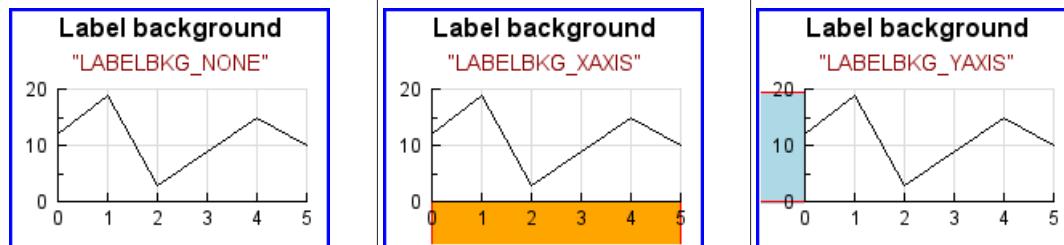
1. LABELBKG\_NONE
2. LABELBKG\_XAXIS
3. LABELBKG\_YAXIS
4. LABELBKG\_XAXISFULL
5. LABELBKG\_YAXISFULL

## 6. LABELBKG\_XY

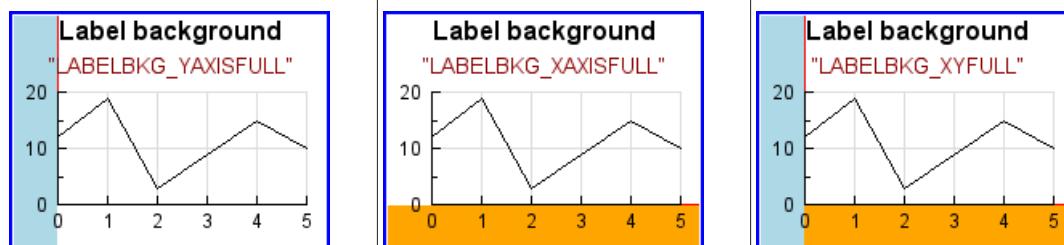
## 7. LABELBKG\_XYFULL

The way these possible background types works is best described by viewing the example figures below

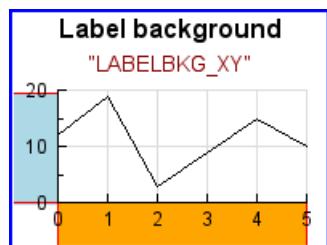
**Figure 14.25** LABELBKG\_NONE | **Figure 14.26** LABELBKG\_XAXIS | **Figure 14.27** LABELBKG\_YAXIS  
(axislabelbkgex01.php) (axislabelbkgex02.php) (axislabelbkgex03.php)  
[example\_src/  
axislabelbkgex01.html] [example\_src/  
axislabelbkgex02.html] [example\_src/  
axislabelbkgex03.html]



**Figure 14.28** LABELBKG\_YAXISFULL | **Figure 14.29** LABELBKG\_XAXISFULL | **Figure 14.30** LABELBKG\_XYFULL  
(axislabelbkgex04.php) (axislabelbkgex05.php) (axislabelbkgex06.php)  
[example\_src/  
axislabelbkgex04.html] [example\_src/  
axislabelbkgex05.html] [example\_src/  
axislabelbkgex06.html]



**Figure 14.31** LABELBKG\_XY  
(axislabelbkgex07.php)  
[example\_src/  
axislabelbkgex07.html]



## 14.8.4. Hiding and rotating labels

For the x-axis it is common to rotate the label to better fit long text labels. This is done with a call to

- Axis::SetLabelAngle(\$aAngle)

Remember that in order to set an arbitrary angel on the labels the font family must be one of the TTF font families. Bitmap fonts only support horizontal and vertical rotation.

There are actually five variant for hiding the aaxis

- `Axis::HideLabels($aHide=true)`

This will hide all the labels on the axis

- `Axis::HideFirstLastLabels($aHide=true)`

Will hide the first and last label

- `Axis::HideFirstLabels($aHide=true)`

Will hide the first label

- `Axis::HideLastLabels($aHide=true)`

Will hide the last label

- `Axis::HideZeroLabel($aHide=true)`

This will avoid printing a "0.00" label. This can be useful when the axis are set to be crossing at the origin. This was for example used in Figure 14.5, “Example of AXSTYLE\_BOXIN axis style (funcex2.php)”

## 14.8.5. Fine tuning the automatic scales

By default the auto-scaling algorithm tries of find as tight upper and lower bands as possible as long as they are even multiple of the chosen major step size. In some instances some more space is needed. For example to have extra room at the top of the graph to show data labels. In order to have some extra space the method

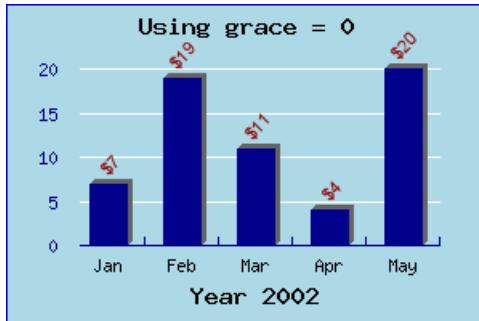
- `LinearScale::SetGrace($aGraceTop,$aGraceBottom=0)`

should be used. This method will adjust the autos calling algorithm by adding the specified extra "grace"at the minimum and maximum value of the scale. The grace value is specified as an integer [0,100] and specifies a percentage to add to the min and max value before the auto scaling is made. Depending on the scale step and overall scale several scale values might give the same resulting scale. For example

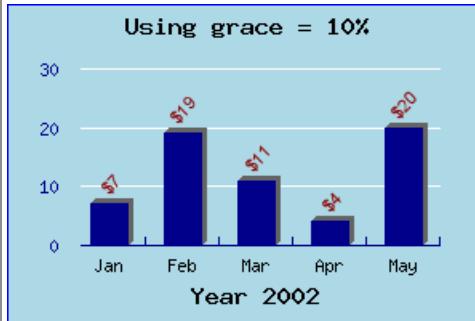
```
$graph->yscale->scale->SetGrace(20,20);
```

will add 20% extra values on the top and bottom values of the data series before the auto scale is determined. The following four examples shows how this can be used in practice. In Figure 14.32, “The original graph (grace\_ex0.php)” a bar graph without grace value is created. As can be seen since the scale is very tight the labels of the individual bars are almost out of the graph plot area. In the following three plots results of using different values for grace are shown.

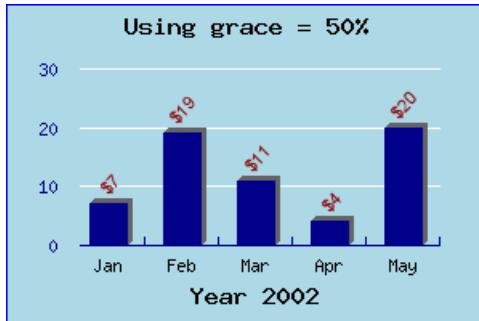
**Figure 14.32.** The original graph  
(grace\_ex0.php)  
[example\_src/grace\_ex0.html]



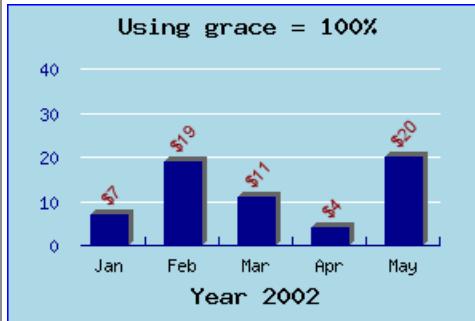
**Figure 14.33.** Using a 10% grace  
(grace\_ex1.php)  
[example\_src/grace\_ex1.html]



**Figure 14.34.** Using a 50% grace  
(grace\_ex2.php)  
[example\_src/grace\_ex2.html]

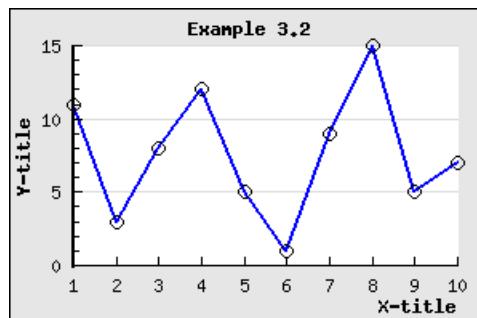


**Figure 14.35.** Using a 100% grace  
(grace\_ex3.php)  
[example\_src/grace\_ex3.html]



We finish this section by noting that adding a large grace value might sometimes make it necessary to move one or both of the axis manually. To explain this we start with the graph in Figure 14.36, “The original line graph (example3.2.php) [example\_src/example3.2.html]

**Figure 14.36.** The original line graph (example3.2.php) [example\_src/example3.2.html]

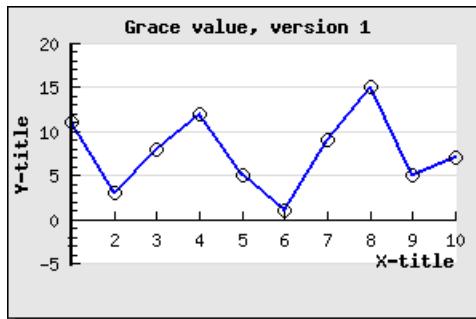


as can be seen the added plot marks (see ??) are outside the boundary of the plot area and to solve this problem we add the line

```
$graph->yaxis->scale->SetGrace(10 , 10);
```

to the script. This now gives the result shown in Figure 14.37, “Adding grace values. Note x-axis position at y=0 (example3.2.1.php)”

**Figure 14.37. Adding grace values. Note x-axis position at y=0 (example3.2.1.php) [example\_src/example3.2.1.html]**

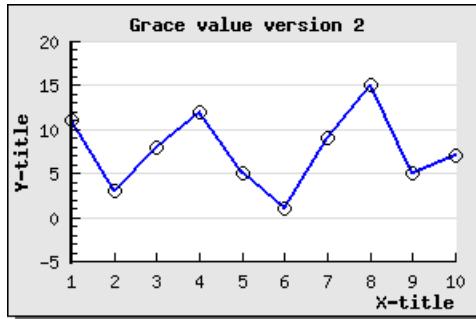


The problem is now apparent. Since the x-axis is placed at y=0 by default the x-axis is no longer at the bottom of the plot as was probably the intention. To remedy this situation we have to manually place the x-axis at the minimum y-scale position. This is done by adding the line

```
$graph->xaxis->SetPos('min');
```

to the previous script. Doing this gives us back the wanted position of the x-axis as shown in Figure 14.38, “Adjusting the position of the x-axis manually (example3.2.2.php)”

**Figure 14.38. Adjusting the position of the x-axis manually (example3.2.2.php) [example\_src/example3.2.2.html]**



## 14.8.6. Manually altering the appearance of tick marks

To adjust the appearance of the tick marks the following three methods can be used

- Axis::SetTickSide(\$aSide)

For x-axis this specifies if the tick should be drawn above or below the axis and for the y-axis it specifies if the tick marks should be on the left or on the right of the axis. Possible values of the side parameter are one of the following defines

- SIDE\_LEFT
- SIDE\_RIGHT
- SIDE\_TOP

- SIDE\_BOTTOM
- Axis::HideTicks(\$aHideMinor=true,\$aHideMajor=true)

Calling this method will make it possible to hide either noe of or both the minor or the major tick marks.

- Axis::SetTickSize(\$aMajSize,\$aMinSize=3)

This specifies the size (in pixels) of the major and minor tick marks.

## 14.8.7. Manually specifying scale labels

As was mentioned before it is perfectly possible to manually specify the scale labels. This is normally done for "text" scales that are most commonly used with bar graphs.

Specifying a text scale for the axis (normally the x-axis) will "reserve" a position for every data value that must be manually set. In some ways this is the same as having a linear scale with major and minor step size set to 1.

As a very basic example of a text scale with a bar graph Figure 14.40, "Manual text scale with month labels (manual\_textscale\_ex1.php)" shows some fictive values for each month of the year. In that example the scale has been set to a text scale and each value has been specified as an array containing all the names of the months.

### Tip

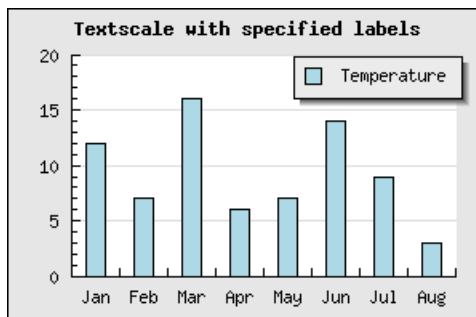
An easy way to get hold of the names of the months is to use the library global instance \$gDate-Locale of class DateLocale. This class has several utility methods. For example the two methods

- GetShortMonthName(\$aNbr)
- GetLongMonthName(\$aNbr)

will return the proper month names (in either short or short or long format) in the current locale. For more details see Figure 14.40, "Manual text scale with month labels (manual\_textscale\_ex1.php)"

**Figure 14.39. Manual text scale example**

**Figure 14.40. Manual text scale with month labels  
(manual\_textscale\_ex1.php)  
[example\_src/  
manual\_textscale\_ex1.html]**



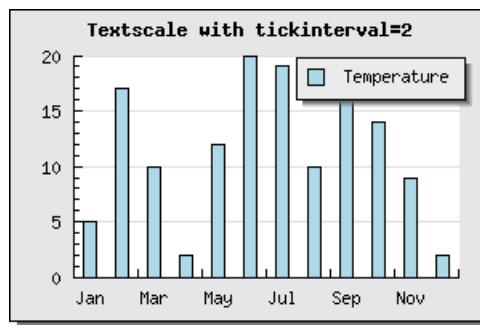
Since the manual text scale sets a scale label for every single data point this would give problem for the case where we have a large number of data points where it is not possible to label every data point. In order to control this the following two methods are used

- `Axis::SetTextTickInterval($aStep, $aStart=0)`

This method controls the interval between the tick marks for a text scale. Since by default every tick will have a label it also specifies how dense the labels will be. The following examples clarifies how this works

### Example 14.2. Adjusting manual text tick interval

**Figure 14.41. Setting text tick interval=2 (manual\_textscale\_ex2.php)  
[example\_src/manual\_textscale\_ex2.html]**



In the above example only ever second text tick mark is set and since the labels are also placed at every tick marks only every second label are shown.

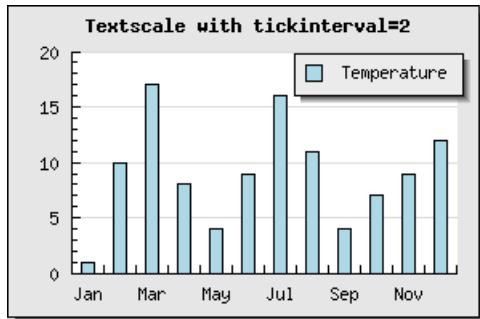
- `Axis::SetTextLabelInterval($aStep)`

This method complements the previous method. This method exclusively determines where the labels are place. This makes it possible to have more tick marks than labels. The previous example Figure 14.41, “Setting text tick interval=2 (manual\_textscale\_ex2.php)” look a bit strange since there are no tick-marks between every second bar. A better solution in this case would be to have tick marks between each bar but only label every second tick mark. This is exactly what can be done with this method. It specifies that every n:th tick mark should have a label.

In Figure 14.42, “Labels at every 2:nd tick mark (manual\_textscale\_ex3.php)” we have used this method instead.

### Example 14.3. Adjusting the interval for the labels

**Figure 14.42. Labels at every 2:nd tick mark (`manual_textscale_ex3.php`)**  
[[example\\_src/manual\\_textscale\\_ex3.html](#)]



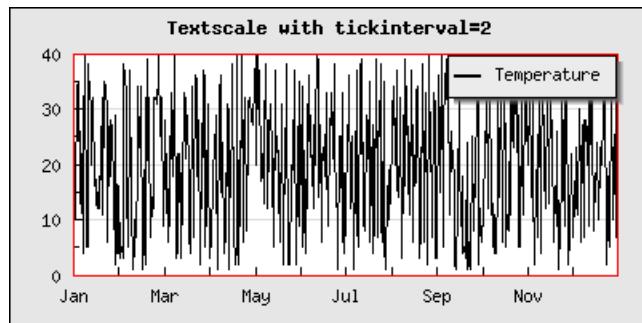
Finally we show an example where there is a need to combine the above two methods to get a reasonable looking scale. This happens when there are so many data points that it is not possible to show every tick marks. To show that text scales can also be used with other plot types than bar we have chosen to show a line graph where we show data that has 40 data points per month (480 data points per year). Let us assume that in this application we want to have tick marks at every 40 points (i.e. a total of 12 tick marks) and have labels only on every 2:th tick label (i.e. a total of 6 labels).

### Caution

The label array must have labels for every tick mark even if they are not shown.

### Example 14.4. Adjusting both text tick mark and label interval

**Figure 14.43. Tick marks every 40 points and labels every 2:nd tick mark (`manual_textscale_ex4.php`)**  
[[example\\_src/manual\\_textscale\\_ex4.html](#)]



In addition to the text scale is it in principle also possible to use a manual scale with integer scale as shown in Section 4.2, “Graphing the number of sun spots during the 19th Century”

### Note

It is also perfectly legal to override the default labels for the Y (and Y2) axis in the same way as we have shown for the x-axis but this is probably a less used scenario.

## Caution

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 the ordinal number of the label.

### 14.8.8. Emphasize of parts of the scale

Sometimes it is a good idea to emphasize part of the scale. It can be on either the x- or the y-axis. The library offers the possibility to add a band with a separate background color for part of the scale in either x, y (or both) directions.

A plotband is defined as an instance of class `PlotBand` that is added to the graph in the same way as normal plots. A plot band can have a number of different styles to achieve the wanted affect and it can be placed either on top or behind the plot.

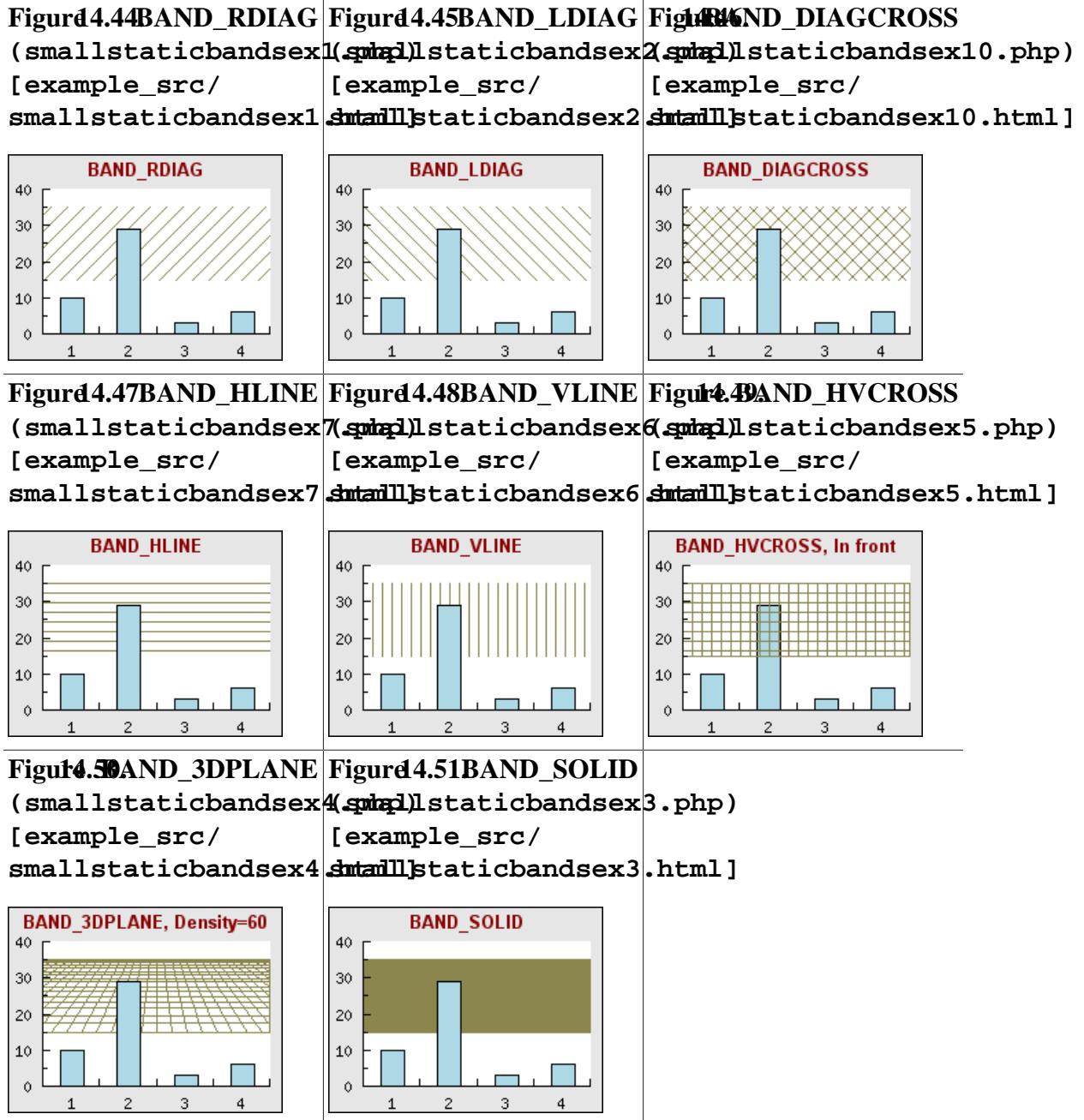
An instance is created and added to the graph as shown by the following example

```
// Add a horizontal band
$band = new PlotBand(HORIZONTAL,BAND_RDIAG,15,35,'khaki4');
$band->ShowFrame(false); // No border around the plot band
$graph->Add($band);
```

The code snippet above adds a horizontal band with a diagonal pattern (from top left to bottom right). The horizontal band is shown between the y-scale values [15,35].

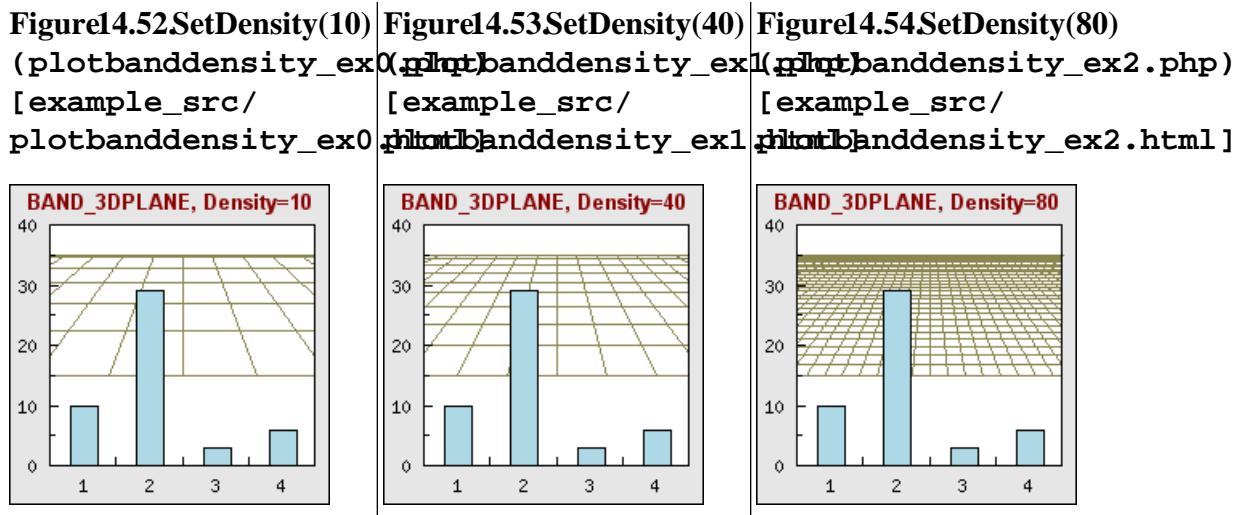
The types of plotbands supported by the library is shown in the table below

### Example 14.5. Different densities for plot bands



For some of these patterns it is also possible to specify an additional density parameter via the method `PlotBand::SetDensity()` that specifies how dense the pattern should be. For example Example 14.6, “Different densities for various plot bands” shows three variants of the 3D plane with different density parameters.

### Example 14.6. Different densities for various plot bands



#### Tip

3D planes actually carry another possible modification. You can specify the vanish point to change the perspective used. You can't access the method to change the horizon directly but you can access it through

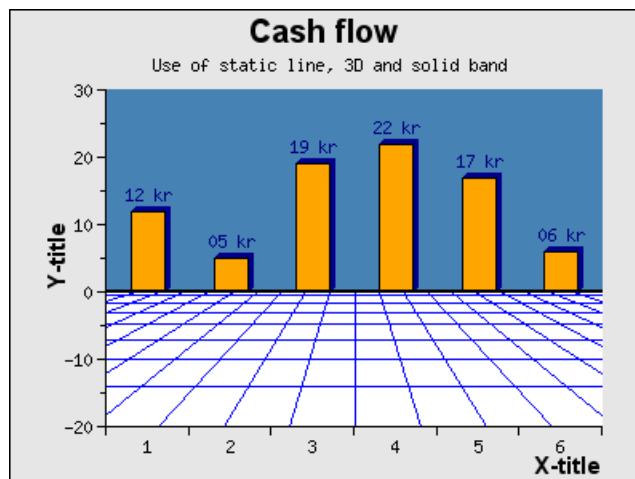
```
$band ->prect->SetHorizon($aHorizon)
```

assuming that the band is a 3D plane.

To finish this section we show one slightly more creative use of background plot bands in Figure 14.56, “Creative use of plot bands (staticbandbarex7.php)”

### Figure 14.55. Creative use of plot bands

**Figure 14.56. Creative use of plot bands (staticbandbarex7.php)**  
[\[example\\_src/staticbandbarex7.html\]](#)



## 14.8.9. Adding static lines for specific scale values in the graph

In addition to the plot band it is possible to add static lines to emphasize a specific scale value. A plot line is added to the graph by creating an instance of the class `PlotLine` in much the same way as for plot bands. The following code snippets show how this can be done

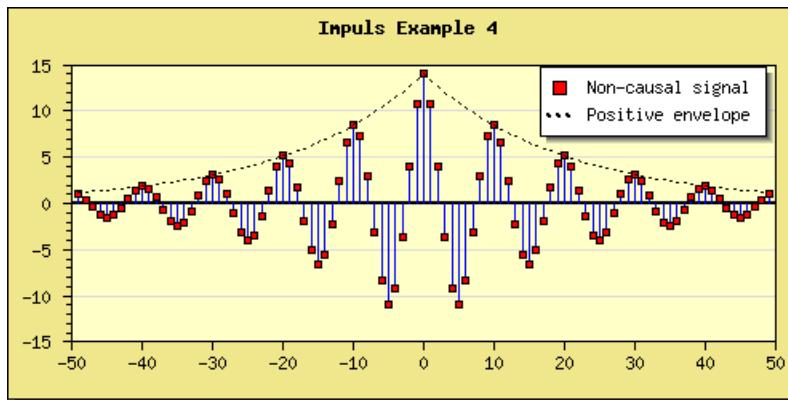
```
// Add mark graph with static lines
$line = new PlotLine(HORIZONTAL,10,"black",2);
$graph->AddLine($line);
```

The above code snippets adds a black plot line (weight=2) at scale position y=10.

In addition to using this feature to show min/max values (or perhaps critical limits) another usage can be to show a zero-line as the following example in Figure 14.58, “Adding a static line at y=0 to simulate an extra 0-axis (`impulsex4.php`)” shows.

**Figure 14.57. Use of a static line to simulate an extra x-axis at y=0**

**Figure 14.58. Adding a static line at y=0 to simulate an extra 0-axis (`impulsex4.php`) [example\_src/impulsex4.html]**



## 14.9. Using a logarithmic scale

In all the previous discussion we have used linear scales. Another option is to use a logarithmic scale. This is often used when there is a need to display both very small and very large values in the same graph. The library supports the use of logarithmic scales on both the x- and y-axis (as well as the optional extra y-axis).

The logarithmic scale support is defined in the module file "`jpgraph_log.php`" so this must be included in order to access this feature.

To illustrate the use of a logarithmic scale we will make a graph which uses two y-axis and make the second y-axis have a logarithmic scale (we will actually take the graph in Figure 15.16, “Adding and adjusting the position of the legend box (`example6.php`)” and change the second y-axis to be a logarithmic scale instead).

In order to use a logarithmic scale on the second (Y2) axis we only need to change

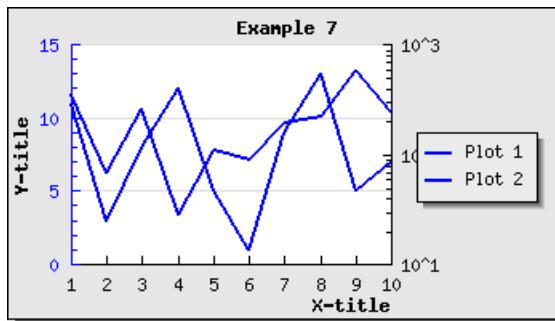
```
$graph->SetY2Scale('lin');
```

to

```
$graph->SetY2Scale('log');
```

and the library will take care of the rest. The result of this is shown in Figure 14.59, “Changing the Y2 scale from linear to logarithmic (example7.php)”

**Figure 14.59. Changing the Y2 scale from linear to logarithmic (example7.php) [example\_src/example7.html]**

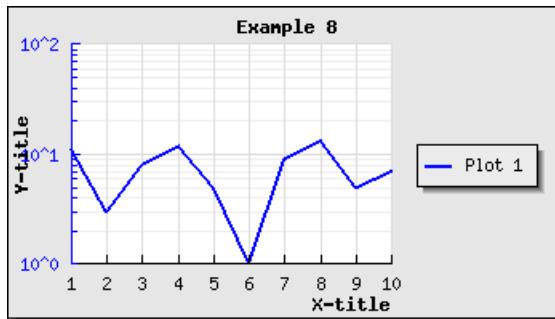


By default only the major grid lines on the y-axis are shown. By adding the lines

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

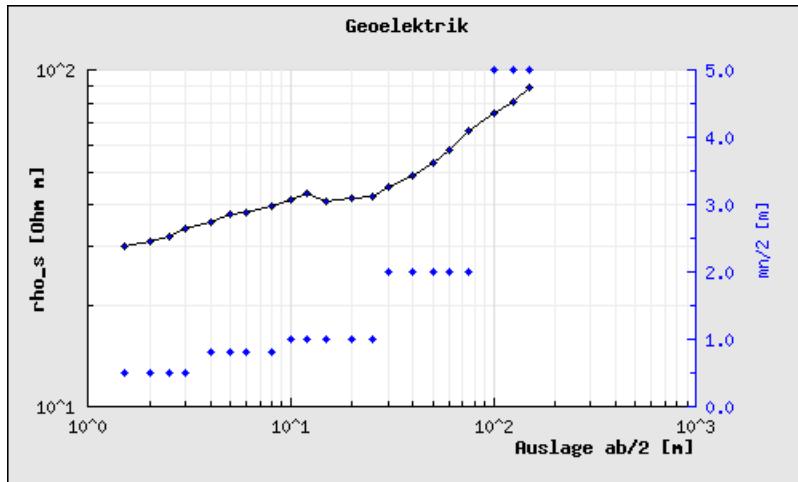
We can enable both the x-grid lines as well as the minor grid lines on the y-axis to get the result shown in Figure 14.60, “Enabling minor grid lines on the y-axis and also grid lines on the x-axis (example8.php)” below

**Figure 14.60. Enabling minor grid lines on the y-axis and also grid lines on the x-axis (example8.php) [example\_src/example8.html]**



In addition to using a logarithmic scale on the y-axis to generate what is commonly known as a "lin-log" scale it is also possible to use a logarithmic scale on the x-axis to get a "log-log" plot (as is often used in electrical engineering). An example of this is shown in Figure 14.61, “An example of a log-log plot (where both the y- and x-axis use a logarithmic scale) (loglogex1.php)”

**Figure 14.61. An example of a log-log plot (where both the y- and x-axis use a logarithmic scale) (loglogex1.php) [example\_src/loglogex1.html]**



## Note

The example in Figure 14.61, “An example of a log-log plot (where both the y- and x-axis use a logarithmic scale) (loglogex1.php) ” also makes use of scatter plots which we have not yet introduced but is a way to draw a plot of a number of data points specified by both there x- and y-coordinates.

## Tip

If you think the first value of the Y-axis is to close to the first label of the X-axis you have the option of either increasing the margin (with a call to `SetLabelMargin()`) or to hide the first label (with a call to `Axis::HideFirstTickLabel()`)

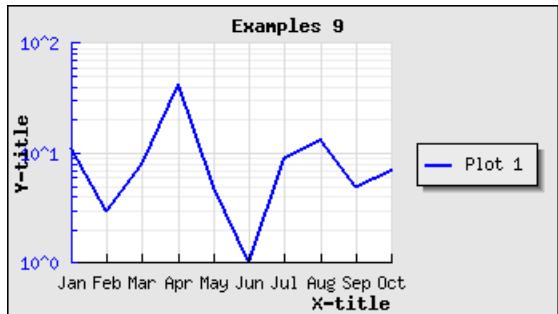
As a final example of using a logarithmic scale we show how to combine a text scale for the x-axis and a logarithmic scale for the y-axis. As we already shown in Section 4.2, “Graphing the number of sun spots during the 19th Century” it is possible to manually specify the labels that should be used on the axis with a call to `Axis::SetTickLabels()`. Let's make use of this and the built-in library super global variable “\$gDateLocale” which is an instance of a date utility class to get a list of the name of all the months in a localized fashion.

To use the name of the months on the x-axis we haev to add the following two lines to the previous example

```
<?php
$montnames = $gDateLocale->GetShortMonth();
$graph->xaxis->SetTickLabels($montnames);
?>
```

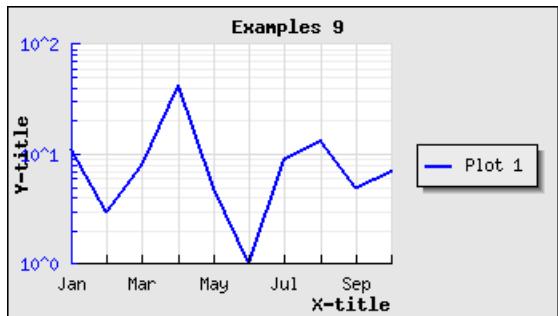
The result of adding these two lines are shown in Figure 14.62, “Using a text-log scale (example9.php) ” below

**Figure 14.62. Using a text-log scale (`example9.php`) [`example_src/example9.html`]**



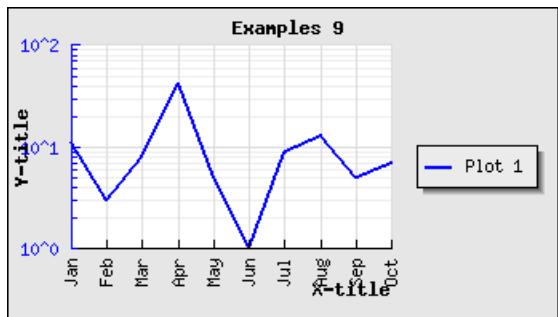
Since the name of the months are quite close we can change what labels should be displayed as was discussed in Section 14.8.7, “Manually specifying scale labels” to only show every second label. Doing this change gives the result shown in Figure 14.63, “Adjusting the text scale so that only every second labels are displayed. (`example9.1.php`)”

**Figure 14.63. Adjusting the text scale so that only every second labels are displayed. (`example9.1.php`) [`example_src/example9.1.html`]**



Another option if we think that the labels are too close is to rotate them. In Figure 14.64, “Rotating the x-axis labels 90 degree (`example9.2.php`)” we have kept all labels but rotated them 90 degree.

**Figure 14.64. Rotating the x-axis labels 90 degree (`example9.2.php`) [`example_src/example9.2.html`]**



## Caution

Remember that the built-in bitmap fonts only supports 0 and 90 degree text. TTF fonts support texts at an arbitrary angle.

## 14.10. Using a date/time scale

The easiest way to get a date time scale for the X-axis is to use the pre-defined "dat" scale. To be able to use that it is first necessary to include the module "jpgraph\_date.php" and then specify the scale, for example as "datlin" in the call to `Graph::SetScale()` as the following code snippet shows.

```
<?php
require_once("jpgraph/jpgraph.php");
require_once("jpgraph/jpgraph_line.php");
require_once("jpgraph/jpgraph_date.php");
...
$graph = new Graph (...);
$graph->SetScale('datlin');
...
?>
```

It is possible to use a selectable degree of automation when dealing with date scales. The easiest way is to let the library deal with all the details and make its own decision on how to format the scale. Unfortunately this might not always be exactly what was intended and hence it is also possible to give the library hints on how the formatting should be done or specify to a great detail the exact formatting needed.

Both the y- and the x-axis can have a date scale but the most common case is to only use date scale on the x-axis.

### 14.10.1. Specifying the input data

No matter how the formatting is done the input data is assumed to be a timestamp value, i.e. the number of seconds since epoch (as defined on the local system executing the graph). In PHP the current timestamp value is returned by the function `time()`.

This means that it is always mandatory to specify two input vectors for a plot. One vector for the Y-data and one vector for the x-data. The following line of code prepares a line plot for the use with a date scale

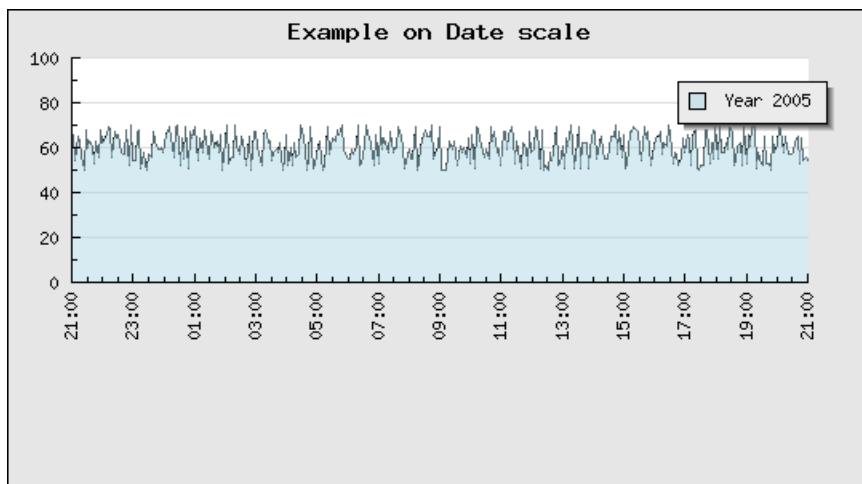
```
$lineplot = new LinePlot($ydata , $xdata);
```

#### Caution

Note the order of the data vectors. The Y-data always comes first.

A first example is shown in Figure 14.65, “A first date scale example (dateaxisex2.php)” where we have created a number of random points with a sample interval of 240s (chosen completely arbitrary)

**Figure 14.65. A first date scale example (`dateaxisex2.php`) [example\_src/  
`dateaxisex2.html`]**



Please review the script that creates this graph before continuing since we will base all further examples on this.

## 14.10.2. Adjusting the start and end date alignment

As can be seen from the example in Figure 14.65, “A first date scale example (`dateaxisex2.php`)” the scale starts slightly before the first data point.

Why?

This is of course by purpose in order to make the first time label start on an "even" value, in this case on an hour. Depending on the entire interval of the graph the start value will dynamically adjust to match the chosen date/time interval, this could for example be on an even minute, even 30min, even hour, even day, even week and so on. It all depends on the scale.

The alignment of the start (and end) date can also be adjusted manually by using the two methods

- `DateScale::SetTimeAlign($aStartAlign, $aEndAlign)`

The following symbolic constants can be used to define the time alignment

1. Alignment on seconds

- `MINADJ_1`, Align on a single second (This is the lowest resolution)
- `MINADJ_5`, Align on the nearest 5 seconds
- `MINADJ_10`, Align on the nearest 10 seconds
- `MINADJ_15`, Align on the nearest 15 seconds
- `MINADJ_30`, Align on the nearest 30 seconds

2. Alignment on minutes

- `MINADJ_1`, Align to the nearest minute
- `MINADJ_5`, Align on the nearest 5 minutes

- MINADJ\_10, Align on the nearest 10 minutes
- MINADJ\_15, Align on the nearest 15 minutes
- MINADJ\_30, Align on the nearest 30 minutes

3. Alignment on hours

- HOURADJ\_1, Align to the nearest hour
- HOURADJ\_2, Align to the nearest two hour
- HOURADJ\_3, Align to the nearest three hour
- HOURADJ\_4, Align to the nearest four hour
- HOURADJ\_6, Align to the nearest six hour
- HOURADJ\_12, Align to the nearest tolw hour

• DateScale::SetDateAlign(\$aStartAlign,\$aEndAlign)

The following symbolic constants can be used to define the date alignment

1. Day alignment

- DAYADJ\_1, Align on the start of a day
- DAYADJ\_7, Align on the start of a week
- DAYADJ\_WEEK, Synonym to DAYADJ\_7

2. Monthly alignment

- MONTHADJ\_1, Align on a month start
- MONTHADJ\_6, Align on the start of halfyear

3. Yearly alignment

- YEARADJ\_1, Align on a year
- YEARADJ\_2, Align on a bi-yearly basis
- YEARADJ\_5, Align on a 5 year basis

Some examples will clarify the use of these methods.

**Example 14.7. We want the time adjustment to start on an even quarter of an hour, i.e. an even 15 minute period.**

```
$graph->xaxis->scale->SetTimeAlign(MINADJ_15);
```

**Example 14.8. We want the time to start on an even 2 hour**

```
$graph->xaxis->scale->SetTimeAlign(HOURADJ_2);
```

### 14.10.3. Manually adjusting the ticks

Since a date scale is special case of an integer scale (the underlying format of dates a timestamps which are large integers) it is perfectly possible to manually adjust the interval between each label. The interval is specified in seconds with a call to the usual

```
LinearTicks::Set($aMajorTicks,$aMinorTick)
```

as the following code snippet shows

```
<?php
$graph->SetScale('datint');

// Adjust the start time for an "even" 5 minute, i.e. 5,10,15,20,25, ...
$graph->xaxis->scale->SetTimeAlign(MINADJ_5);

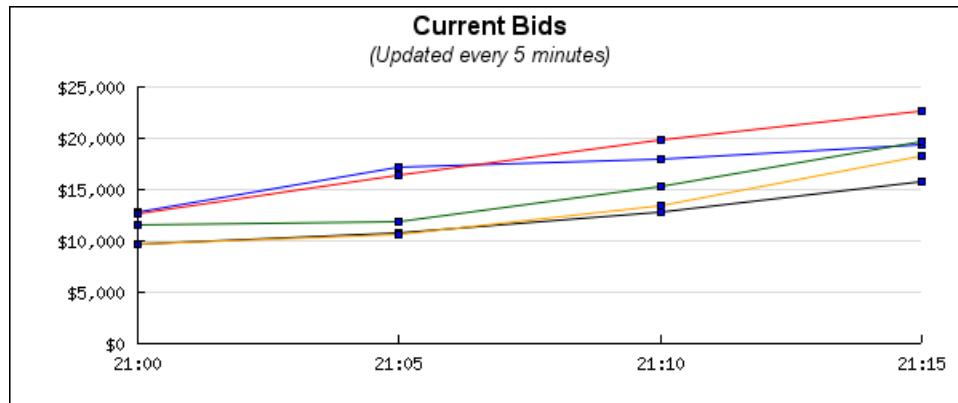
// Force labels to only be displayed every 5 minutes
$graph->xaxis->scale->ticks->Set(5*60);

// Use hour:minute format for the labels
$graph->xaxis->scale->SetDateFormat('H:i');
?>
```

In the above example the scale will start on an "even" 5 minute boundary and then have labels every 5 minutes exactly. Of course strictly speaking the adjustment with MINADJ\_5 is not necessary but will avoid that the scale starts on whatever happens to be the initial value of the data (which will control how the first label is placed).

The graph in Figure 14.66, “Manually adjusting the tick labels for a date scale (datescaleticksex01.php)” shows an example where an online auction is displaying how the bids are increasing from each participant every 5 minutes. The example also shows how to adjust the label format on the y-axis to show currency values with 1000' separator by using a label callback (using the PHP function `number_format()` ).

**Figure 14.66. Manually adjusting the tick labels for a date scale (datescaleticksex01.php) [example\_src/datescaleticksex01.html]**



### 14.10.4. Adjusting the label format

The default label format always tries to use the shortest possible unique string. To manually set a label format the method

- `DateScale::SetDateFormat($aFormatString)`

The format string uses the same format convention as the PHP function `date()`

is used. For example

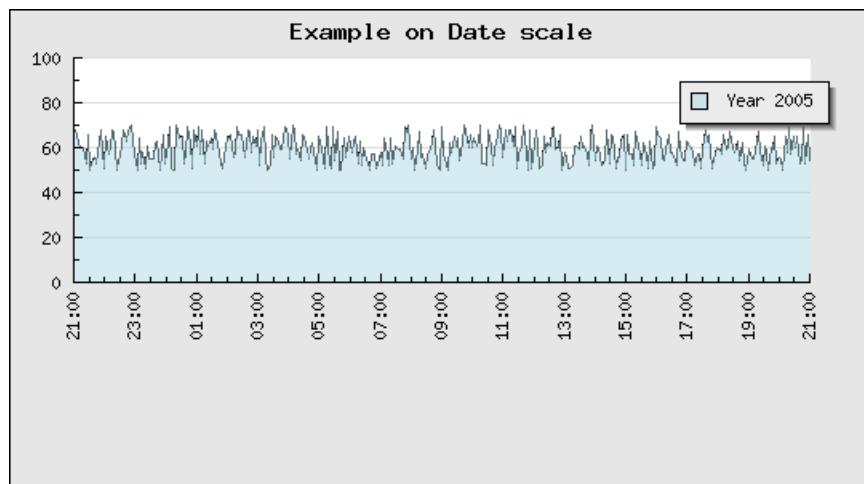
```
$graph->xaxis->scale->SetDateFormat('H:i');
```

will display the hour (24h) and minutes in the label separated by a colon (':'). Using this format string together with the modified start/end alignment

```
$graph->xaxis->scale->SetTimeAlign(MINADJ_10);
```

with our previous date example will give the result shown in Figure 14.67, “Adjusting label formatting of a date scale (`dateaxisex4.php`) [`example_src/dateaxisex4.html`]”

**Figure 14.67. Adjusting label formatting of a date scale (`dateaxisex4.php`) [`example_src/dateaxisex4.html`]**



## 14.10.5. Adjusting the automatic density of date labels

As with the linear scale it is possible to indicate what density of scale ticks is needed. This is (as usual) specified with a call to `Graph::SetTickDensity($aMajDensity, $aMinDensity)` for example as

```
$graph->SetTickDensity(TICKD_DENSE);
```

## 14.10.6. Creating a date/time scale with a manual label call-back

In the following we will assume that all data points are specified by a tuple (*time-value, date-value*) where the date/time is specified as a timestamp in seconds in the same format as is returned by the PHP function `time()`.

### Caution

Be careful if data is gathered from a different time zone and whether it is given in UTC or in the local time/date.

A label formatting callback routine will get called each time a label is to be drawn on the scale. The one parameter given to the callback function is the current time value. The returned string is then used as a label.

What we do is that we specify that the x-scale should be an ordinary integer "int" scale (remember that the data values are timestamps which are integers). We then install our custom label formatting callback (with a call to `Graph::SetLabelFormatCallback()`) which given a timestamp returns a suitable label as a string.

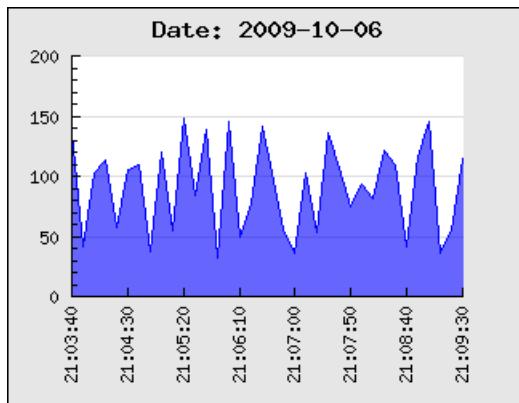
In our example we will use the PHP function `date()` to convert between the time stamp value and a suitable textual representation of the time/date value.

The callback we use is

```
// The callback that converts timestamp to minutes and seconds
function TimeCallback ($aVal) {
 return Date ('H:i:s' , $aVal);
}
```

Using some random data we can now generate the graph shown in Figure 14.68, "Manually creating a date scale (dateaxisex1.php)"

**Figure 14.68. Manually creating a date scale (dateaxisex1.php)  
[example\_src/dateaxisex1.html]**



In the above example we have specified the x-scale manually to make sure that the min/max values on the X-axis exactly matches the min/max x-data values to not leave gaps (as discussed above) between the data and the start/end of the scale.

The defined callback function will be called for each of the displayed labels. Since we are using an integer scale the labels will be set according to an suitable scale when the time stamp values are interpreted as integers.

Using integer scales this will not work very well since the library determines label positions to be at even positions (e.g. every 2,5,10, 20,50,100 etc) to suit the auto-scaling since the library will assume that the data is integers and not time stamp values.

The best way to solve this is to use an integer x-scale together with a callback function with a manually specified scale.

In order to setup the scale a bit of manually work is needed. Depending on the data to be displayed one should ensure that the scale starts and ends at suitable times and that the tick interval chosen fits with an even multiple of minutes, hours, days or what is best suited for the time range that is to be displayed.

The following code example illustrates this. It creates some "fake" data that is assumed to be sampled time based data and sets up some suitable scales and tick interval. This script may be used as a basis for more advanced handling of the time data.

```

<?php // content="text/plain; charset=utf-8"
// Example on how to treat and format timestamp as human readable labels
require_once("jpgraph/jpgraph.php");
require_once("jpgraph/jpgraph_line.php");

// Number of "fake" data points
DEFINE('NDATAPOINTS',500);

// Assume data points are sample every 10th second
DEFINE('SAMPLERATE',10);

// Callback formatting function for the X-scale to convert timestamps
// to hour and minutes.
function TimeCallback($aVal) {
 return Date('H:i', $aVal);
}

// Get start time
$start = time();
// Set the start time to be on the closest minute just before the "start" timestamp
$adjstart = floor($start / 60);

// Create a data set in range (20,100) and x-positions
// We also apply a simple low pass filter on the data to make it less
// random and a little smoother
$data = array();
$xdata = array();
$data[0] = rand(20,100);
$xdata[0] = $adjstart;
for($i=1; $i < NDATAPOINTS; ++$i) {
 $data[$i] = rand(20,100)*0.2 + $data[$i-1]*0.8;
 $xdata[$i] = $adjstart + $i * SAMPLERATE;
}

// Assume that the data points represents data that is sampled every 10s
// when determining the end value on the scale. We also add some extra
// length to end on an even label tick.
$adjend = $adjstart + (NDATAPOINTS+10)*10;

$graph = new Graph(500,250);
$graph->SetMargin(40,20,30,50);

// Now specify the X-scale explicit but let the Y-scale be auto-scaled
$graph->SetScale("intlin",0,0,$adjstart,$adjend);
$graph->title->Set("Example onTimeStamp Callback");

// Setup the callback and adjust the angle of the labels
$graph->xaxis->SetLabelFormatCallback('TimeCallback');
$graph->xaxis->SetLabelAngle(90);

// Set the labels every 5min (i.e. 300seconds) and minor ticks every minute
$graph->xaxis->scale->ticks->Set(300,60);

$line = new LinePlot($data,$xdata);
$line->SetColor('lightblue');
$graph->Add($line);

$graph->Stroke();
?>

```

## 14.10.7. Using the "DateScaleUtils" class to make manual date scale

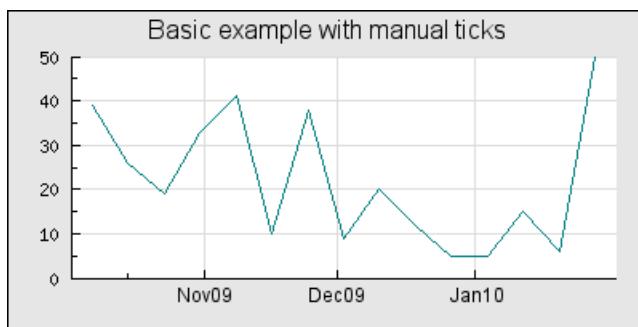
In this section we will show a very common use case where the x-axis have the unit of timestamps and where the start of each month is labeled. This is difficult (impossible) to do with the automatic tick marks since this requires the distance between consecutive tick marks to be different.

### Warning

To use manual tick marks the basic scale should be an integer scale since the underlying data are integer time stamps, for example `SetScale('intlin')`.

Figure 14.69, “Adding a label at the start of every month (`manualtickex1a.php`)” shows a basic example of what we want to achieve

**Figure 14.69. Adding a label at the start of every month (`manualtickex1a.php`)  
[[example\\_src/manualtickex1a.html](#)]**



To use the manual tick marks two steps are required

1. Determine where the tick marks should be. Both major (which can have a label and grid marks) and minor ticks can be specified. Optionally if complex labels, which cannot be calculated by a format string (either as date or `printf()` format) needs to be created.
2. Call the appropriate method to set the tick marks and optional labels.

We remind the reader that the following methods (in class Axis) are available to set the tick marks

- `Axis::SetTickPositions($aMajTickPos, $aMinTickPos=NULL, $aLabels=NULL)`
- `Axis::SetMajTickPositions($aMajTickPos, $aLabels=NULL)`

The second method above is strictly speaking not necessary, it is just a convenience function for those cases where only major ticks and labels should be set.

The following related method will also be used in this example

- `Axis::SetLabelFormatString($aFormat, $aIsDateFormat=FALSE)`

This method has been available for a long time in the library but it has recently gained the second argument. With this argument it is possible to tell if the formatting string should be interpreted as format according to the standard `printf()` format or if it should be interpreted as a format string to be used with the `date()` function.

Finally we will use a utility function that is available in "jpgraph\_utils.inc.php" in the class **DateScaleUtils**

- **DateScaleUtils::GetTicks(\$aData, \$aType==DSUTILS\_MONTH1)**

Possible values for the second argument (\$aType) are

<b>Date type</b>	<b>Description</b>
DSUTILS_MONTH	Major and minor ticks on a monthly basis
DSUTILS_MONTH1	Major and minor ticks on a monthly basis
DSUTILS_MONTH2	Major ticks on a bi-monthly basis
DSUTILS_MONTH3	Major ticks on a tri-monthly basis
DSUTILS_MONTH6	Major on a six-monthly basis
DSUTILS_WEEK1	Major ticks on a weekly basis
DSUTILS_WEEK2	Major ticks on a bi-weekly basis
DSUTILS_WEEK4	Major ticks on a quad-weekly basis
DSUTILS_DAY1	Major ticks on a daily basis
DSUTILS_DAY2	Major ticks on a bi-daily basis
DSUTILS_DAY4	Major ticks on a quad-daily basis
DSUTILS_YEAR1	Major ticks on a yearly basis
DSUTILS_YEAR2	Major ticks on a bi-yearly basis
DSUTILS_YEAR5	Major ticks on a five-yearly basis

The **DateScaleUtils::GetTicks()** is a utility function that given an array of timestamps returns an array of major and minor tick mark positions that marks the start and middle of each month when we use the **DSUTILS\_MONTH1** type specifier.

To make the graph in figure 1 we first note that it is probably a good idea to specify the min and max value of the X-axis ourself rather than letting the auto scale algorithm do that. Since the timestamps are possibly quite large values and the auto scaling algorithm will try to make the start and end values be "even" (for example multiples of 5,10,100, .. and so on).

Secondly we need to chose what scale we will use. In this case it doesn't really matter if we chose a integer (int) or a linear (lin) scale. But since timestamps by definition are integers we select an int scale for the X-axis.

Finally we need to decide what format to have on the labels. For this example we chose to show it as "Dec05" to indicate "December 2005". The format string needed to select this is "My" which we will use as argument for the **SetLabelFormatString()** method. Since the width of the label is medium wide we add some empty space on each side of the graph after we positioned the ticks to avoid the first label "hitting" the Y-axis labels. This could happen if the start of the first month on the axis is very near the X-Y axis conjunction.

We will now walk through the code to create the image in Figure 14.69, "Adding a label at the start of every month (manuallickex1a.php)" and explain each step.

First we create some random data for the X and Y axis

```
<?php
$datay = array();
$datax = array();
```

```

$ts = time();
$n=15; // Number of data points
for($i=0; $i < $n; ++$i) {
 $datax[$i] = $ts+$i*700000;
 $datay[$i] = rand(5,60);
}
?>

```

Then we get the tick positions for the start of the months

```

<?php
list($tickPositions, $minTickPositions) = DateScaleUtils::GetTicks($datax);
?>

```

We also add a bit of space "grace value" at the beginning and end of the axis

```

<?php
$grace = 400000;
$xmin = $datax[0]-$grace;
$xmax = $datax[$n-1]+$grace;
?>

```

It is now time to add the standard code to setup a basic graph. We also set the previously calculated tick positions and the label formatting string we want to use.

Note that we are careful at making sure the x-axis always start at the minimum y-value (by calling `SetPos()`), by default the x-axis is otherwise positioned at y=0 and if the y-scale happens to start at, say y=10, then the x-axis is not shown.

```

<?php
$graph = new Graph(400,200);
$graph->SetScale('intlin',0,0,$xmin,$xmax);
$graph->title->Set('Basic example with manual ticks');
$graph->title->SetFont(FF_ARIAL,FS_NORMAL,12);
$graph->xaxis->SetPos('min');
$graph->xaxis->SetTickPositions($tickPositions,$minTickPositions);
$graph->xaxis->SetLabelFormatString('My',true);
$graph->xaxis->SetFont(FF_ARIAL,FS_NORMAL,9);
$graph->xgrid->Show();
?>

```

Finally we create and add the plot to the graph and send back the graph to the browser.

```

<?php
$p1 = new LinePlot($datay,$datax);
$p1->SetColor('teal');
$graph->Add($p1);
$graph->Stroke();
?>

```

## 14.10.8. When to use manual and when to use automatic date scale?

The previous sections showed how to make use of the utility class `DateScaleUtils` to manually set the tick marks. The astute reader will also recall the possibility to use a "date" scale, i.e. specifying the scale for example as

```
<?php
// Use a date-integer scale
$graph->SetScale('datint');
?>
```

So what is the difference and when should this be used?

The answer is that the functionality to some extent overlap but with the manual scale creation there is the possibility to more exact specify the distance between the label marks. By using the "Date" scale the library will automatically adjust the labels to have a suitable distance depending on the span of the data and the size of the graph. This is "easier" but will give little control over the intervals.

With the manual scale, on the other hand, it is possible to exactly specify the distance (e.g. every three month) between each label on the expense of a few more lines of code.

There is also one more important difference an that is that with a "date" scale the tick marks will always be adjusted so that the end and beginning of the scale falls on a major tick marks. If the data to be visualized doesn't completely cover this span there might be "gaps" in the data at the beginning or/and at the end of the x-scale. With a manual scale it is possible to set the min and max x-scale value to match exactly the min/max x-values in the data and have the plot begin and end at exactly the beginning and end of the x-axis. This is not possible to guarantee with a date scale.

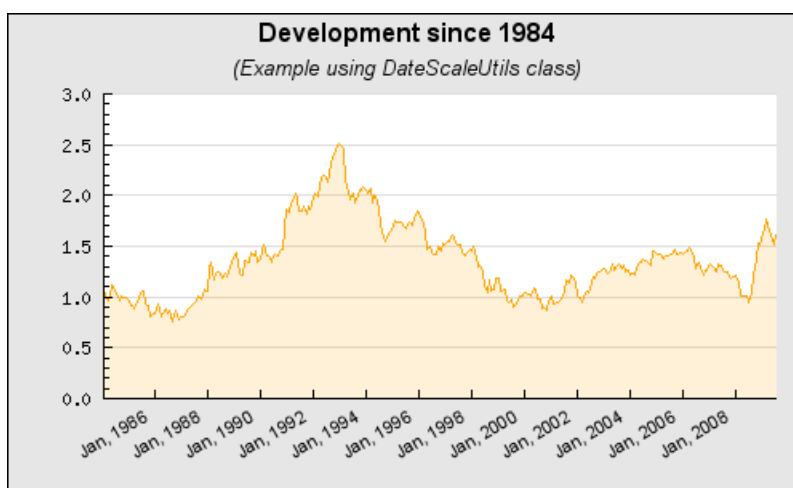
Let's put this knowledge to use and compare "side by side" the difference between these two ways of creating a date scale by creating a graph with the same data but using these two different methods.

In the first example (shown in Figure 14.70, “Manually specified date scale (`dateutilex01.php` )” we use a manually set tick scale with an explicitly set min/max value for the x-axis. The labels on the graph are formatted with a call to

```
$graph->xaxis->SetLabelFormatString('M, Y', true);
```

the second parameter ('`true`') will make the library interpret the format string as specifying a date format string.

**Figure 14.70. Manually specified date scale (`dateutilex01.php`)**  
[[example\\_src/dateutilex01.html](#)]

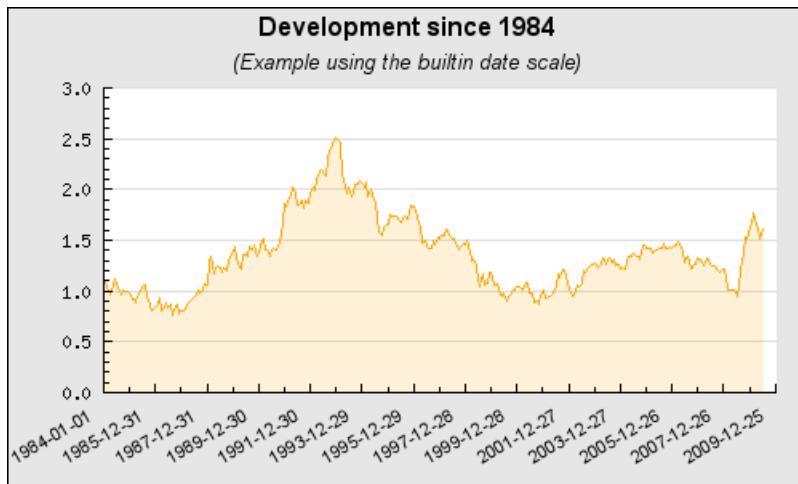


The second variant will use the exact same data but this time we will use a date scale. This is accomplished by first including the necessary support module `jpgraph_date.php` and then specifying the scale as

```
$graph->SetScale('datlin');
```

In order to make the two graphs have exactly the same label format we also use the same format string as in the previous graph, i.e. `$graph->xaxis->SetLabelFormatString('M, Y', true);`. The result of formatting is shown in Figure 14.71, “Using an automatic date scale (`dateutilex02.php`)”.

**Figure 14.71. Using an automatic date scale (`dateutilex02.php`)**  
[[example\\_src/dateutilex02.html](#)]



Comparing Figure 14.71, “Using an automatic date scale (`dateutilex02.php`)”, and Figure 14.70, “Manually specified date scale (`dateutilex01.php`)” we can see that with the automatic scaling the tick marks match the beginning and ending of the x-scale but at the expense of a small gap at the end of the data since the data doesn't extend quite as far as the scale.

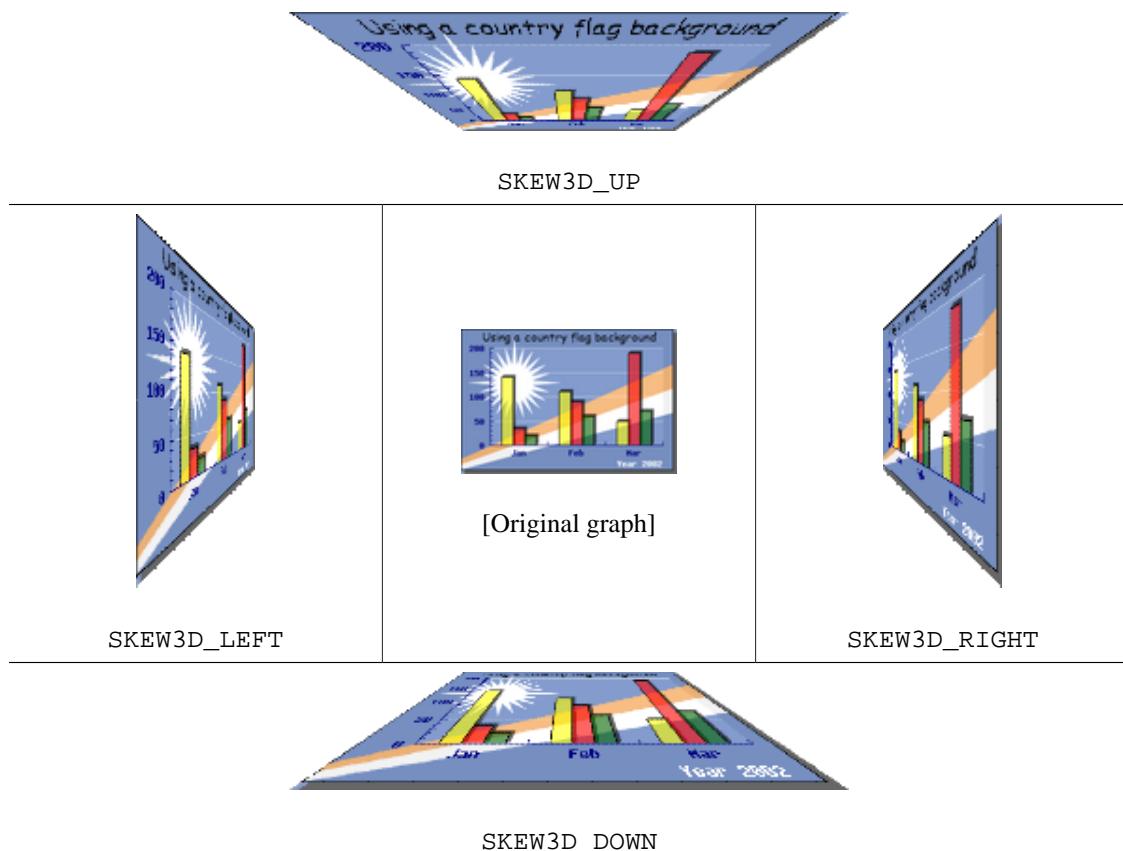
## 14.11. Adding shearing image transformation to the graph

As a final “touch” on the image it is possible to apply a shearing transformation to the generated image. This can be used to give the image a “3D” perspective. The transformation is done by the method `Graph::Set3DPerspective()`. It should be noted that since these transformations are all done in PHP they are (as all image processing) quite processor intensive.

In order to get access to the transformation functionality the module “`jpgraph_imgtrans.php`” must first be included in the script.

In Figure 14.72, “Different types of shearing transformation” the original image is shown in the middle and the four types of shearing is shown around in positions indicating the type of shearing. The symbolic name for the type of shearing is shown below each image.

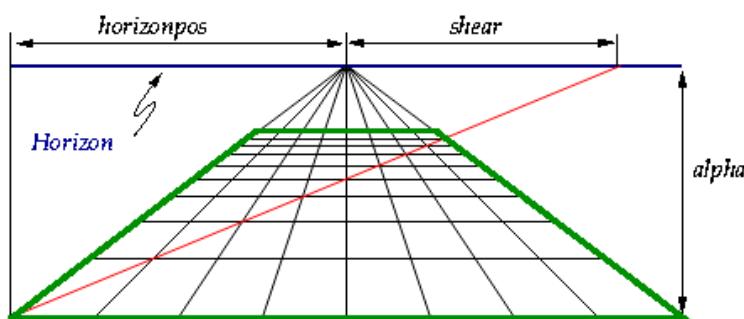
**Figure 14.72. Different types of shearing transformation**



The transformation is specified with the method

- `Graph::Set3DPerspective($aDir, $aAlpha=100,$aShear=120,$aQuality=false,$aFillColor='#FFFFFF', $aBorder=false,$aMinSize=true,$aHorizonPos=0.5)`

**Figure 14.73. Explaining the shearing parameters**



The different parameters that effect the transformation can now be explained with the help of Figure 14.73, “Explaining the shearing parameters”

**\$aDir** This is the symbolic constant to define which of the four basic types of transformation as shown in Figure 14.72, “Different types of shearing transformation”

<b>\$aAlpha</b>	This defines the distance from the bottom of the image to the artificial horizon
<b>\$ashear</b>	This defines from the perspective vanish point on the artificial horizon to the crossing of the "shearing line" on the artificial horizon..
<b>\$aHorizPos</b>	This specifies the distance from the left edge of the image to the perspective vanish point
<b>\$aQuality</b>	With this parameter set to <b>true</b> the algorithm will do additional image interpolation to increase the quality of the resulting transformation on the expense of further processing time.
<b>\$aFillColor</b>	Specifies the background fill color to be used. A value of false (the default) indicates no fill
<b>\$aBorder</b>	Add a border around the transformed image
<b>\$aMinSize</b>	The transformed image is usually smaller than the original image and if this parameter is set to true then the resulting image will be as small as it can be. If it is false then the original image size will be kept.

This might be regarded as "gimmick" factor but has proven useful in batch (off-line) processing to produce a sequence of images that gives the appearance of a graph that rotates into place.

## 14.12. Rotating graphs

The library supports an arbitrary rotation of a the plot area as well as some special convenience method to rotate the plot area 90 degree which most often is used to draw a horizontal bar graph instead of a (standard) vertical bar graph.

### Caution

Adding a rotation transformation will make the graph generation slightly slower since each point of the graph as to go through a transformation step before being stroked on to the image. The library tries to mitigate this as much as possible by using a pre-calculated transformation matrix and also makes further optimizations for the special case of 90 degree rotations.

### Caution

Any background images (see Section 14.15, “Adding images and country flags to the background of the graph”) will not be rotated with the graph. This limitation exists since the performance and memory usage of doing real image transformation in PHP would be too poor. Any background images needing rotation must be rotated outside in some image manipulation program (e.g. Gimp, IrfanView).

When a plot area is rotated there are two things to be aware of

1. *individual labels on the axis are not rotated*

The design decision behind this is that bit mapped fonts cannot be arbitrarily rotated and rotating TTF fonts will decrease (in general) the readability. If angle rotation is needed on the labels it is still possible to use the method `Axis::SetLabelAngle()`

### Note

Since the anchor point for labels is by default the optimum for graph at 0 degree the anchor point and alignment for the labels on the axis should probably be adjusted to

get a better visual appearance on the rotated graph. This is accomplished by the method  
`Axis::SetLabelAlign()`

2. *any background image or background gradient is not rotated*

The design decision behind this is purely computational Doing a full image rotation would be excruciating CPU intensive using PHP.

## 14.12.1. Free rotation of the plot area

The rotation of the plot area is controlled with the following two methods

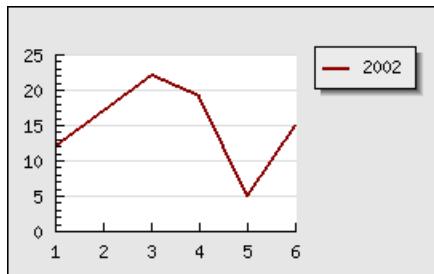
- `Image::SetAngle($aAngle)`  
"\$Angle" = Angle of rotation specified in degrees. A positive angle specifies a clockwise rotation.
- `Image::SetCenter($aX, $aY)`  
Specifies the center of rotation

The Image class is the lowest graphic layer in the library and it is access through the instance variable "\$img" of the Graph class. So for example to rotate a plot are 45 degree the following line has to be added

```
$graph->img->SetAngle(45);
```

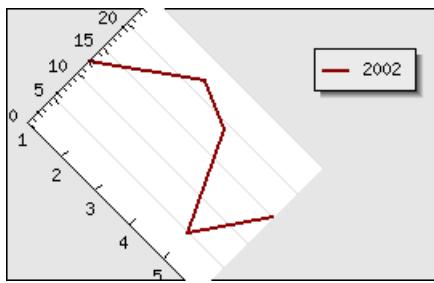
By default the center of rotation will be the center point of the plot area. The following examples will clarify this. We will use the (very) basic graph shown in Figure 14.74, “Original unrotated graph (rotex0.php)” to demonstrate rotation.

**Figure 14.74. Original unrotated graph (rotex0.php) [example\_src/  
rotex0.html]**

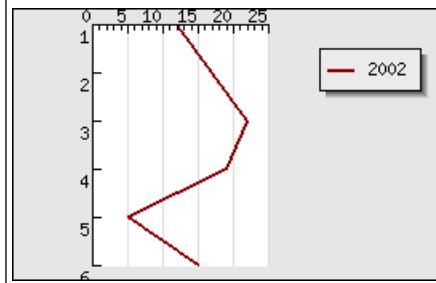


In Figure 14.75, “Rotating the plot area 45 degrees (rotex1.php)” and Figure 14.76, “Rotating the plot area 90 degrees (rotex2.php)” we have rotated the plot area around (the default) the center of the plot area

**Figure 14.75. Rotating the plot area 45 degrees (rotex1.php)**  
[example\_src/rotex1.html]

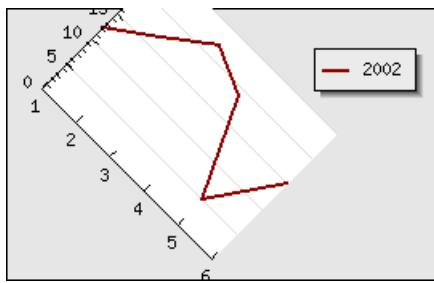


**Figure 14.76. Rotating the plot area 90 degrees (rotex2.php)**  
[example\_src/rotex2.html]

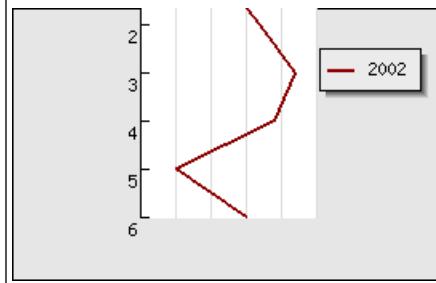


In the above two examples the center of the rotation was the center point of the plot area. If we instead change the center of rotation to be the center of the entire graph we get the result shown in Figure 14.77, “Rotating the plot area 45 degrees (rotex3.php)” and Figure 14.78, “Rotating the plot area 90 degrees (rotex4.php)”.

**Figure 14.77. Rotating the plot area 45 degrees (rotex3.php)**  
[example\_src/rotex3.html]

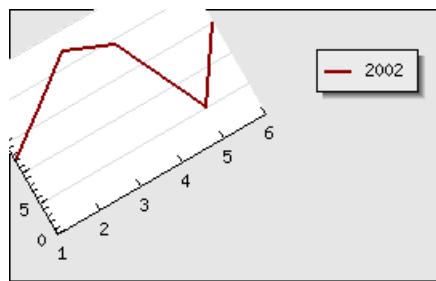


**Figure 14.78. Rotating the plot area 90 degrees (rotex4.php)**  
[example\_src/rotex4.html]



As a final example in Figure 14.79, “Rotating the plot area -30 degree around the bottom left corner (rotex5.php)” we show the result of rotating the plot area -30 degree around the bottom left point in the graph

**Figure 14.79. Rotating the plot area -30 degree around the bottom left corner (rotex5.php)** [example\_src/rotex5.html]



## 14.12.2. Rotating the plot area 90 degree

As can be seen above in Figure 14.76, “Rotating the plot area 90 degrees (rotex2.php)” and Figure 14.78, “Rotating the plot area 90 degrees (rotex4.php)” the rotation does not alter the overall size and margin

of the graph even though we probably should do so in order to better accommodate the rotated plot areas topography. It is of course perfectly possible to adjust the size of the graph manually.

The slight complication with general rotation is that the margins also rotates, this means that if the graph is rotated 90 degrees the left margin in the image was originally the bottom margin. In addition by default the center of the rotation is the center of the plot area and not the entire image (if all the margins are symmetrical then they will of course coincide). This means the center of the rotation will move with the margin (since the specify the exact location of the plot area)..

So for the case of rotating a graph 90 degree the library provides a convenience method to do both the rotation and specifying the margin at the same time (to avoid the mental exercise described above) by providing the method

- `Set90AndMargin( $aLeft=0,$aRight=0,$aTop=0,$aBottom=0 )`

Rotates the plot area 90 degrees and sets the graph margin areas

This method is probably most commonly used with bar graphs to create horizontal instead of vertical bars. See the section on bar graphs, ??, for more example on this.

## 14.13. Using anti-aliasing in the graph generation

The library have partial support for the use of anti-aliasing when crating graphs. The level of support is different depending on the actual graph type and there are some caveats to be aware of.

### Note

One question that is often asked is why does the library not support full anti-aliasing in every type of graphs? The answer is that since neither GD graphic library does not support real anti-aliasing this is done in the library itself most of the time. Doing full image anti-aliasing is very CPU intensive and can for large graph easily take four times longer than no anti-aliasing. In addition the way full image anti-aliasing is done is by creating larger canvas which the graph is drawn on and then do a smoothing operation, basically scale the image to the correct size puts some limitation on the details resolution. For example text needs special handling since the TTF fonts are already anti-aliased and cannot be re-anti-aliased a second time.

### 14.13.1. Anti-aliasing for line drawing graphs

This applies to all graphs that makes use if line drawing. For example line plots and radar charts. This is the only "true" anti-aliasing done in the library.

This anti aliasing is enabled by calling the method

- `Image::SetAntiAliasing($aFlg=true)`

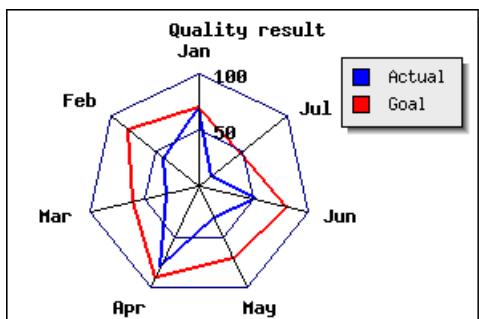
As a concrete example the following line can be used to enable anti-aliasing for both line and wind rose plots

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

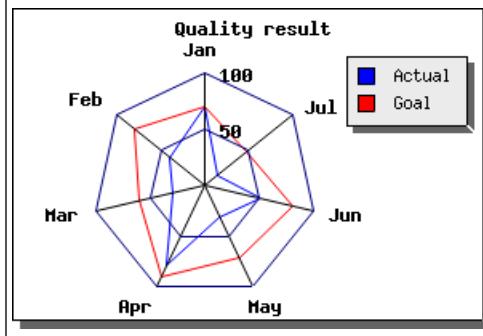
The algorithm used for line anti-aliasing makes a gradient color transition between foreground line color and background while plotting a line using a standard mid-point line algorithm. To achieve best visual result always use a dark line color on a light colored background.

The images below shows the difference between a plain graph on the left and the same graph drawn with anti-aliasing enabled on the right.

**Figure 14.80. Plain radar plot (radarex8.php) [example\_src/radarex8.html]**



**Figure 14.81. Anti-aliased radar plot (radarex8.1.php) [example\_src/radarex8.1.html]**



There are a number of limitations with the current implementation that should be noted. The reason these limitation exists is that for reasons of CPU load we have chosen not to implement full anti-aliasing for arbitrary shapes.

1. Speed! Doing anti-aliased lines in PHP is not cheap. Anti-aliased line drawing is roughly 6-8 times slower then lines without anti-aliasing So using anti-aliasing isn't suitable together with high-load often updated graphs. Remember to use the caching feature (see Section 5.6, "Efficient graph generation using the built-in cache subsystem".)
2. Lines will ignore any width and only have a single line-width of approximately=1. It is not possible to set the line width when anti-alias is used. (This would require a full implementation of anti-aliased polygons which is not implemented.)
3. The colors will be a little bit "weaker" since they will now not consist of one dominant color but rather being made up by at least two different colors.
4. Anti-aliasing does not work very well together with background images since it assumes a the same solid color on each side of the line. Doing a more advanced anti-aliasing algorithm would simple take to much processing power.
5. Even when anti-aliasing is enabled fast line-drawing will be used for horizontal, vertical and diagonal lines since these get sampled at high enough frequency anyway and doesn't benefit from anti-aliasing. This optimization has a slight visual impact, as can be seen in Figure 14.81, "Anti-aliased radar plot (radarex8.1.php)" The vertical line appear to the human eye to have a slightly different color and looks a bit thinner.

### Note

The difference between a standard line and one line with anti-aliasing is shown in large magnification in ???. The approach used in the library roughly corresponds to an even weighted 3x3 Bartlett-filter. It would be possible to apply some more advanced filtering techniques to achieve an even better result but doing high-intensive 2D signal processing on a HTTP-server is not a brilliant idea. The algorithm used in the library is a reasonable trade-off between visual appearance and efficiency.

**Figure 14.82. Anti-aliasing up-close.** The figure shows the difference between a standard line (on-top) and the corresponding anti-aliased line (on-the bottom)



## 14.13.2. Anti-aliasing in pie graphs

For Pie plots a full image anti-aliasing algorithm is used. The original image is scaled to twice its specified width and then down sampled to the required size. This will give a low pass filtering effect which takes the edges away from the image and gives a smooth appearance. However the filtering is quite processor intensive.

Anti-aliasing for pie plots are enabled by calling

- `PieGraph::SetAntiAliasing($aFlg=true)`

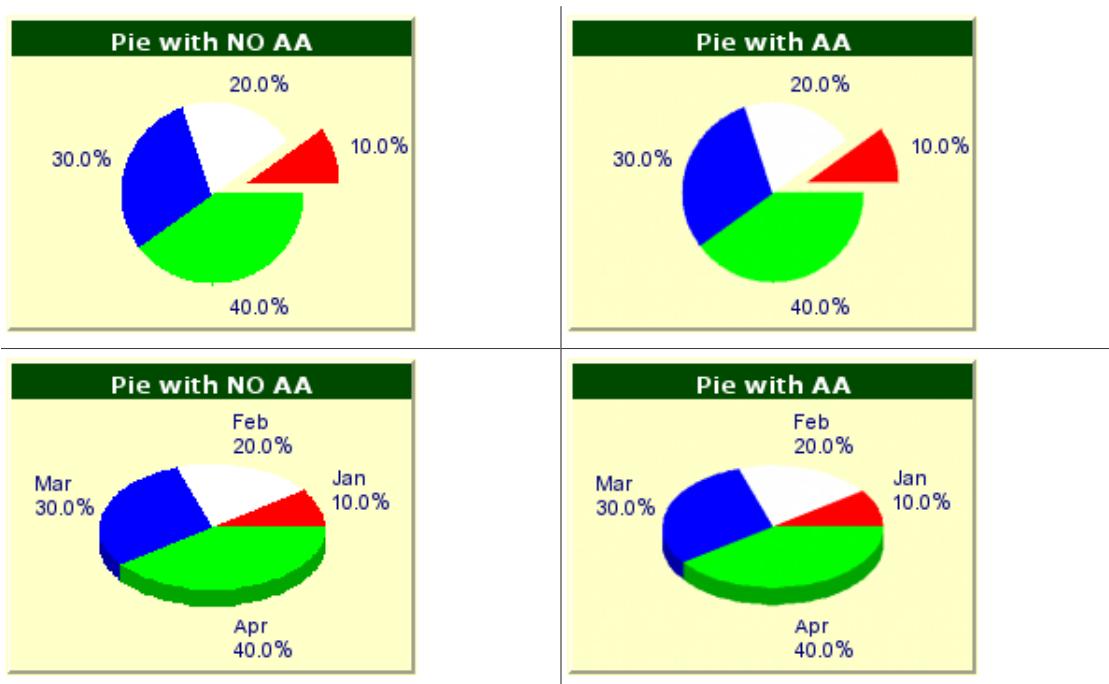
as the following code snippet shows

```
$piegraph->SetAntiAliasing();
```

There is inherent limitation using this method and that is that edges on the Pie can not be used.

The figure below shows the visual impact the anti-aliasing have on both 2D and 3D Pie plots.

**Figure 14.83. Affects of using anti-alias for Pie-graphs**



### 14.13.3. Anti-aliasing in Windrose plots

This uses the same algorithm as the Pie plots with similar restrictions and limitations.

### 14.13.4. Anti-aliasing for Contour plots

This uses almost the same algorithm as Pie plots. The additional feature here is that it is possible to specify the size of the over-sampling. This can be value between 2-5. However, using an oversampling larger than 3 has very little additional visual improvement but will significantly increase the processing time.

## 14.14. Adding icons (and small images) to the graph

In addition to the standard background image you can also add an arbitrary number of icons onto the background of the graph. These icons are created with a call to the special Plot class `IconPlot` defined in the module "`jppgraph_iconplot.php`". The image which is the base of the icons can be

1. an arbitrary image from a file
2. one of the built-in country flags

The basic structure of adding an icon somewhere on the graph is to first create an instance of the `IconPlot` class and then position the icon at the wanted x,y-position on the graph and finally add the object to the graph using the standard `Graph::Add()` method. The constructor for the `IconPlot` class have the following signature

- `__construct($aFile='', $aX=0, $aY=0, $aScale=1.0, $aMix=100)`

The parameters should be self explanatory. The `$aMix` factor specifies the degree of alpha blending between the background and the icon. A value of 100 means no blending and a value of 0 means that the icon is not shown at all, only the background.

Some useful methods in this class are

- `SetCountryFlag($aFlag, $aX=0, $aY=0, $aScale=1.0, $aMix=100, $aStdSize=3)`

This method specifies that the specified country flag should be used

- `SetPos($aX, $aY)`

Same as the optional arguments in th constructor. The coordinates are specified as absolute pixels where (0,0) is the top left corner.

- `CreateFromString($aStr)`

Creates the image from a string instead of reading it from a file.

- `SetScalePos($aX, $aY)`

Specifies the position using the scale coordinate as specified by the x- and y-scales

- `SetAnchor($aXAnchor='left', $aYAnchor='center')`

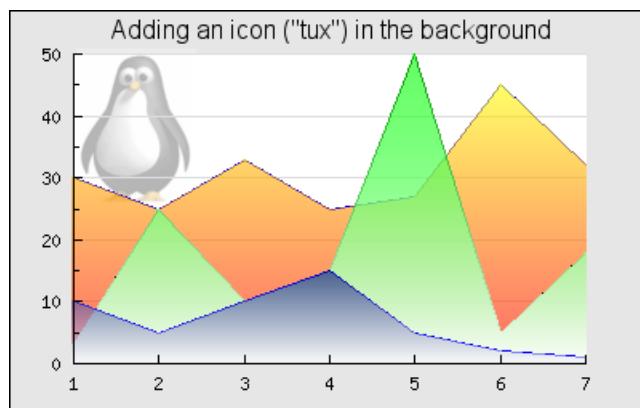
Sets the anchor point of the icon, i.e. the position in the icon that should be aligned with the specified icon position in the graph.

The following short example shows how an icon is created and added to a graph

```
$icon = new IconPlot('myimage.png',0.2,0.3,1,30);
$icon->SetAnchor('center','center');
$graph->Add($icon);
```

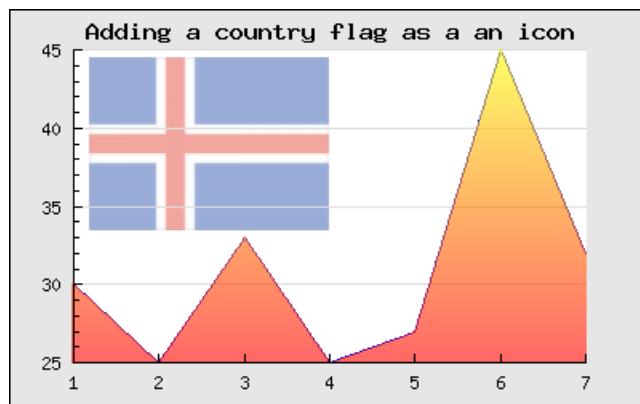
It is possible to control how much of the icon should be blended into the background by specifying a percentage (1-100). The example below in Figure 14.84, “Mixing an icon image into the background of the graph. The area plot in the graph uses alpha blending to achieve see-through affect (lineiconex1.php)” shows how to mix in the picture of “Tux the penguin” into the background of a filled line graph.

**Figure 14.84. Mixing an icon image into the background of the graph. The area plot in the graph uses alpha blending to achieve see-through affect (lineiconex1.php) [example\_src/lineiconex1.html]**



To specify any of the roughly 200 country flags as an icon the first step is to create an empty Icon and then call the `IconPlot::SetCountryFlag()` method with the appropriate parameters. This is illustrated below by adding the Icelandic flag into the background as an icon.

**Figure 14.85. Adding a country flag icon in the background (lineiconex2.php) [example\_src/lineiconex2.html]**



## Caution

Some older versions of PHP (< 4.3.3 using the built-in GD) have problems rendering blended images. If you have this problem then you need to upgrade to a more recent version of PHP.

## 14.15. Adding images and country flags to the background of the graph

The background can not only be a solid (or semi-transparent) color. It is also possible to use images as backgrounds. These can be arbitrary user specified images in any format (as determined by their file suffix). Since the library has a built in support for all established countries (as of 20 Dec 2008) in regards to their flags (all flags are encoded within the library) it is also possible to use a country flag as a background. This could be considered a special case of the user defined image background.

1. Graph::SetBackgroundImage(\$aFileName, \$aBgType=BGIMG\_FILLPLOT, \$aImgFormat='auto')
2. Graph::SetBackgroundImageMix(\$aMix)
3. Graph::SetBackgroundImagePos(\$aXpos, \$aYpos)
4. Graph::SetBackgroundCountryFlag(\$aName, \$aBgType=BGIMG\_FILLPLOT, \$aMix=100)

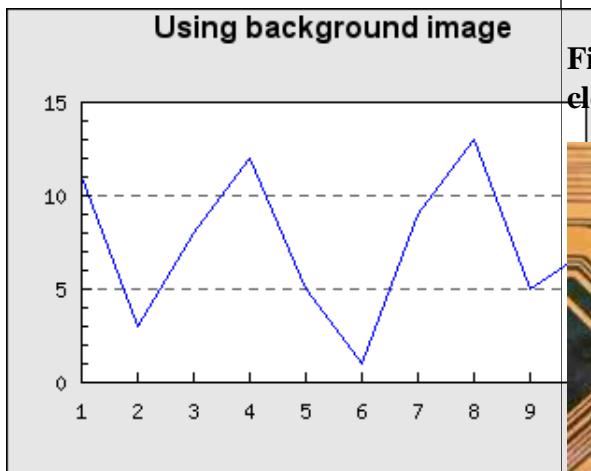
For both background images and flags it is possible to define how the image should be used on the page with the background type parameters. This parameter specifies how the image should be placed on the graph. The following options are available:

- BGIMG\_FILLPLOT . This means that the background image will be scaled to that it exactly fits the plot area
- BGIMG\_FILLFRAME . This means the the background image will be scaled so that it exactly fits the entire graph
- BGIMG\_COPY . This means that the background image will be copied in its original size without any size adjustments
- BGIMG\_CENTER . Almost the same as BGIMG\_COPY but this also means that the image will be automatically centered in the graph
- BGIMG\_FREE, This is a variant of the BGIMG\_COPY but in this case the image will be free on its own and have no color blending with the margin color

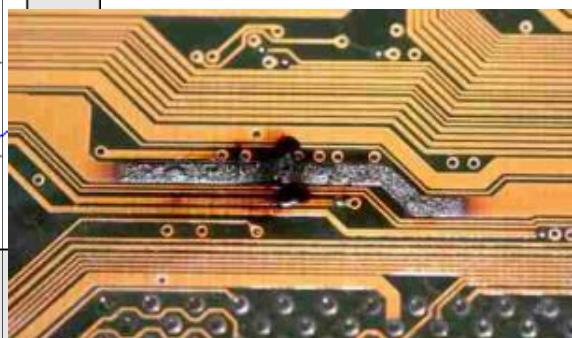
To help explain these types of backgrounds they are illustrated below by adding the image in Figure 14.87, “Background image (a closeup of our burnt server)” as a background to the plain graph in Figure 14.86, “The graph that will be used to add backgrounds to”. In order not to have the background image take too much emphasis in the graph we specify the mixing to 25% with a call to

```
$graph->SetBackgroundImageMix(25);
```

**Figure 14.86. The graph that will be used to add backgrounds to**

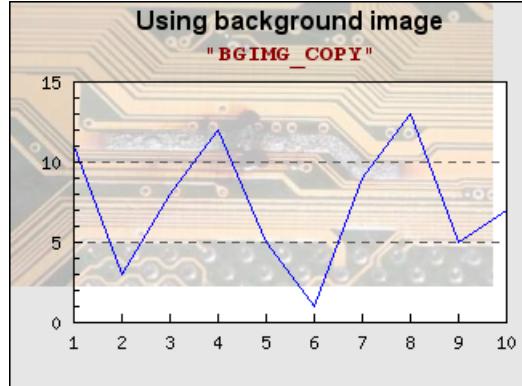


**Figure 14.87. Background image (a closeup of our burnt server)**



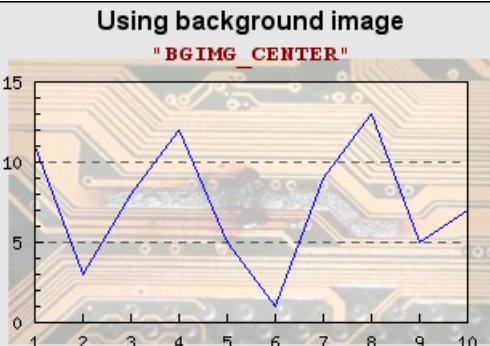
In the table below the various options for adjusting the size and position of the background image are shown

**Figure 14.88. BGIMG\_COPY**  
(background\_type\_ex0.php)  
[example\_src/  
background\_type\_ex0.html]



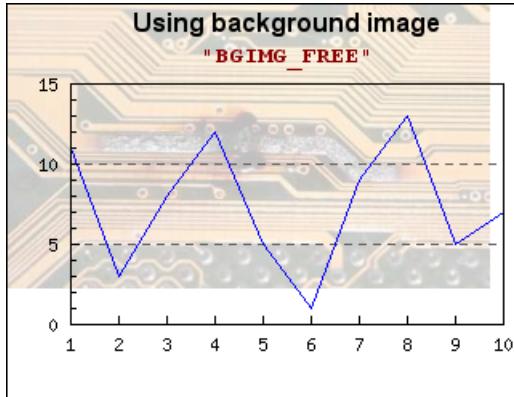
BGIMG\_COPY

**Figure 14.89. BGIMG\_CENTER**  
(background\_type\_ex1.php)  
[example\_src/  
background\_type\_ex1.html]

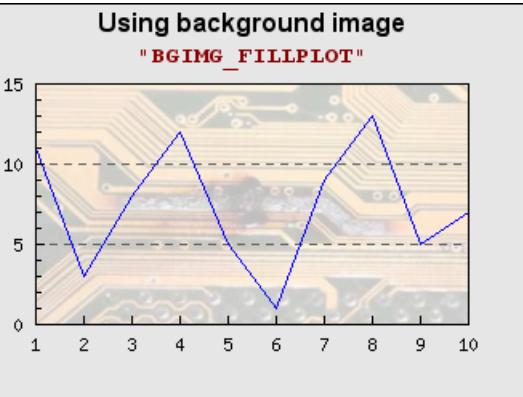


BGIMG\_CENTER

**Figure 14.90. BGIMG\_FREE**  
(background\_type\_ex2.php)  
[example\_src/  
background\_type\_ex2.html]



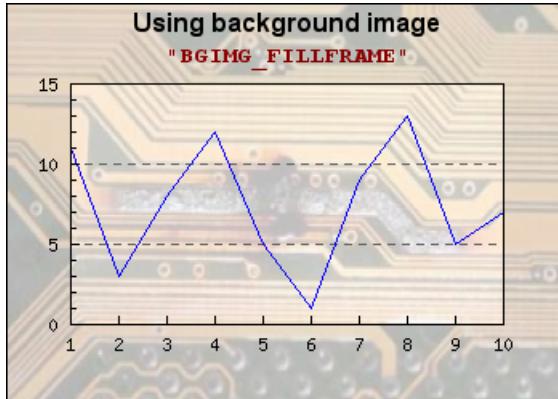
**Figure 14.91. BGIMG\_FILLPLOT**  
(background\_type\_ex3.php)  
[example\_src/  
background\_type\_ex3.html]



BGIMG\_FREE

BGIMG\_FILLPLOT

**Figure 14.92. BGIMG\_FILLFRAME**  
(background\_type\_ex4.php)  
[example\_src/  
background\_type\_ex4.html]



BGIMG\_FILLFRAME

## 14.15.1. Using country flags as backgrounds

The method `Graph::SetBackgroundCountryFlag($aName, $aBgType, $aMix)` makes it possible to use an of the built-in country flags as a background i the same way as any other user specified background image. All included country flags as of Dec 2008 are listed in Appendix F, *List of all country flags*. The background country flag is specified by its short name as listed in Appendix F, *List of all country flags*.

## Caution

The dynamics of world politics and geographic boundaries will no doubt make the list of included countries obsolete almost at the same time the library is released. The intention is that all currently known country and similar geographic entities should be included. Any missing flags shall and can not be interpreted as any political stand it is merely a consequence of political changes since the library was released or a possible oversight or simply a mistake either human or algorithmically in how the country flags are produced.

## 14.16. Using background gradients

A color gradient background will change the color from a start color to a finish color in even steps (the number of steps depends on the size of the graph). The direction of the gradient is controlled by specifying a style parameter for the gradient. The signature for the gradient background method is

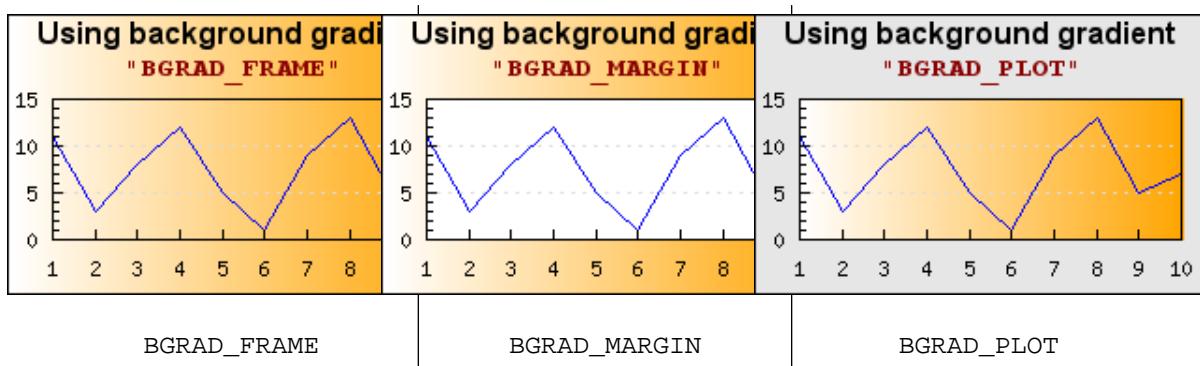
1. Graph::SetBackgroundGradient(\$aFrom='navy', \$aTo='silver',  
\$aGradType=2, \$aStyle=BGRAD\_FRAME)

The last style parameter specifies where in the graph the gradient style should be applied and the different options are

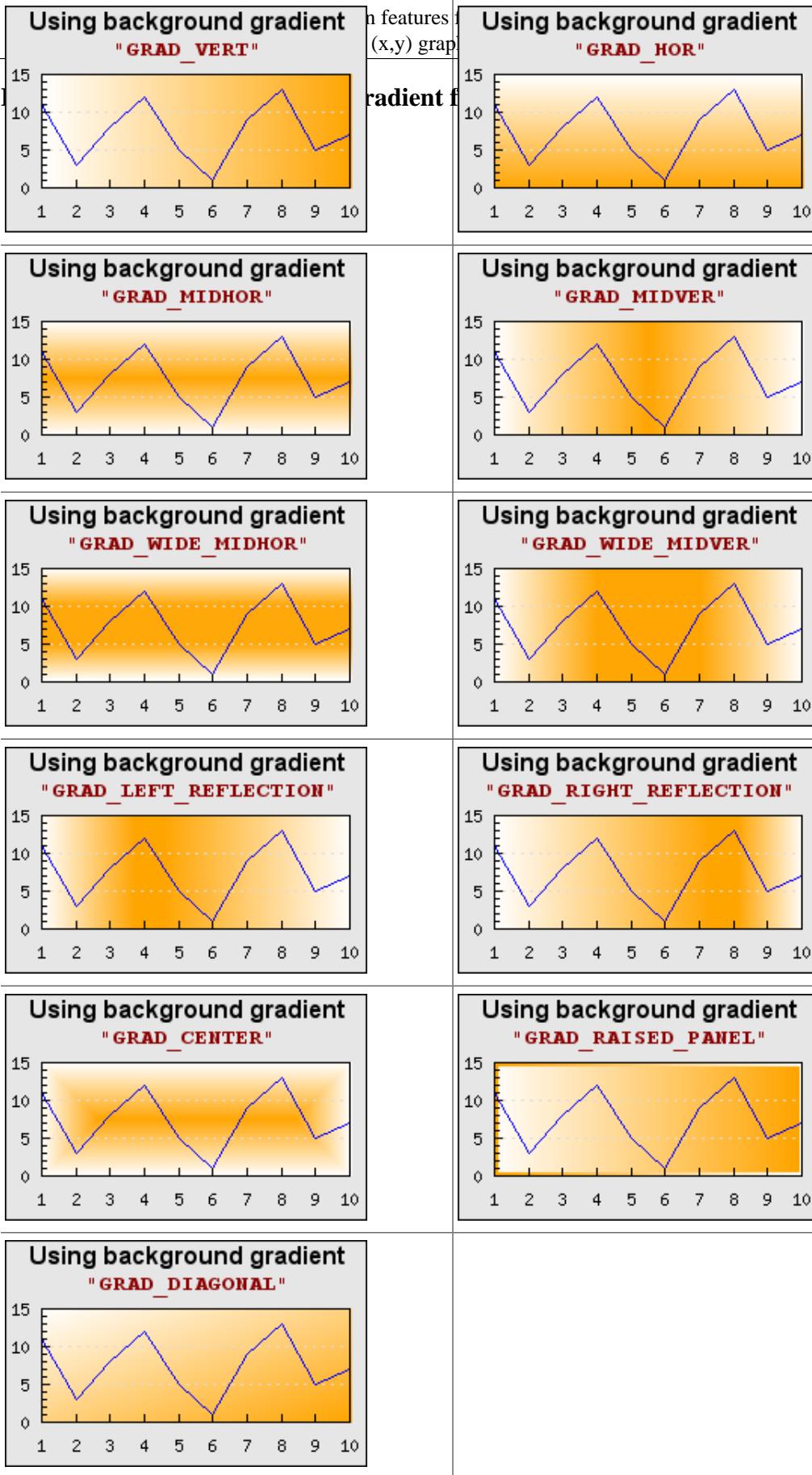
- BGRAD\_FRAME, The entire graph frame will be filled
- BGRAD\_MARGIN, Only the margin area (the graph area outside the plot area) will be filled
- GRAD\_PLOT, Only the plot area will be filled

This is illustrated in Figure 14.93, “What area of the graph the gradient should affect”

**Figure 14.93. What area of the graph the gradient should affect**



The different types of gradient fills are shown in Figure 14.94, “Different types of gradient fills” where the gradient from color is “white” and the to color is “orange”



## 14.16.1. Generating gradient background off-line

A drawback of using gradient backgrounds is that they are processing intensive and since they are generated each time the graph is generated it can take substantive amount of time for large graphs. To help with this the library offers a small (very simple) utility that can be run to create gradient images off line. These images can later on be used as (static) background images. In the distribution there is a small utility in the "Examples/" directory called "mkgrad.php". Pointing the browser to this script will show a basic form as shown in Figure 14.95, "The "mkgrad" utility to create gradient images". The script will allow the creation of a gradient images using any of the predefined colors and gradient types.

**Figure 14.95. The "mkgrad" utility to create gradient images**

### Generate gradient background

The dialog box has a blue border and contains the following fields:

- Width:
- Height:
- From Color:
- To Color:
- Gradient style:
- Filename: (empty to stream)
- Ok button

## 14.17. Adding arbitrary texts to the graph

In much the same was as icon images can be added to the plot so can arbitrary text strings be added. This is done by creating an or several instances of the Text class (one per string that is needed). This feature is typically used to add clarifications or other information related to the actual graph.

These text instances are then added to the graph via the normal Graph::Add( ) method.

The position of the text in the image/graph can be given as either absolute pixels, fractions of the width/height of the graph or as scale values according to the specified (or automatically determined) scale.

To show some ways of positioning the text we use a very simple bar graph not to distract from the text. We first just add a single text line with most of the settings their default value. We do this by adding the following lines

```
<?php
// Create an instance of the Text class and set the string at the same time
$txt = new Text('This is a text');

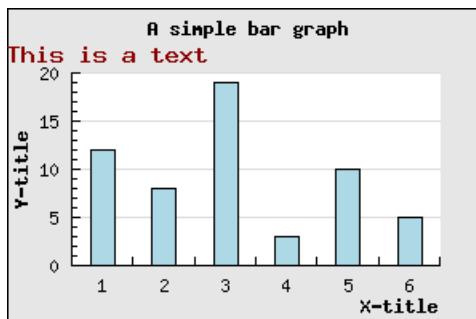
// Position the string at absolute pixels (0,20).
// ((0,0) is the upper left corner)
$txt->SetPos(0, 20);
```

```
// Set color and font for the text
$txt->SetColor('red');
$txt->SetFont(FF_FONT2,FS_BOLD);

// ... and add the text to the graph
$graph->AddText($txt);
?>
```

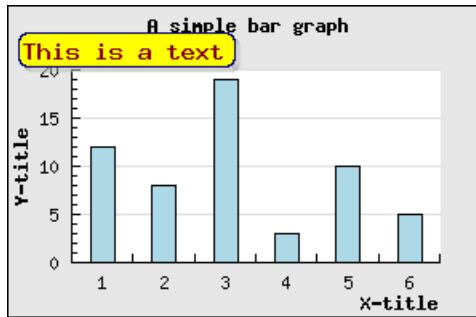
The resulting graph can be seen in Figure 14.96, “Adding a text object to a graph (example25.php) ”

**Figure 14.96. Adding a text object to a graph (example25.php) [example\_src/example25.html]**



Let's make the text stand out a bit more by having a background color, framing the text box and adding a drop shadow by using the method `Text::SetBox()`. The resulting graph can be seen in Figure 14.97, “Making the text stand out a bit more by adding a background color and frame (example25.1.php) ”

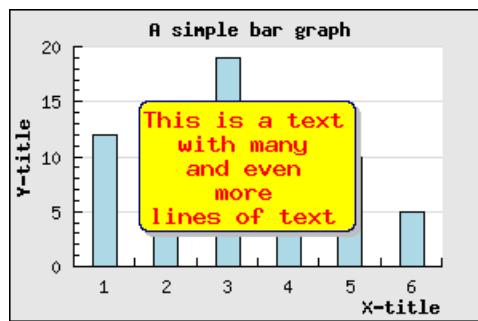
**Figure 14.97. Making the text stand out a bit more by adding a background color and frame (example25.1.php) [example\_src/example25.1.html]**



In order to use a text with several lines each line needs to be separated by a newline ("\\n" character). The default paragraph alignment is left edge but as was discussed in ?? paragraph alignment can be adjusted.

In our final example we place a text with several lines in the middle of the plot area.

**Figure 14.98.** Adding a text object with multiple rows of text. Paragraph alignment is set to "center" (`example25.2.php`) [`example_src/example25.2.html`]



### Tip

To set the position of the text using scale positions instead use the method

- `Text::SetScalePos($aX,$aY)`

---

## **Part IV. Creating linear and non-linear graphs**

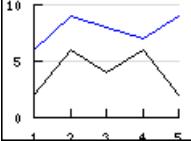
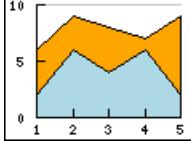
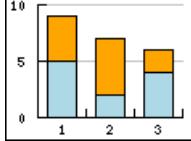
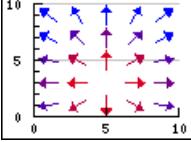
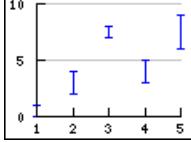
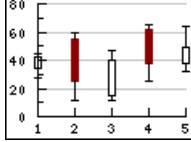
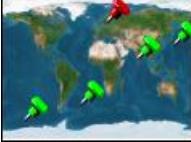
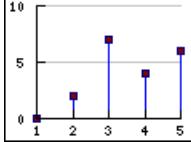
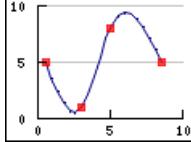
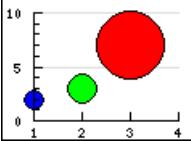
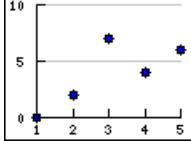
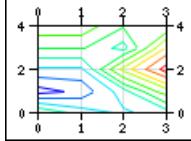
---

## Table of Contents

15.	Different types of linear (cartesian) graph types .....	196
15.1.	Basic Line and area graphs .....	197
15.2.	Bar graphs .....	228
15.3.	Error plot graphs .....	242
15.4.	Stock graphs .....	243
15.5.	Scatter graphs .....	245
15.6.	Contour graphs .....	254
15.7.	Combining several different plot types in the same graph .....	262
15.8.	Creating several graphs in the same image .....	264
16.	Non-Linear graph types .....	271
16.1.	Pie graphs .....	271
16.2.	Radar graphs .....	289
16.3.	Polar graphs .....	298
16.4.	Gantt charts .....	305
17.	Additional graph types .....	348
17.1.	LED bill boards .....	348
17.2.	Captcha generation .....	351
17.3.	Canvas graphs .....	353
18.	Miscellaneous formatting and tools .....	362
18.1.	Linear regression analysis .....	362

# Chapter 15. Different types of linear (cartesian) graph types

**Figure 15.1. Supported linear graph types in the library**

 a) Line plot (See Section 15.1.1, “Creating a line graph”)	 b) Area plot (See Section 15.1.10, “Creating a filled line graphs (a.k.a. area plots)”)	 c) Bar plot (See Section 15.2, “Bar graphs”)
 a) Field plot (See Section 15.5.3, “Field plots”)	 b) Error plot (See Section 15.3, “Error plot graphs”)	 c) Stock plot (See Section 15.4, “Stock graphs”)
 a) Geo-map plot (See Section 15.5.5, “Creating Geo-maps”)	 b) Impuls (stem) plot (See Section 15.5, “Scatter graphs”)	 c) Spline plot (See Section 15.1.15, “Constructing smooth line plots with Cubic Splines”)
 a) Balloon plot (See Section 15.5.4, “Balloon plots”)	 b) Scatter plot (See Section 15.5, “Scatter graphs”)	 c) Contour plot (See Section 15.6, “Contour graphs”)

## 15.1. Basic Line and area graphs

Line graphs (we will use the term line graph to refer to an entire graph and the term line plot to refer to a single data series in a line graph) is together with bar graphs the simplest and perhaps the most commonly used graph type. In Section 4.2, “Graphing the number of sun spots during the 19th Century” we have already shown an example of a line graph without explaining too much of the details. The remainder of the section will go into some more details on the options available when creating a line graph.

### 15.1.1. Creating a line graph

A line graph always make use of one or several instances of the class `LinePlot` which represent one plotted data series in the graph. In all our example we follow the naming convention to always name the instance of the `LinePlot` class as "`$lineplot`"

The absolutely simplest line graph that is possible to create is shown in Figure 15.2, “The most simple line graph (`example0-0.php`)”

#### Example 15.1. The most simple line graph (`example0-0.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_line.php');

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

// Size of the overall graph
$width=350;
$height=250;

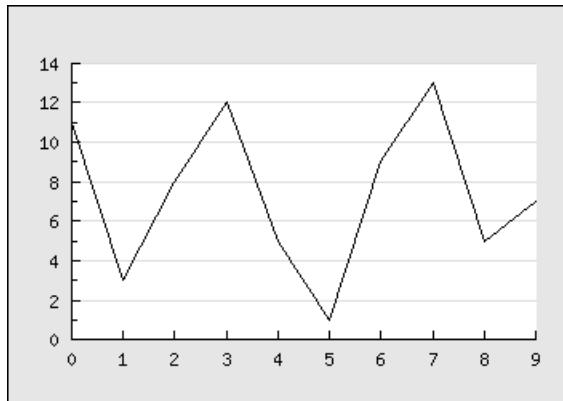
// Create the graph and set a scale.
// These two calls are always required
$graph = new Graph($width,$height);
$graph->SetScale('intlin');

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

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

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

**Figure 15.2. The most simple line graph (`example0-0.php`) [`example_src/example0-0.html`]**



There are a number of things to point out here

- Both the X and Y axis have been automatically scaled and since we didn't provide any x-scale the data point have been numbered automatically starting at 0
- By default the Y-grid is enabled and displayed in a "soft" color
- By default the overall graph has a black bordered and a light gray margin
- By default the size of the margin around the plot area is automatically calculated

While the above example is a perfectly fine graph it looks a bit poor and we could probably make use of a graph title as well as titles on the axis to explain the units we are working with. So lets change the simple graph in Figure 15.2, “The most simple line graph (`example0-0.php`)” by adding a few lines to set some titles and get the modified graph shown in Figure 15.3, “Adding some titles (`example2.php`)”

**Example 15.2. Adding some titles (`example2.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_line.php');

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

// Size of the overall graph
$width=350;
$height=250;

// Create the graph and set a scale.
// These two calls are always required
$graph = new Graph($width,$height);
$graph->SetScale('intlin');

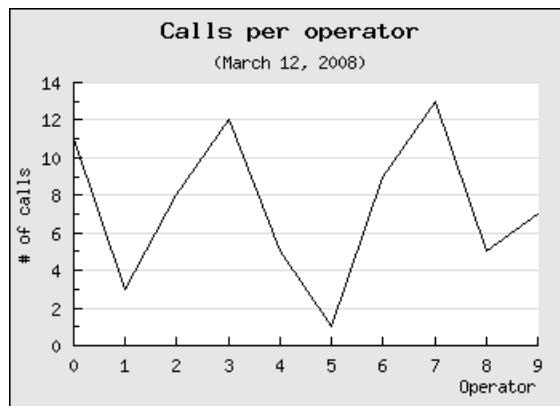
// Setup margin and titles
$graph->SetMargin(40,20,20,40);
$graph->title->Set('Calls per operator');
$graph->subtitle->Set('(March 12, 2008)');
$graph->xaxis->title->Set('Operator');
$graph->yaxis->title->Set('# of calls');

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

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

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

**Figure 15.3. Adding some titles (`example2.php`) [example\_src/  
`example2.html`]**



Worth noting in this example are

- The main graph class instance (\$graph) is used as the base to access most properties of the overall graph
- The margins have been slightly increased to account for the titles of the axis
- The default position for the title of the x-axis is on the far right and for the y-axis it is placed centered in the middle and rotated in a 90 angle (vertical).

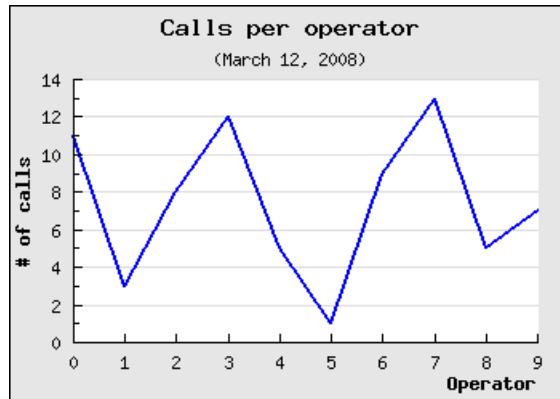
A nice change would now be to have the titles of the axis in a bold font and the line plot a little bit thicker and in blue color. Let's do that by adding the lines

```
<?php
$graph -> title -> SetFont (FF_FONT1 , FS_BOLD);
$graph -> yaxis -> title -> SetFont (FF_FONT1 , FS_BOLD);
$graph -> xaxis -> title -> SetFont (FF_FONT1 , FS_BOLD);
$lineplot -> SetColor ('blue');
$lineplot -> SetWeight (2); // Two pixel wide
?>
```

As was explained in Chapter 8, *Text and font handling* this will adjust the fonts of the titles to make use of a bold variant of the built-in bitmap fonts. Please note the consistent naming conventions used in the library. Most objects support a common set of basic methods to adjust font, size and colors wherever such concepts make sense.

The result of adding these lines are shown in Figure 15.4, “Changing fonts of the axis titles and adjusting plot weight (example3.php)” (click on the link in the title to see the full source).

**Figure 15.4. Changing fonts of the axis titles and adjusting plot weight (example3.php) [example\_src/example3.html]**

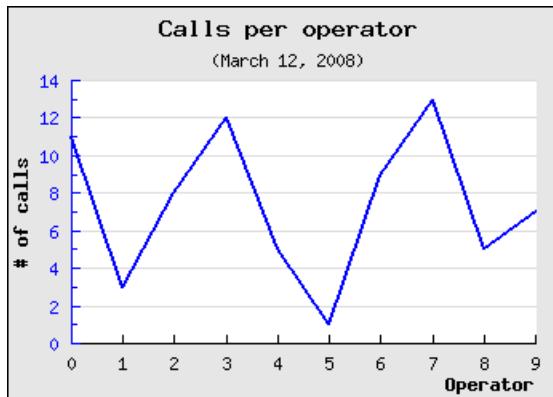


As a final touch lets make the y-axis have the same color as the data series to stronger show the connection between the data and the scale. At the same time we take the opportunity to add a drop shadow to the overall graph. We do both of these things by adding the following two lines at appropriate places in the script

```
<?php
$graph->SetShadow();
$graph->yaxis->SetColor('blue');
?>
```

The results is shown in Figure 15.5, “Adding drop shadow and changing axis color (example3.0.1.php)” below

**Figure 15.5. Adding drop shadow and changing axis color (example3.0.1.php)  
[example\_src/example3.0.1.html]**



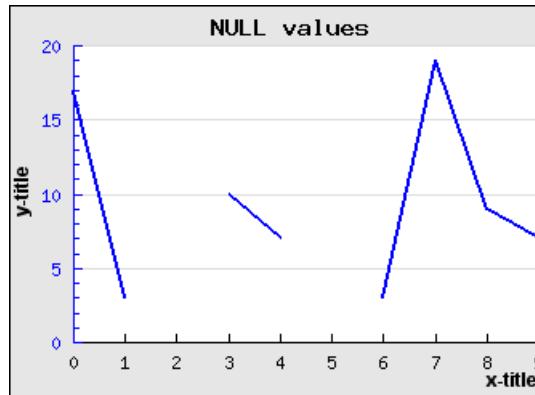
### Tip

Remember that the library has multiple ways to handle null data values as described in Section 13.6, “Different types of NULL data handling”

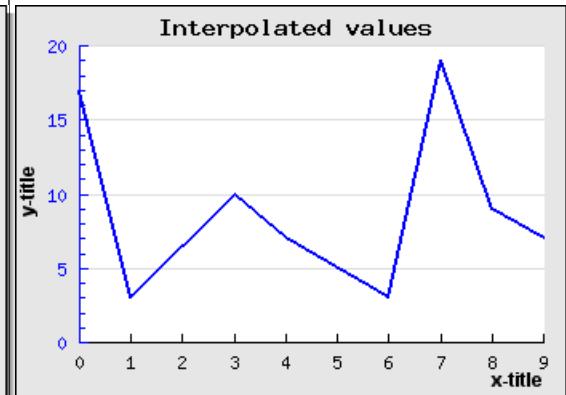
## 15.1.2. Automatic interpolation of unknown data

Line graphs supports automatic linear interpolation of missing data point if the missing data point is replaced by a '-' (hyphen character) for more on handling of NULLs in the input data see Section 13.6, “Different types of NULL data handling”.

**Figure 15.6. Original null values (example3.0.3.php)  
[example\_src/  
example3.0.3.html]**



**Figure 15.7. Using '-' to get interpolated lines (example3.0.2.php)  
[example\_src/  
example3.0.2.html]**



There is one option to control the behavior of the interpolation together with accumulated line plots and that is how to handle the case where the initial or ending data are unknown.

For accumulated line plots (see below) this is especially critical since each value is plotted with an offset of the "previous" plot and hence all values must be known. By default if the first data value is unknown it will be set to the same value as the first found non-null value. It is also possible, for accumulated line plots, to force the first and last unknown value to be equal to 0 (zero). This is controlled by the method

- `AccLinePlot::SetInterpolateMode($aForceZero)`

A value of true for the `for` argument will force any first or last unknown values to be interpreted as 0 for an accumulated line plot.

### 15.1.3. Adding marks to the plot (a.k.a. plot marks)

Another common embellishment of plots is to add markers for each data point. The library supports a large number of built-in plot marks as well as the ability to use arbitrary images as plot marks. Plot marks are instantiated as an instance of class `PlotMark` defined in the module `"jpgraph:plotmark.inc.php"`

There are three types of built-in plot marks

#### 1. Line based.

These marks are drawn directly by the library at the appropriate places in the graph. These marks are simple rectangles, squares, stars etc. The size and colors (both edge and fill) are user adjustable.

#### 2. Image based symbols.

These marks look much more "refined" and are actually small built-in images that are scaled and placed (copied) to the appropriate position in the graph. Since these are predefined images they are only available in a certain number of colors and shapes.

#### 3. Country flags

As was previously discussed the library supports (as of Dec 2008) all known countries and it is possible to use the country flags both as background in the graphs as well as plot marks.

A full list of all available built-in plot marks and their symbolic names are given in Appendix E, *Available plot marks*.

For now lets keep things simple and just add a small triangle at each of the specified data points by adding the lines

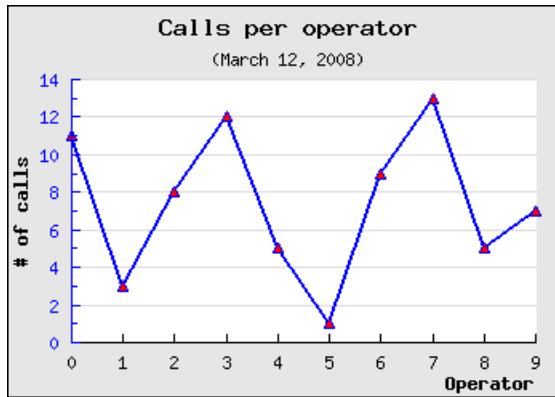
```
<?php
$lineplot->mark->SetType(MARK_UTRIANGLE);
$lineplot->mark->SetColor('blue');
$lineplot->mark->SetFillColor('red');
?>
```

### Caution

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 `SetColor()` for the line since the marks color will always be reset to the lines color when you set the line color.

The result after making these modifications are shown in Figure 15.8, "Adding basic plot marks to the plot (example3.1.php)"

**Figure 15.8. Adding basic plot marks to the plot (example3.1.php)  
[example\_src/example3.1.html]**



### Tip

In addition to the built in plot marks it is also possible to use a user specified image as a plot mark. See Figure 14.11, “Adding a left,right and center footer (footerex1.php)” for an example of how to use this feature. To use this feature the plot mark type is specified as MARK\_IMG and the file name of the image is given. For example the following line will use the image file “myimage.jpg” as plot marks and scale the image to 50% of its original size

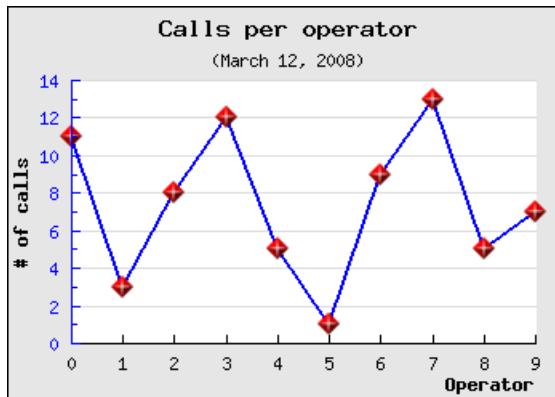
```
$lineplot->mark->SetType(MARK_IMG, 'myimage.jpg', '0.5');
```

As a final example we show an example of using one of the built-in image marks by adding the line

```
$lineplot->mark->SetType(MARK_IMG_DIAMOND, 'red', 0.5);
```

This will add a red diamond mark scaled to 50% of its original size to better fit the overall size of the graph. The result of adding this is shown in Figure 15.9, “Using one of the built-in images as plot mark, MARK\_IMG\_DIAMOND (example3.1.1.php) [example\_src/example3.1.1.html]”

**Figure 15.9. Using one of the built-in images as plot mark,  
MARK\_IMG\_DIAMOND (example3.1.1.php) [example\_src/  
example3.1.1.html]**



## Tip

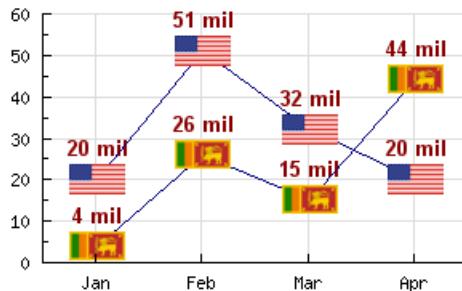
Note that some of the built in images are available in different sizes. The reason is that even though it is possible to scale them by the third argument there is a visual degradation to scale an image larger than its original size since some pixels needs to be interpolated. Reducing the size with a scale < 1.0 gives much better visual appearance.

In order to use one of the available country flags the type is specified as MARK\_FLAG1, MARK\_FLAG2, MARK\_FLAG3 or MARK\_FLAG4 which represent the native size of the flag (in increasing order). The second argument to `SetType()` is either the ordinal index number of the country flag or its short name (as listed Appendix F, *List of all country flags*). The following example illustrates both these methods of specifying the country flags. The two critical lines are

```
<?php
$p1->mark->SetType(MARK_FLAG1,197);
$p2->mark->SetType(MARK_FLAG1,'united states');
?>
```

and the result of creating a graph with some data using country flags as data markers are shown in Figure 15.10, “Using country flags as plot marks (`markflagex1.php`)”

**Figure 15.10. Using country flags as plot marks (`markflagex1.php`) [[example\\_src/markflagex1.html](#)]**



In addition the plot mark formatting shown above plot marks also supports formating through the use of a callback function. The callback function will be passed the y-value as its only argument and the callback function must return an array consisting of three value, weight, color and fill-color. This could be used to for example alter the colors of the plot marks depending on the actual value. A common use of this feature is to create "balloon" scatter plot where a variable sized filled circle is positioned at specific data points. This is a way to create a 2D plot which conveys three values at each data point, x,y and size. In the section on Scatter plot (see Section 15.5, “Scatter graphs”) we show an example of this.

### 15.1.4. Displaying the values at the data points

Lets continue the previous example by making some minor adjustments to also show the values at each data point. The data label at each data point is represented by the instance variable "`$value`". This is an instance of the class `DisplayValue` and all normal text attributes can be adjusted (e.g. color, size, fonts etc). The value is applied to all labels.

In addition to the usual text formatting it is also possible to adjust how the numeric data labels is formatted. This is done by one of two ways.

1. by submitting a suitable format string. This format string follows the same syntax as the `printf()` format string.

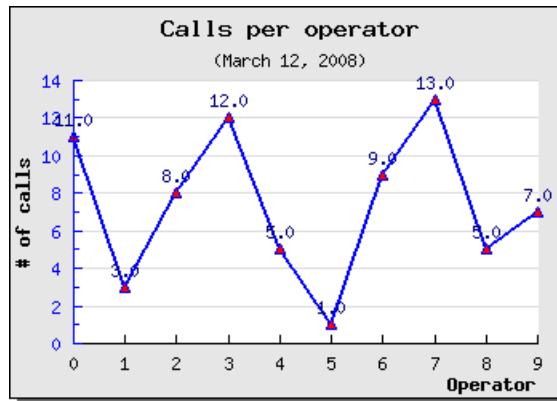
2. by specifying a format callback function. This callback function must take exactly one argument (which is the data value) and return the wanted string that should be displayed.

In order to display the values the first thing that must happen is to enable the values. This is done by a call to

```
$lineplot->value->Show();
```

The result of adding this line is shown in Figure 15.11, “(example3.3.php)”

**Figure 15.11. (example3.3.php) [example\_src/example3.3.html]**

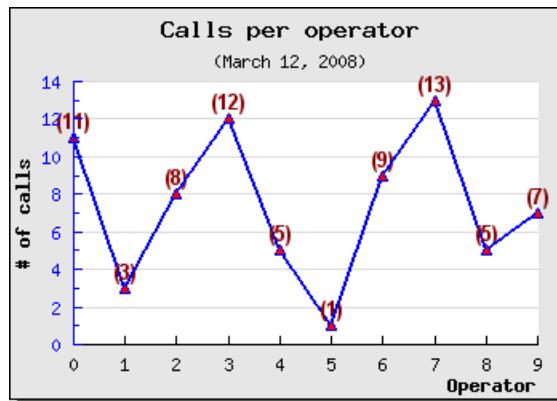


We can now modify the appearance of the labels by changing the font and changing the way the labels are formatted. We do this by adding the following lines to the previous example

```
<?php
$lineplot->value->SetFont(FF_ARIAL,FS_BOLD,10);
$lineplot->value->SetColor('darkred');
$lineplot->value->SetFormat('(%d)');
?>
```

The resulting graph can be seen in Figure 15.12, “Changing the appearance of data labels (example3.4.php)”

**Figure 15.12. Changing the appearance of data labels (example3.4.php) [example\_src/example3.4.html]**



### Tip

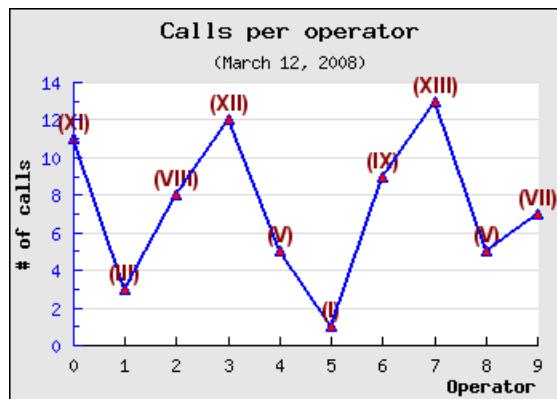
To get 1000' separators use the PHP function number\_format() as a callback function

## Tip

To use method in classes as callback the callback method has to be specified as an array with two string entries. The first entry must be the class and the second entry must be the method name. Note that callback method should be static as no instance context can be created.

As a final example we make use of a callback function to format the values as *Roman Numerals*. The result of this can be seen in Figure 15.13, “Formatting display values as roman numerals (example3.4.1.php)”

**Figure 15.13. Formatting display values as roman numerals (example3.4.1.php) [example\_src/example3.4.1.html]**



## 15.1.5. Adding several data series to the same graph

Up to now all examples we have shown have only had one data series. As was mentioned in the introduction a graph can have an unlimited number of data series (plots) although from a practical consideration (and "viewability") it is probably best to restrict the number of data series in one graph to less than 5-6.

The steps to do this is exactly a repetition of what the examples have shown up to now. The only thing that is needed is to create the second data series, by creating a new instance of a the LinePlot class, set the attributes, and finally add it to the graph.

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

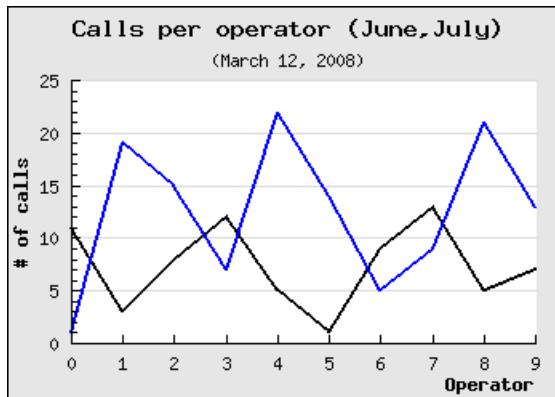
```
<?php
$ydata2 = array(1 , 19 , 15 , 7 , 22 , 14 , 5 , 9 , 21 , 13);

// Create a new data series with a different color
$lineplot2 = new LinePlot ($ydata2);
$lineplot2->SetWeight (2);

// Also add the new data series to the graph
$graph->Add($lineplot2);
```

Making these additions will create the graph in Figure 15.14, “Adding a second data series (example4.php)”,

**Figure 15.14. Adding a second data series (`example4.php`) [`example_src/example4.html`]**



There are two things to note here

1. The Y-scale has changed to accommodate the larger range of Y-values for the second graph.
2. The colors for each successive data series is allocated automatically but of course it is also possible to set the color manually.
3. 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.

### Caution

Do not mix both manually and automatically assigned colors. When the library assigns colors to a new line plot it will not check if a certain color has been previously manually set and used.

## 15.1.6. Adding a second Y-axis

As was discussed in Section 14.6, “Using multiple y-axis” it is possible to add multiple y-axis to a graph. The most common use of this feature is to just use one extra y-axis with a different scale on the right side of the graph. To make this common case as easy as possible to manage the library provides some convenience method to work with one second y-axis (and scale).

The second y-axis is accessed through the “`$y2axis`” property of the Graph class and its use is completely analogue to the primary y-axis (“`$yaxis`”).

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 accommodate for the bigger dynamic range of the second plot. (One other way of solving this particular problem could be to use a logarithmic y-scale).

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 to create a new data array with large values and secondly we need to specify a scale for the Y2 axis. This is done by adding the lines

```
<?php
$y2data = array(354 , 200 , 265 , 99 , 111 , 91 , 198 , 225 , 293 , 251);
$graph->SetY2Scale('lin');
?>
```

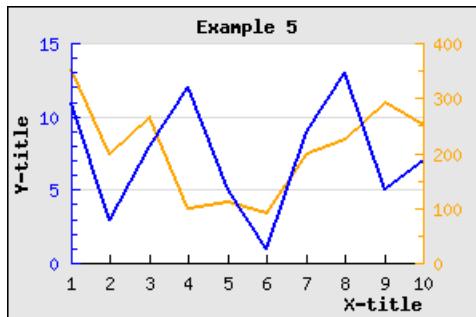
To instruct the library to add this data series to the second y-axis we have to make use of the method

- `Graph::AddY2()`

which is used in analogy with the usual `Graph::Add()`

To make the graph easier to read we set the color of the second y-axis to be the same as the second data series. The result of this is shown in Figure 15.15, “Adding a second y-axis to the graph (`example5.php`)” below.

**Figure 15.15. Adding a second y-axis to the graph (`example5.php`)  
[[example\\_src/example5.html](#)]**



## 15.1.7. Adding a legend box to the graph

Once we have multiple data series there is a need to separate them and that is usually done by adding a legend box with suitable titles. As was shown in Section 14.4, “Adjusting the position and layout of the legend” it is possible to both add a legend box and adjusts its position. Lets now continue the previous example by adding a suitable legend box to separate the two data series.

Each plot type has a '`SetLegend()`' method which is used to name that plot in the legend. So to name the two plots in the previous example we have been working with so far we need to add the two lines

```
<?php
$lineplot->SetLegend('Plot 1');
$lineplot2->SetLegend('Plot 2');
?>
```

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 graph. 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.

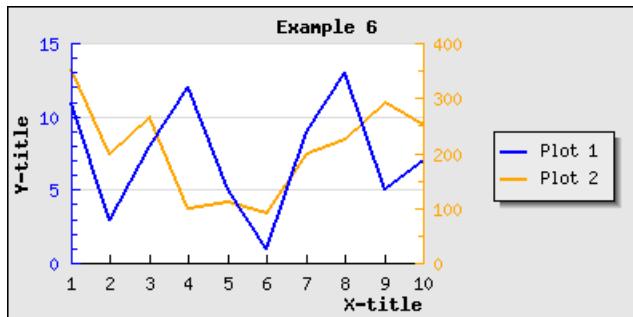
First we increase the right margin and then we place the legend so that it is roughly centered in the right margin area. We will also enlarge the overall image so the plot area doesn't get too squeezed.

The legend properties is accessed through the '`$legend`' property of the graph. So in order to adjust the position (as was described in Section 14.4, “Adjusting the position and layout of the legend” ) we add the line

```
$graph->legend->SetPos(0.05, 0.5, 'right', 'center');
```

This will then give the graph shown in Figure 15.16, “Adding and adjusting the position of the legend box (example6.php)”

**Figure 15.16. Adding and adjusting the position of the legend box (example6.php) [example\_src/example6.html]**

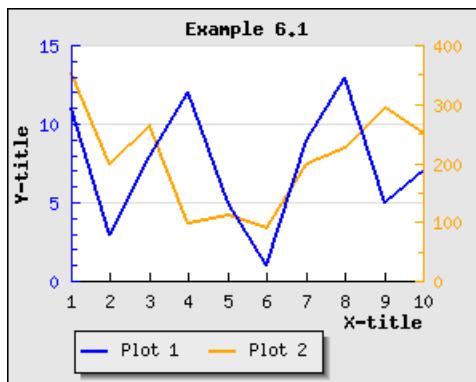


As a reminder we briefly discuss the working of the `SetPos()` method.

The position is specified as a fraction of the overall width and height of the entire graph. This makes it possible for to resize the graph without disturbing the relative position of the legend. The second two arguments specifies the anchor point in the legend box that should be aligned with the specified position.

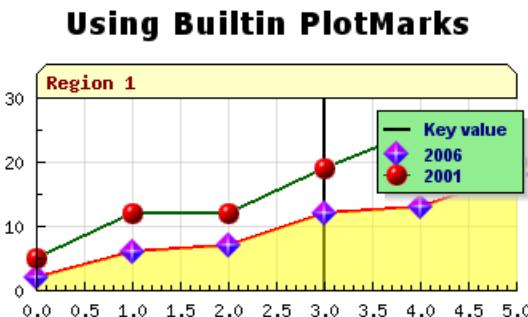
As can be seen in Figure 15.16, “Adding and adjusting the position of the legend box (example6.php)” the legends are by default placed in a column. It is also possible to adjust this by calling either the `Legend::SetLayout()` method or specifically set the number of columns to use by calling `Legend::SetColumns()`. Lets change the layout so the legend texts are set in one row and place the legend box at the bottom of the graph. If we do this we get the result shown in Figure 15.17, “Adjusting the layout of the texts in the legend box (example6.1.php)”

**Figure 15.17. Adjusting the layout of the texts in the legend box (example6.1.php) [example\_src/example6.1.html]**



As a final example lets combine what we have learnt up to now, i.e. adding plot marks, adding multiple data series to the same graph and adding a legend box into one example. In Figure 15.18, “Using plot marks with several data series and a legend (builtinplotmarksex1.php)” we have in addition used the feature with “tabbed” titles which is an alternative way to put titles on graph (as discussed in Section 14.2.3, “Formatting and specifying the titles of the graph”).

**Figure 15.18. Using plot marks with several data series and a legend (builtinplotmarksex1.php) [example\_src/builtinplotmarksex1.html]**



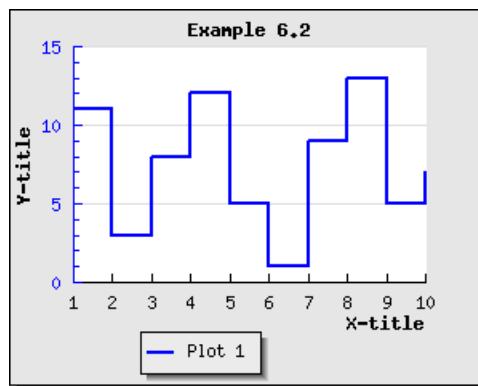
## 15.1.8. Changing the style of the line plot - using the step-style

Step style refers to an alternate way of rendering line plots by not drawing a direct line between two adjacent points but rather draw two segments. The first segment being a horizontal line to the next X-value and then a vertical line from that point to the correct Y-value at that instance. This is also known as "zero-order hold" (as compared with the first-order hold which is an alternative name for combining each data point with a straight line). Enabling step style is done by calling the method

- `LinePlot::SetLineStyle($aFlg=true)` Figure 15.21, “Having the grid line on top of a filled line plot (filledlineex01.1.php)”

In Figure 15.19, “Using the "Step style" for line plots (example6.2.php) ” we have enabled the step style for a basic line graphs to illustrate this concept.

**Figure 15.19. Using the "Step style" for line plots (example6.2.php) [example\_src/example6.2.html]**



## 15.1.9. Optimizing line plot using "fast drawing"

For line plots with a large amount of data point that is drawn with a solid line it is possible to speed up the construction of the graph by calling the method:

- `LinePlot::SetFastStroke($aFlg=true)`

This will avoid some of the overhead associated with drawing lines of arbitrary style. The limitations with this optimization is

1. Only solid lines, no styles on the lines are allowed (including no step-style)
2. No plot marks
3. No value labels
4. No area plot , i.e. no filled line graphs and no filled partial areas

For line plots with a large amount of data point the savings can be quite substantially and in the order of 40-50% speedup.

## 15.1.10. Creating a filled line graphs (a.k.a. area plots)

A filled line plot (also known as an area plot) can be created in two ways depending on whether automatic color handling is sufficient or there is a need to manually specify the color.

### 1. Case 1: Using automatic fill color

In this case there is only need to tell that we want a filled line graph by calling the method

- `LinePlot::SetFilled($aFlg=true)`

The color assigned to the filled area will be set automatically

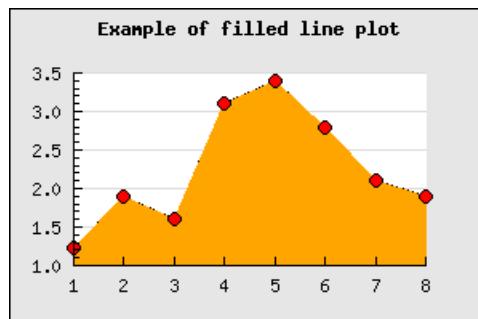
### 2. Case 2: Setting a manual fill color

In this case the method called to set the fill color will make an implicit call to `SetFilled()`. Setting the fill color is done by the method

- `LinePlot::SetFillColor($aColor)`

In Figure 15.20, “A basic filled line graph (`filledlineex01.php`) ” a basic filled line graph is shown which also have plot marks and an adjusted color so that the line and the fill have different colors.

**Figure 15.20. A basic filled line graph (`filledlineex01.php`)  
[[example\\_src/filledlineex01.html](#)]**



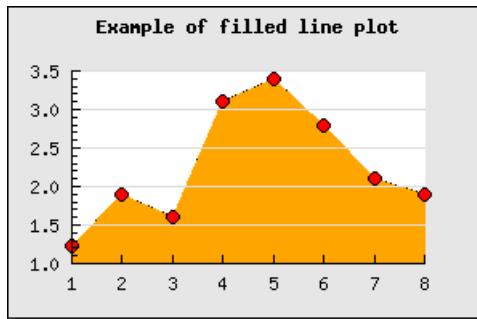
There are a couple of thing worth noting

- 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.

- When using legends with filled line plot the legend will show the fill color and not the bounding line color.
- The area filled is the area between the x-axis at  $y=0$  and the data value
- By default the line color will be the same as the fill color. This means that if a different line color is needed then the call to `LinePlot::SetColor()` must be **after** the call to `LinePlot::SetFillColor()`
- Grid lines are by default drawn behind the plot (or rather the plot is drawn on top of the plot area). However, this can be adjusted so that the grid lines will always be on top of the line plots. This is done by calling the method
  - `Graph::SetGridDepth($aDepth)` where `$aDepth` is either `DEPTH_BACK` or `DEPTH_FRONT` symbolic defines

An alternative way of handling this is to make the fill color semi-transparent by setting the alpha-blending for the fill color. See Section 7.4.1, “Specifying the alpha channel (color transparency)” and Figure 15.18, “Using plot marks with several data series and a legend (`builtinplotmarksx1.php`)” above for a real example. Just making this grid depth adjustments will give the result shown in Figure 15.21, “Having the grid line on top of a filled line plot (`filledlineex01.1.php`)”

**Figure 15.21. Having the grid line on top of a filled line plot (`filledlineex01.1.php`) [example\_src/  
`filledlineex01.1.html`]**



### Tip

By default the fill is only done between  $y=0$  and the line plot. In some cases it might be useful to have the fill go all the way down to whatever the minimum y-value is (for example if the x-axis is always positioned at the minimum y-value). This can be accomplished by calling the method

- `LinePlot::SetFillFromYMin($aFlg=true)`

## Filling from the top

As was mentioned in the previous paragraph the fill normally goes from the bottom and up to the line specified by the data series. Another variant is to have the fill go from the top of the plot area down to the line.

There is primary one use for this type of fill and that is to create a "mask" for a background image to make the illusion that the area below the line is filled with the image while the area above the line is the normal plot background.

This is done by first telling the library the fill shall be from the top with a call to the method

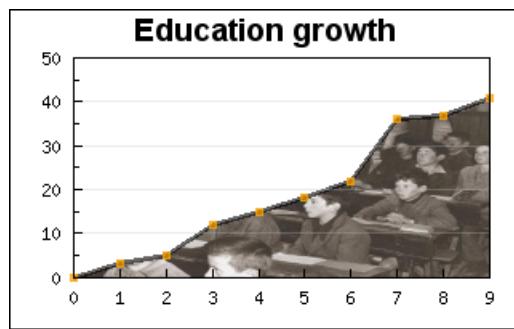
- `LinePlot::SetFillFromYMax($aFlg=true)`

then we create the data series as usual but specify the fill color as the wanted background color and also add the image we want as a background image.

Since we only want to use the first plot as a mask we can also specify the line weight to 0 (to avoid the edge lines of the plot going from the first and last point to the top. To have a nice line we can just add a second line plot which is not filled and is just used to draw the line in our specified color and weight.

An example of how this can look is shown in Figure 15.22, “Creating the effect of an area fill with an image (`lineimagefillex1.php`)” below which shows a fictive growth in education which is illustrated with an old class room photography.

**Figure 15.22. Creating the effect of an area fill with an image (`lineimagefillex1.php`) [[example\\_src/lineimagefillex1.html](#)]**



### Tip

To have the grid lines on top of the area plot (so they are visible since they are by default drawn at the bottom) the depth of the grid lines can be set with a call to the method

- `Graph::SetGridDepth($aDepth)`

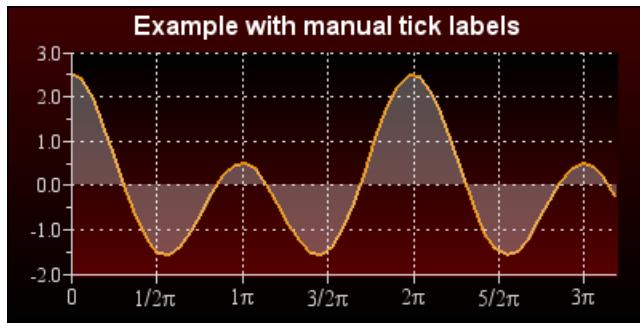
For example as in

```
$graph->SetGridDepth(DEPTH_FRONT);
```

## Filling from 0 or from bottom

As a complement to what was described in the previous section it is also possible to fill from the bottom. By default an area is filled from the 0-line to the boundary of the data series as is shown in Figure 15.23, “Filling from the 0-line (The default) (`manualline3.php`)” below

**Figure 15.23. Filling from the 0-line (The default) (manualtickex3.php)  
[example\_src/manualtickex3.html]**

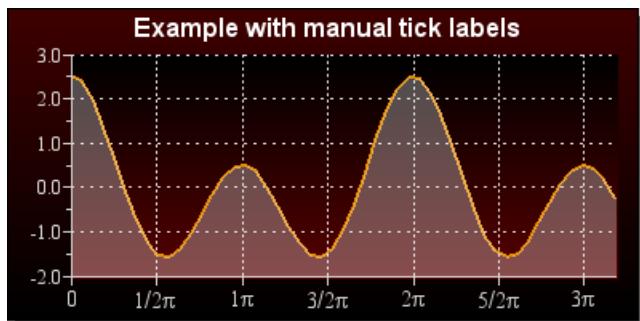


By making a call to

- `LinePlot::SetFillFromYMin($aFlg=true)`

The area will instead be filled from the bottom as is shown in Figure 15.24, “Filling from the bottom (manualtickex4.php) ”

**Figure 15.24. Filling from the bottom (manualtickex4.php)  
[example\_src/manualtickex4.html]**



### Note

In Figure 15.23, “Filling from the 0-line (The default) (manualtickex3.php) ” and Figure 15.24, “Filling from the bottom (manualtickex4.php) ” we have used gradient fill in both the plot area and the amrgin area. This is a feature that was introduced in free-version3.0.5 and in pro-version 3.1.3p

## Using gradient fills

In addition to the solid color fill it is also possible to use gradient fills for are graphs. To specify a gradient fill for a line graph the following method in LinePlot class is used

- `LinePlot::SetFillGradient($aFromColor, $aToColor, $aNumColors=100, $aFilled=true)`

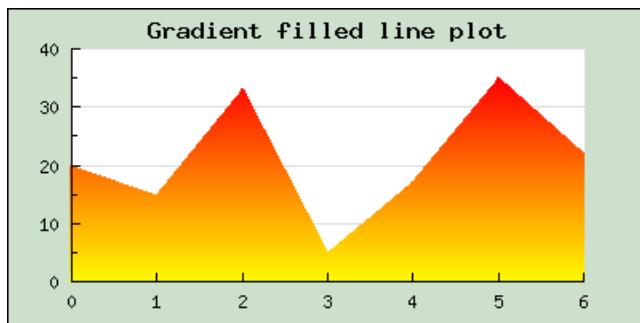
`$aFromColor, $aToColor`, The starting and ending color

`$aNumColors=100`, The number of colors to use in the transition between from and to color

`$aFilled=true`, Enable/disable gradient filling

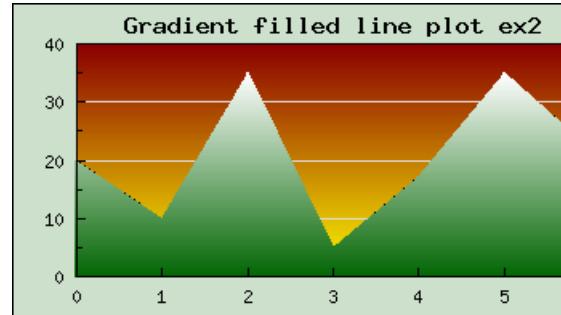
Some examples of typical use of this is shown below

**Figure 15.25.** A basic gradient fill using default values  
(gradlinefillex1.php) [[example\\_src/gradlinefillex1.html](#)]

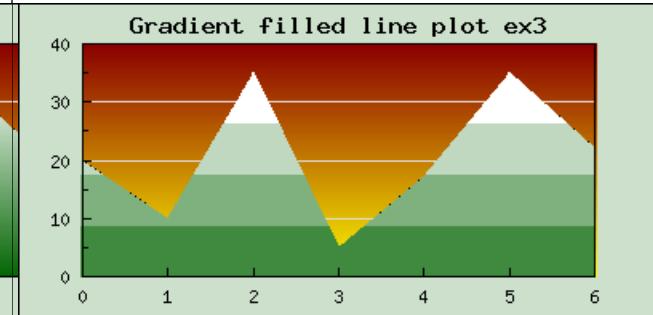


The following two examples shows the effect of changing the number of intermediate colors that are used to get from the "from color" and to the "to color".

**Figure 15.26.** Using the default number of intermediate colors (gradlinefillex2.php) [[example\\_src/gradlinefillex2.html](#)]



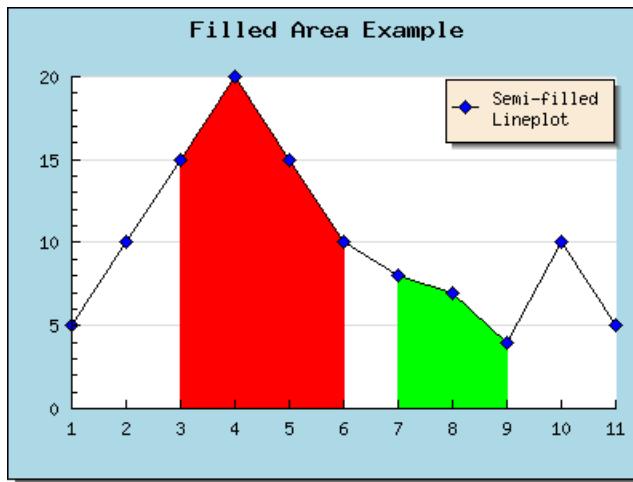
**Figure 15.27.** Only using 4 colors in total between start and finish color (gradlinefillex3.php) [[example\\_src/gradlinefillex3.html](#)]



### 15.1.11. Partially filled area graphs

In addition to filling the entire area between the line plot and the x-axis (at  $y=0$ ) the library also offers the possibility to add areas limited by the line and a specified interval on the x-axis. Several such areas can be added and each area having a different color. In ?? a basic example of how this can look is shown.

**Figure 15.28. Adding two partially filled areas to a line plot (partiallyfilledlineex1.php) [example\_src/partiallyfilledlineex1.html]**



The areas (one or more) are created by calling the method

- `LinePlot::AddArea($aMin=0,$aMax=0,$aFilled=LP_AREA_NOT_FILLED,$aColor="gray9",$aBorder=LP_AREA_BORDER)`

The extension of the area along the x-axis is given by the `$aMin` and `$aMax` values.

The third argument specifies whether the area should be filled or not. This argument can have the values

- `LP_AREA_FILLED`
- `LP_AREA_NOT_FILLED`

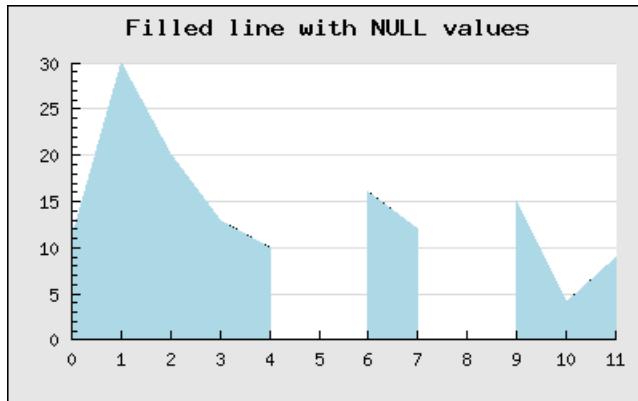
The fourth argument specifies the fill color and the fifth argument specifies if the area should have a border (edge) or not. If the area has a border it will be the same color as the line. The values for the fourth argument can be

- `LP_AREA_BORDER`
- `LP_AREA_NO_BORDER`

## 15.1.12. Filled lines with NULL values

The NULL value handling follows the same principle as was described for line plots. If the NULL value is specified as '-' the line will be interpreted but if it is specified as 'x' it will be broken up as is shown in Figure 15.29, "Area plot with 'x' NULL values (filledlineex03.php)"

**Figure 15.29. Area plot with 'x' NULL values (`filledlineex03.php`)  
[[example\\_src/filledlineex03.html](#)]**



### 15.1.13. 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] 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.

An accumulated graph plot is represented by class `AccLinePlot` which is a container class for line plots. This means that the `AccLinePlot` needs to be "fed" a number of ordinary instances of `LinePlot`.

Any number of ordinary line graphs may be added together (up to the limit of readability of the plot).

For example, to add three line plots in an accumulated line plot graph the following code is needed

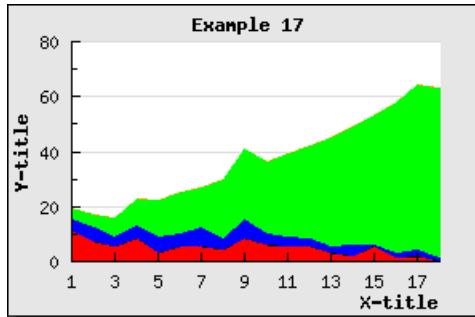
```
<?php
// First create the individual plots
$p1 = new LinePlot($datay_1);
$p2 = new LinePlot($datay_2);
$p3 = new LinePlot($datay_3);

// 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);
?>
```

Using some random data for the areas can produce the result shown in Figure 15.30, "A basic accumulated area plot (`example17.php`) "

**Figure 15.30. A basic accumulated area plot (`example17.php`)  
[`example_src/example17.html`]**



### 15.1.14. Accumulated line graphs with given X-labels

Creating an accumulated line plot, (or a filled accumulated area plot), with multiple data sets at given x-coordinates poses some specific problems when the coordinates for the different data sets are not given at the same x-coordinates. This is a generic problem and has nothing to do with library in particular. This section wil deal with one possible way of handling this by making sure that missing data points are created if they are "missing".

To understand the problem we will make one simplifications (that is of no real consequence for the end result) that can be stated as

1. the X-coordinates for all data tuples are whole positive number.
2. the X-coordinates are in sorted order (non-descending)

The core issue can be illustrate as follows. Lets assume that we want to make an accumulated graph showing the two data sets

```
data set 1 == (0,5), (2,10), (3,10), (5,20)
data set 2 == (0,7), (1,12), (2,5), (5,10)
```

In the above notation the tuple (0,5) means a data point with X-coordinate = 0 and Y-coordinate = 5.

What the library now needs to do is to first plot data set 1. No problem. When it then becomes time to plot the second data set we face an issue. The only points where we now the Y-value of data set 1 is at the given discrete points (0,2,3,5).

Plotting the first tuple for data set 2 shown above gives an absolute starting point at

```
(0,5+7) == (0,12)
```

The next data point we know for data set 2 is (1,12) so we need to plot this. But now we can see that we do not know the value for data set 1 at X-coordinate = 1. We only know the values at coordinates 0 and 2. This gives us a problem. We need to know at what offset we should plot this data point in data set 2 and we have no direct way of calculating this.

Now, one might argue that we could just interpolate between the data points (0,5) and (2,10) the Y-value at X=1 (doing a linear interpolation this would give the data point (1,7.5)) so why doesn't the library simply do this? It surely could be done.

## Note

In real life using this approach would be much more complex. First of all we need to create a linear succession of all X-values used in all data sets to create an ordered set and then fill in the blanks so that all data sets have values at all given X-coordinates. Those of you familiar with signal processing will recognize this as an (almost) up-sampling of the original data sets followed by a low pass filter.

However, by design the library doesn't do this. The crucial observation here is that it can not be a graphic libraries responsibility to "create" missing data points by making assumption that a particular polynomial interpolation is valid (in this case a first degree approximation). What if a linear interpolation is not representative for the data set given? Perhaps a second degree approximation would be more accurate.

So, this kind of data preparation must be done in the domain of the given data set where knowledge of the underlying data will allow an accurate preparation of the input to a graphing script if we insist of plotting an accumulated graph. One could argue that accumulated data plots can only be done for data series with the same X-coordinates.

## Preparing the input data

So what if we are still required to do an accumulated plot even when we don't have all the data sets at the same X-coordinates? Going back to our original two data sets, hereafter referred to as DS1 and DS2 there are 2 manual steps (as described above) that needs to happen.

1. Identify all X-data points that needs to exist
2. Create values for all data sets at those points

So, in DS1 and DS2 the union of the two data sets X-coordinates are

```
x_coordinates == union(DS1_x, DS2_x) == 0,1,2,3,5
```

This will force us to augment the two data sets as

```
data set 1 == (0,5), (1,??), (2,10), (3,10), (5,20)
data set 2 == (0,7), (1,12), (2, 5), (3,??), (5,10)
```

Where I have added '??' to indicate values that needs to be computed in order to draw an accumulated line/area plot at specific values. Now assume that we are able to find the missing data for these points by some method to be

```
data set 1 == (0,5), (1, 8), (2,10), (3,10), (5,20)
data set 2 == (0,7), (1,12), (2, 5), (3, 2), (5,10)
```

Are we now ready to plot these data sets? Unfortunately not quite. The remaining problem is that since the library only handles accumulated plots without a given X-coordinate (using an X-coordinate for the individual line plots will have no affect - and its behaviour is undefined). This means that the data points are assumed to be equ-distance apart - and this is almost true for the data sets above. There is 1 unit between them apart from the two last tuples which in fact have a distance of 2 units. In fact the library only plots data sets with a given Y-coordinate and then assumes that the x-coordinate is a linear ordering of (0,1,2,..)

So in order to create a linear equ-distance ordered set we need to further augment the two data sets as

```
data set 1 == (0,5), (1, 8), (2,10), (3,10), (4,??), (5,20)
```

```
data set 2 == (0,7), (1,12), (2, 5), (3, 2), (4,??), (5,10)
```

So this means that we need to manually calculate another interpolated value. If we know we can make a linear interpolation (or perhaps find the data at this point) it will give us

```
data set 1 == (0,5), (1, 8), (2,10), (3,10), (4,15), (5,20)
data set 2 == (0,7), (1,12), (2, 5), (3, 2), (4, 6), (5,10)
```

This final data set is now ready to be sent to the `AccLinePlot` class. It is left as a (non-trivial) exercise to the reader to define and implement a function that performs the steps outlined above to create proper data sets before reading on.

## Creating plots with non-trivial X-coordinates

With non-trivial X-coordinates we mean for example timestamps or perhaps real numbers. For timestamps it is not so difficult. What we need to do is to identify the proper interval (in the original timestamp domain) and then create a mapping between that domain and the natural numbers (0,1,2,3,...).

The reason for this is that the library only accepts Y-coordinates as argument to the accumulated data series and will make the implicit assumption that when it plots the data it will plot the data points at consecutive values as if the X-coordinates had been given as (0,1,2,3,...). Hence we need to manually prepare the data to match this format.

As the final step we manually set the labels for the X-axis according to our interpretation. An example (with some code snippets) will make this approach clear.

### Example - using timestamps

Assume we have the two data sets with timestamps

```
DS1 == (1212199200,12), (1212210000,20), (1212213600,30)
DS1 == (1212199200,12), (1212206400, 8)
```

and we know that the sampling interval between the data points are 7200s (=2 min). Following the same principle as above we need to find the additional values

```
DS1 == (1212199200,12), (1212206400,??), (1212210000,20), (1212213600,30)
DS1 == (1212199200,12), (1212206400, 8), (1212210000,??), (1212213600,??)
```

further assuming that we (by some method) can find these values we can then interpret this data as

```
DS1 == (1212199200,12), (1212206400,16), (1212210000,20), (1212213600,30)
DS1 == (1212199200,12), (1212206400, 8), (1212210000, 0), (1212213600, 0)
```

In the above we have made the explicit assumption that unknown data points at the end can be interpreted as 0 in this particular application.

We now have an ordered sequence of these tuples and we can imagine a mapping that will allow us to write these sequences as

```
DS1 == (0,12), (1,16), (2,20), (3,30)
DS1 == (0,12), (1, 8), (2, 0), (3, 0)
```

The mapping for this is  $xi = 1212199200 + 7200*i$ ,  $i=0..3$  which we use when we put the final labels in the graph.

The only steps that remain to handle timestamps is to manually replace the X-scale (which in this case would be 0,1,2,3) with the calculated values according to the mapping given above.

We do this by creating an array of the timestamps we need to plot and then replace them - in situ - with an application of the standard PHP function `array_walk()` which applies a user defined function to each value in an array and replaces that value with the return value of the user function. In this case we create a user function that implements the mapping stated above with the additional twist that given an argument as a time stamp it returns a suitable human format for that time stamp.

The following code fragments shows how this could be done

```
// Some userdefined human readable version of the timestamp
function formatDate(&$aVal) {
 $aVal = date('Y-m-d H:i', $aVal);
}

$timeStamps = array(212199200, 1212206400, 1212210000, 1212213600);

array_walk($time, 'formatDate');
```

when we now have the labels in a nice human readable format we can put them on the scale labels with

```
$graph->xaxis->SetTickLabels($timeStamps);
$graph->xaxis->SetLabelAngle(90);
```

though strictly not necessary we have also tilted the labels 90 degrees in order to minimize the risk the labels overwrite each other.

If we still think that the labels are too close together ea we can chose to only label every second tick mark. We do this with a call to

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

#### Example using real (i.e. floating point) x-coordinates

In principle this is handled in the same way as what we shown above for timestamps. The additional complexity here spells rounding errors. When we establish the equidistant interval between each data point it will be a real number, potentially an irrational number, which means that we cannot represent it exactly and adding the interval repeated times might cause rounding errors if we are not careful.

Secondly we need to find a mapping between the ordered sequence of the real numbers we have as X-coordinates and the natural numbers which are the implicit X-coordinates assumed by the library.

## A full example

In the example below we artificially create some data sets where all the sets have values at all specified timestamps with the following code

```
<?php
//Create some test data
$xdata = array();
$ydata = array();

// Timestamps - 2h (=7200s) apart starting
$sampling = 7200;
$n = 50; // data points
```

```
// Setup the data arrays with some random data
for($i=0; $i < $n; ++$i) {
 $xdata[$i] = time() + $i * $sampling;
 $ydata[0][$i] = rand(12,15);
 $ydata[1][$i] = rand(100,155);
 $ydata[2][$i] = rand(20,30);
}
?>
```

Since the xdata array is given as timestamps we need to make this more human readable by converting the timestamp using the `date()` function. To do this we create an auxiliary helper function and then use the `array_walk()` standard array function to apply this formatting to all existing values in the timestamp array as follows.

```
// Formatting function to translate the timestamps into human readable labels
function formatDate(&$aVal) {
 $aVal = date('Y-m-d H:i',$aVal);
}

// Apply this format to all time values in the data to prepare it to be display
array_walk($time,'formatDate');
```

The core of the script can now be written. For a change we make some adjustment from the default values of colors and tick mark positioning as a reminder that there is a lot of flexibility in creating the graphs.

```
<?php
// Create the graph.
$graph = new Graph(700, 400);
$graph->title->Set('Accumulated values with specified X-axis scale');
$graph->SetScale('datlin');

// Setup margin color
$graph->SetMarginColor('green@0.95');

// Adjust the margin to make room for the X-labels
$graph->SetMargin(40,30,40,120);

// Turn the tick marks out from the plot area
$graph->xaxis->SetTickSide(SIDE_BOTTOM);
$graph->yaxis->SetTickSide(SIDE_LEFT);

$p0 =new LinePlot($a);
$p0->SetFillColor('sandybrown');
$p1 =new LinePlot($b);
$p1->SetFillColor('lightblue');
$p2 =new LinePlot($c);
$p2->SetFillColor('red');
$ap = new AccLinePlot(array($p0,$p1,$p2));

$graph->xaxis->SetTickLabels($time);
$graph->xaxis->SetTextLabelInterval(4);

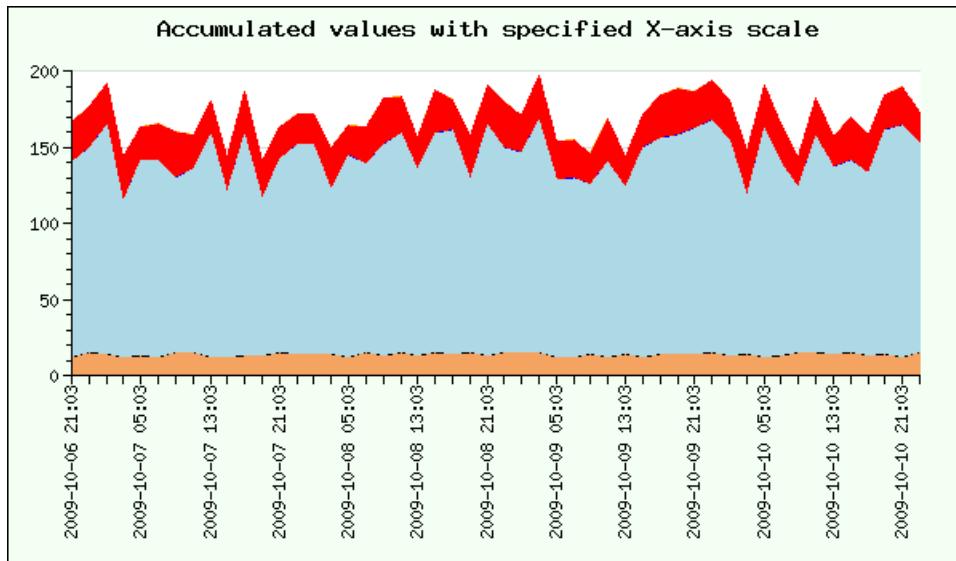
// Add the plot to the graph
$graph->Add($ap);
```

```
// Set the angle for the labels to 90 degrees
$graph->xaxis->SetLabelAngle(90);

// Send the graph back to the browser
$graph->Stroke();
?>
```

The resulting image will now look something like what is shown in ??

**Figure 15.31. Area plot with specified x coordinates  
([prepaccdata\\_example.php](#)) [[example\\_src/prepaccdata\\_example.html](#)]**



## Helper function to create interpolated data

The function `InterpolateData()` below takes two array of arrays and one integer as arguments. The first array of arrays contains the X-coordinates for each data set and the second array of arrays contains the Y-coordinates for all the data sets. The final integer argument is the distance (or sample rate) that should be assumed between each X-coordinate.

The function will return a tuple. The first element in the returned tuple is a single array with all the X-values that should be used and the second element is an array of arrays with all the Y-data sets with all data specified for each X-coordinate. Any missing Y values are interpolated using a linear interpolation schema.

So using our first example above as demonstration this would be handled as

```
<?php
$datax = array(
 array(0,2,3,5),
 array(0,1,2,5));

$datay = array(
 array(5,10,10,20),
 array(7,12,5,10));
```

```

list($datax, $datay) = InterpolateData($datax, $datay);

// $datax = array(0,1,2,3,4,5)
// $datay = array(array(5, 8,10,10,15,20),
// array(7,12, 5, 2, 6,10));
?>

```

One possible implementation of this function is given below. It has primarily been written for clarity and not necessary high performance. To interpolate the "missing" Y-values a linear approximation is assumed.

```

<?php
function InterpolateData($aXData,$aYData,$aSampleInterval=1) {

 // First do some sanity checks on the input data
 $nx = count($aXData);
 $ny = count($aYData);
 if($nx != $ny)
 return array(false,-1);

 for($i=0; $i < $nx; ++$i) {
 if(count($aXData[$i]) != count($aYData[$i]))
 return array(false,-2);
 }

 // Create the sorted union of all X-coordinates
 $unionx = array_union($aXData);
 $length = count($unionx);

 // We now have to make sure that the distance between all
 // X-coordinates is 1 unit of the sample interval. If not
 // we will have to insert suitable X-value
 $i=1;
 while($i < $length) {
 $missing = 0;
 $diff = $unionx[$i] - $unionx[$i-1];
 if($diff != $aSampleInterval) {

 // Sanity check to make sure sample interval is an even multiple
 // of the distance between the given X-coordinates
 if($diff % $aSampleInterval !== 0) {
 return array(false,-4);
 }

 $missing = $diff / $aSampleInterval - 1;
 $fill = array();
 for($j=0; $j < $missing; ++$j) {
 $fill[$j] = $aSampleInterval*($j+1)+$unionx[$i-1];
 }
 $unionx = array_merge(
 array_slice($unionx,0,$i),$fill,array_slice($unionx,$i));
 }
 $i += $missing+1;
 $length += $missing;
 }
}

```

```

}

if($length != count($unionx)) {
 // Internal error check
 return array(false,-3);
}

// Now loop through all the individual data sets and find out
// which x-data is missing and hence needs to be interpolated
$n = count($aXData);

for($i=0; $i < $n; ++$i) {
 $missing_values = array_diff($unionx, $aXData[$i]);

 // Now find the position of each missing X-coordinate
 // and use that position in the corresponding Y array
 // to insert an interpolated value
 $m = count($missing_values);
 foreach($missing_values as $key => $val) {
 $idx = array_search($val,$unionx);

 // Now split the Y-array at that position and insert
 // a new sentinel value
 if($idx >= 0) {
 $aYData[$i] = array_merge(
 array_slice($aYData[$i],0,$idx),
 array(NULL),
 array_slice($aYData[$i],$idx));
 }
 }

 // The next step is to actually calculate an interpolated value
 // for the Y-coordinates we don't have. As a special case any
 // beginning or ending non-defined coordinates are set to 0

 // Set all beginning NULL to 0
 for($j=0; $j < $length; ++$j) {
 if($aYData[$i][$j] !== NULL)
 break;
 $aYData[$i][$j] = 0;
 }

 // Set all ending NULL to 0
 for($j=$length-1; $j >= 0; --$j) {
 if($aYData[$i][$j] !== NULL)
 break;
 $aYData[$i][$j] = 0;
 }

 // Calculate the remaining missing values as a linear
 // interpolation and keeping in mind that there might be
 // multiple missing values in a row.
 $j = 0;
 while($j < $length) {

```

```

 if($aYData[$i][$j] === NULL) {
 // How many unknown values in a row?
 $cnt = 1;
 while($j+$cnt < $length && $aYData[$i][$j+$cnt]==NULL) {
 ++$cnt;
 }

 if($cnt == 1) {
 $aYData[$i][$j] = ($aYData[$i][$j-1]+$aYData[$i][$j+1])/2;
 }
 else {
 $step = ($aYData[$i][$j+$cnt] - $aYData[$i][$j-1])/($cnt+1);
 for($k=1; $k <= $cnt; ++$k) {
 $aYData[$i][$j+$k-1] = $step*$k+$aYData[$i][$j-1];
 }
 }
 ++$j;
 }

 return array($unionx,$aYData);
 }

//-----
// Helper function to create the union of two arrays
//-----

// Create the sorted union of all numeric arrays given as argument
function array_union($a) {

 $n = count($a);
 $res = $a[0];
 for($i=1; $i < $n; ++$i) {
 $res = _array2_union($res,$a[$i]);
 }
 sort($res);
 return $res;
};

// Return the union between two numeric arrays
function _array2_union($a,$b) {

 if($a == NULL) return $b;
 if($b == NULL) return $a;

 // A standard "trick" to calculate the union of two arrays
 return array_merge(
 array_intersect($a,$b),
 array_diff($a, $b),
 array_diff($b, $a));
}
?>
```

## 15.1.15. Constructing smooth line plots with Cubic Splines

The library support interpolation between data point by the use of cubic splines. This will make the implicit assumption that the underlying phenomenon that is plotted can be represented by a 3:rd degree polynomial between the given data points (also known as a control points).

Cubic splines have the property that the constructed line will pass through all control points given.

To construct a spline you both the X and Y coordinates for the known data points are needed since the library can make no assumption on the step size between the data points.

The cubic spline functionality in the library is encapsulated in the class `Spline` which is define the module "`jgraph_regstat.inc.php`" which must be added to the included files in the script.

The first step is to construct new `Spline` instance. This class is instantiated by calling the constructor with the two known data arrays (X and Y) as follows.

```
$spline = new Spline($xdata , $ydata);
```

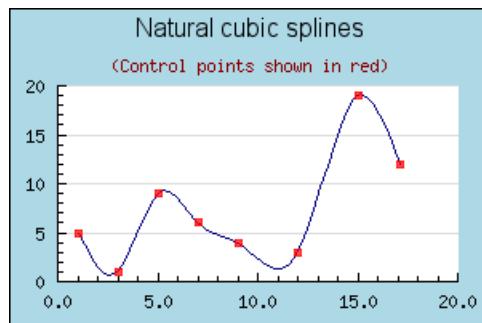
This call initializes the spline with the data points given. These data points are also known as Control points for the spline. This helper class doesn't draw any line itself. Instead it is merely used to get a new (larger) data array which have all the interpolated values. These new value are then used to make the actual line plot. This way gives great flexibility in how to use the interpolated data points.

To get the y- and x-axis data to be plotted we call the method `Spline::Get()` to get an interpolated array containing a specified number of points. So for example the line

```
list($sdatax , $sdatay) = $spline->Get(50);
```

will construct the two new data arrays '\$sdatax' and '\$sdatay' which contains 50 data points. These two arrays are constructed from the control points we specified when we created the '\$spline' object. These data arrays are then used to make the actual plot just as for a "standard" plot. In Figure 15.32, "Constructing a smooth spline curve from 8 control points (`splineex1.php`)" we have used 8 control points to construct a spline.

**Figure 15.32. Constructing a smooth spline curve from 8 control points (`splineex1.php`) [example\_src/splineex1.html]**



In order to make the example more interesting we actually used two plots. First a line plot to get the smooth curve and then a standard scatter plot (discussed later in this manual) which is used to illustrate where the control points are situated.

## Note

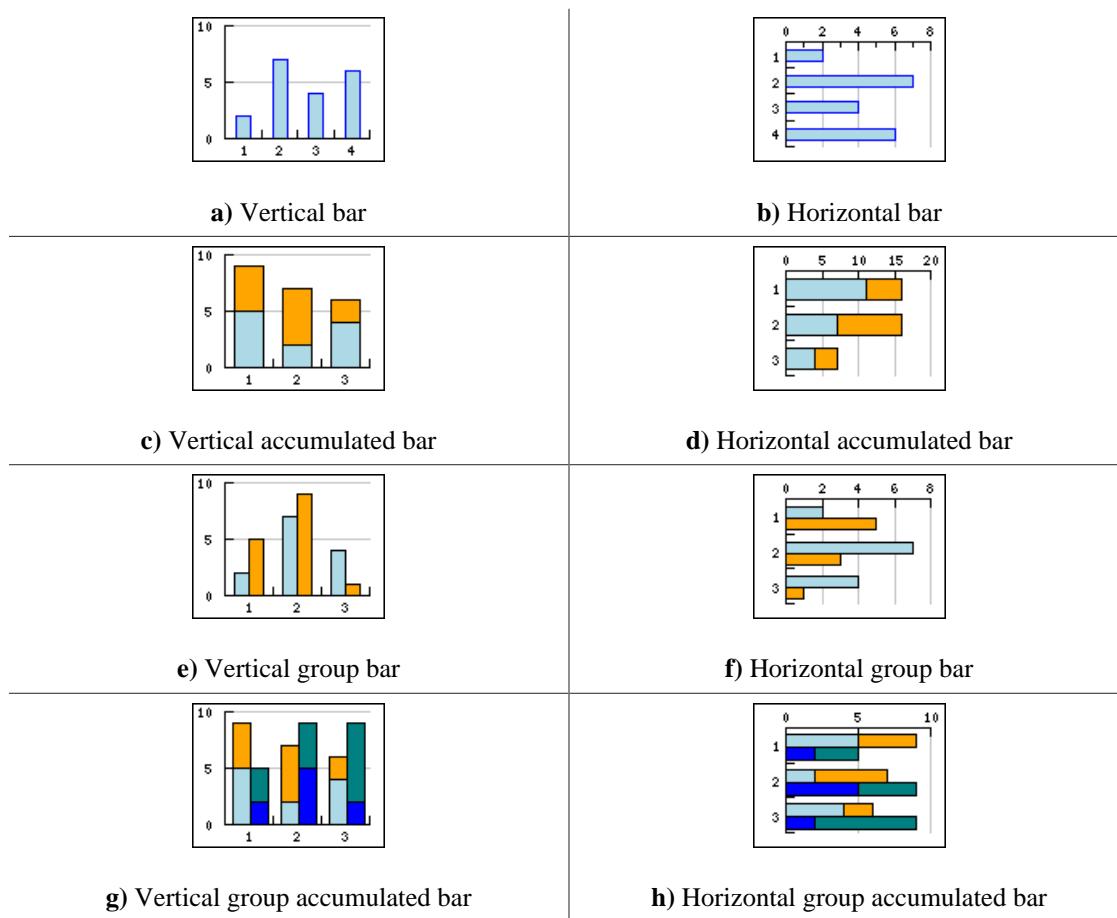
The library also support the construction of *Bezier curves* which is built on a similar concept of constructing a smooth line from a number of control points. The biggest difference between cubic splines and Bezier curves is that while a cubic spline is always guaranteed to pass through all control point a Bezier curve will in general not pass through any control points. Instead the Bezier control point are outside the curve and affects the curvature of the curve. Bezier type curves are not in general used together with data visualization but are instead used to create specific curves, usually for CAD and 3D graphic purposes. In the discussion of Canvas graphs, see Section 17.3, “Canvas graphs”, we discuss this further.

## 15.2. Bar graphs

The library supports 2D vertical and horizontal bar plots as was shown in the introduction section . To use bar plots the bar plot module "jpgraph\_bar.php" must be included in the script.

There are eight fundamental types of bar graphs supported by the library. Examples of the available types are shown in Figure 15.33, “Different types of supported bar graphs”

**Figure 15.33. Different types of supported bar graphs**



Using bar plots is straightforward and works in much the same way as line plots as was discussed in the previous sections.

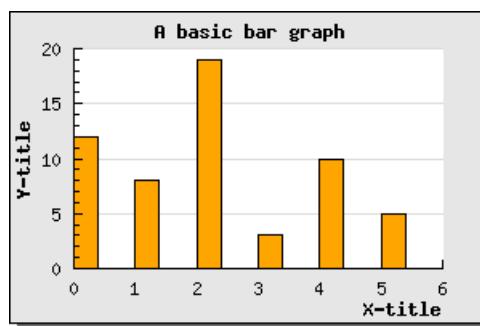
An instance of class `BarPlot` is created with the wanted data and is then either added directly to the graph to create a basic bar plot or is enclosed in with one of the container classes `AccBarPlot` or `GroupBarPlot` which are then added to the graph.

There is however one crucial change that is usually made. The x-scale is usually specified as a "text" type scale. The reasons this are primarily two

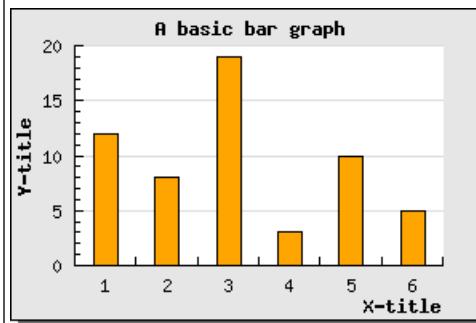
1. to get the alignment of the labels to be between the tick marks and not at the tick marks as is normal for line plots
2. to get the bars to be aligned at the center between the tick marks

The following two examples shows the difference between using an integer scale in Figure 15.34, "Using "int" scale for the x-axis (example19.1.php)" and a text scale in Figure 15.35, "Using "text" scale for the x-axis (example19.php)"

**Figure 15.34. Using "int" scale for the x-axis (example19.1.php)**  
[example\_src/  
**example19.1.html**]



**Figure 15.35. Using "text" scale for the x-axis (example19.php)**  
[example\_src/example19.html]



As can be seen in Figure 15.34, "Using "int" scale for the x-axis (example19.1.php)" the bars have there left edge aligned with the data value and the tick mark (the tick mark can not be seen since it is aligned exactly with the left edge of the bar). In contrast using a text scale in Figure 15.35, "Using "text" scale for the x-axis (example19.php)" adjusts the alignment of tick marks and labels in a way that is more commonly used with bar graphs.

Some other commonly used method to change the appearance of the bar graphs are

- `BarPlot::SetFillColor($aColor)`

This is used to specify the fill color of the bar. The argument can also be an array and in that case each color in the array is used for individual successive bars.

- `BarPlot::SetFillGradient($aFromColor, $aToColor=null, $aStyle=null)`

Specifies a gradient fill style for the bars. See ?? for details.

- `BarPlot::SetPattern($aPattern, $aColor='black')`

Specifies a pattern to be used to fill the bars. See ?? for details

- `BarPlot::SetWidth($aWidth)`

Specifies the width of the individual bars. Of this is an integer value > 1 it is interpreted as the absolute width in pixels. If the values instead is a real number in the range [0,1] it is interpreted as the fraction of the width between the tick marks. By default the width is set to 0.4

### 15.2.1. Accumulated bar plots

Accumulated bar plots will show several data series stacked on top of each other in each bar. They are the barplot variant of accumulated area plots as was discussed previously.

An accumulated bar plot is made by aggregating one or more basic bar plots in the container class `AccBarPlot` as the following code snippet shows

```
<?php
// Some data
$data1y = ...
$data2y = ...

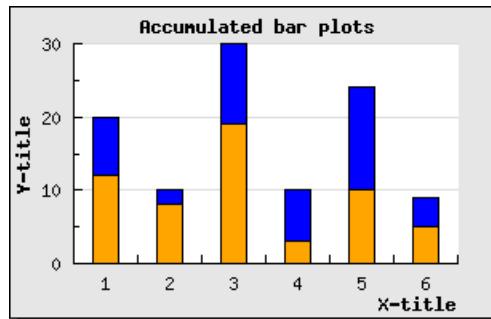
// Create two bar plots
$b1plot = new BarPlot($data1y);
$b1plot->SetFillColor('orange');
$b2plot = new BarPlot($data2y);
$b2plot->SetFillColor('blue');

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

// Add the accumulated plot to the graph
$graph->Add($gbplot);
?>
```

An example of an accumulated bar plot is shown in Figure 15.36, “An accumulated bar plot (example23.php)”

**Figure 15.36. An accumulated bar plot (example23.php) [example\_src/example23.html]**



There are some subtleties when it comes to the formatting of the frames around each bar in an accumulated bar plots that might be useful to know. The basic ambiguity that exists is that when we stack the bars on top of each other to create a new accumulated bar each individual bar has properties that was (or could) be set when each individual barplot was created like the frame around the plot.

For example, take the following basic accumulated bar plot (partial script)

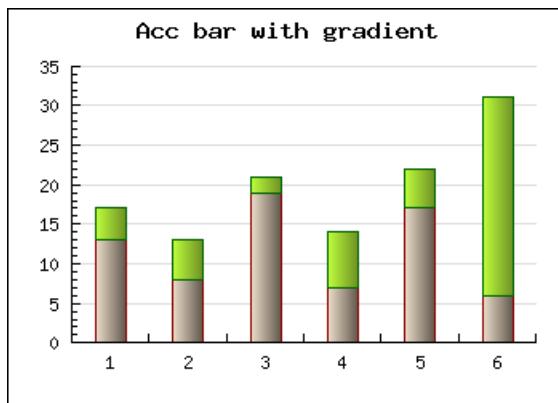
```
<?php
// Create the first bar
$bplot = new BarPlot($datay1);
$bplot->SetFillGradient('AntiqueWhite2','AntiqueWhite4:0.8',GRAD_VERT);
$bplot->SetColor('darkred');

// And the second bar
$bplot2 = new BarPlot($datay2);
$bplot2->SetFillGradient('olivedrab1','olivedrab4',GRAD_VERT);
$bplot2->SetColor('darkgreen');

// Join them in an accumulated (stacked) plot
$accbplot = new AccBarPlot(array($bplot,$bplot2));
$graph->Add($accbplot);
?>
```

As you can see on line 5 and on line 10 we have set the frame color differently in the two individual plots. However, the accumulated bar plot also have a frame color property (and a weight as well) so what shall we use? If we run a full example based on the lines above the result is shown in Figure 15.37, “Accumulated bar with individual frame colors (accbarframeex01.php)” below.

**Figure 15.37. Accumulated bar with individual frame colors (accbarframeex01.php) [example\_src/accbarframeex01.html]**



As can be seen from the graph the bar around each part has the color on the frame that was set on the individual bar. There is one exception though. The line that separates the two bars are shared and will always follow the color of the top bar.

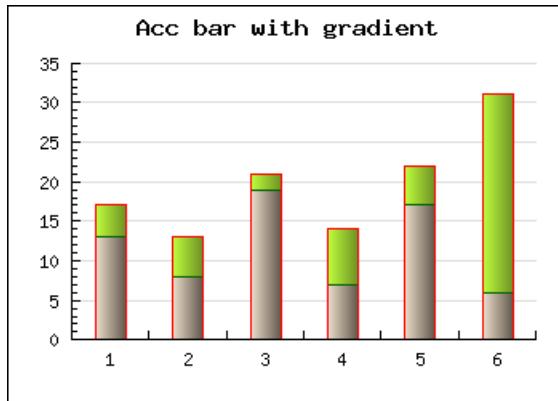
If we instead specify a frame for the accbar by adding the lines

```
<?php
$accbplot = new AccBarPlot(array($bplot,$bplot2));
$accbplot->SetColor('red');
$accbplot->SetWeight(1);
$graph->Add($accbplot);
?>
```

The result would be as is shown in Figure 15.38, “Accumulated bar with unit frame color (accbarframeex02.php)” and the properties of the accumulated bar takes precedence, again with one exception. The divider lines inside the bar is still controlled by the individual plot. By default the line weight

on the accumulated bar is 0 which means that it will not be drawn, that is why we have to set a line weight on line 4 above.

**Figure 15.38. Accumulated bar with unit frame color (accbarframeex02.php)  
[example\_src/accbarframeex02.html]**

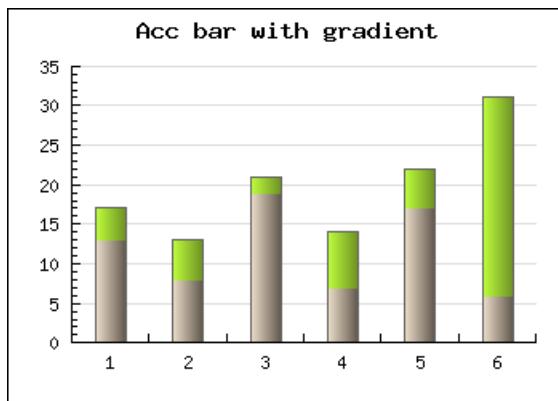


If we instead were to set the individual line weight to zero, i.e.

```
<?php
$bplot->SetWeight(0);
$bplot2->SetWeight(0);
?>
```

and keep the overall frame the result would become as shown in Figure 15.39, “Setting individual frames to weight=0 (accbarframeex03.php)” below

**Figure 15.39. Setting individual frames to weight=0 (accbarframeex03.php)  
[example\_src/accbarframeex03.html]**



## 15.2.2. Grouped bar plots

This uses the same principle as accumulated bar plots but instead of stacking the data series on top of each other they are shown together for the same x-value.

These types of bar graph is used to easy group two or more bars together around each tick (x-value). The bars will be placed immediately beside each other and as a group centered on each tick mark (or between if a text scale is used). A grouped bar is created by aggregating two or more ordinary bar plots and creating a `GroupBarPlot`

```
<?php
// Some data
$data1y = ...
$data2y = ...

// Create the bar plots
$b1plot = new BarPlot ($data1y);
$b1plot->SetFillColor ('orange');

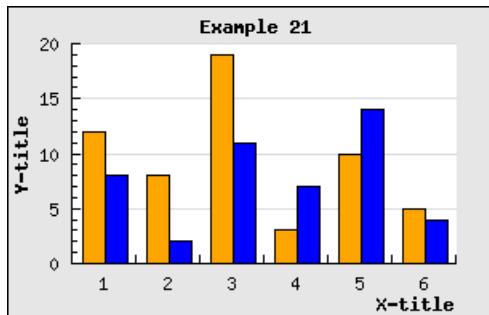
$b2plot = new BarPlot ($data2y);
$b2plot->SetFillColor ('blue');

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

// Add it to the graph
$graph->Add ($gbplot);
?>
```

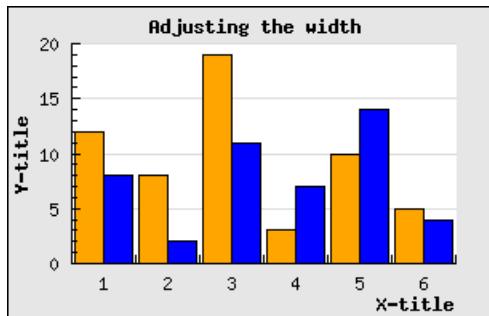
An example of this is shown in Figure 15.40, “A grouped bar plot (example21.php)”

**Figure 15.40. A grouped bar plot (example21.php) [example\_src/example21.html]**



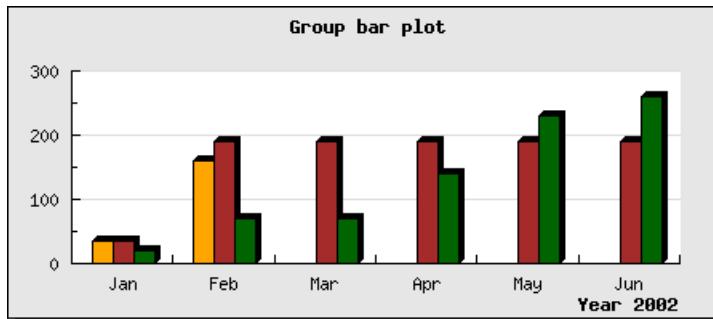
If the `SetWidth()` method is used on the `GroupBarPlot()` it will affect the total width used by all the added plots. Each individual bar width will be the same for all added bars. The default width for grouped bar is 70% of the width between the tick marks in the graph. In Figure 15.41, “Adjusting the width of a group bar plot (example22.php)” an example where the width is set to 90% is shown.

**Figure 15.41. Adjusting the width of a group bar plot (example22.php) [example\_src/example22.html]**



The number of data points in each data series must be the same. This means that if there are no available values they should be specified as 0. An example of this is shown in Figure 15.42, “All data series in a grouped bar graph must have the same number of data points (groupbarex1.php)”

**Figure 15.42. All data series in a grouped bar graph must have the same number of data points (groupbarex1.php) [example\_src/groupbarex1.html]**



### 15.2.3. Grouped accumulated bar graphs

It is perfectly possible to combine the previous bar types to have a grouped accumulated bar plot. This is done in a similar way by aggregating a number of accumulated plots in a group bar plot. The following code snippet shows how this can be done

```
<?php
// 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("brown");

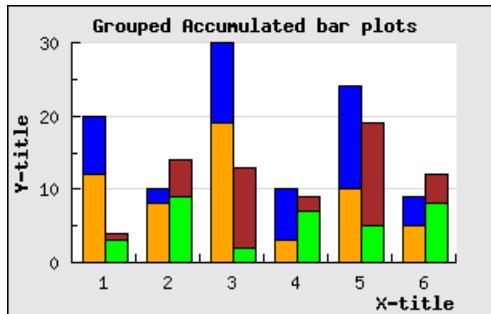
// Create the accumulated bar plots
$ab1plot = new AccBarPlot(array($b1plot , $b2plot));
$ab2plot = new AccBarPlot(array($b3plot , $b4plot));

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

// Add the combination to the graph
$graph->Add($gbplot);
?>
```

An example of this is shown in Figure 15.43, “A grouped accumulated bar graph (example24.php)” below

**Figure 15.43.** A grouped accumulated bar graph (`example24.php`)  
[`example_src/example24.html`]



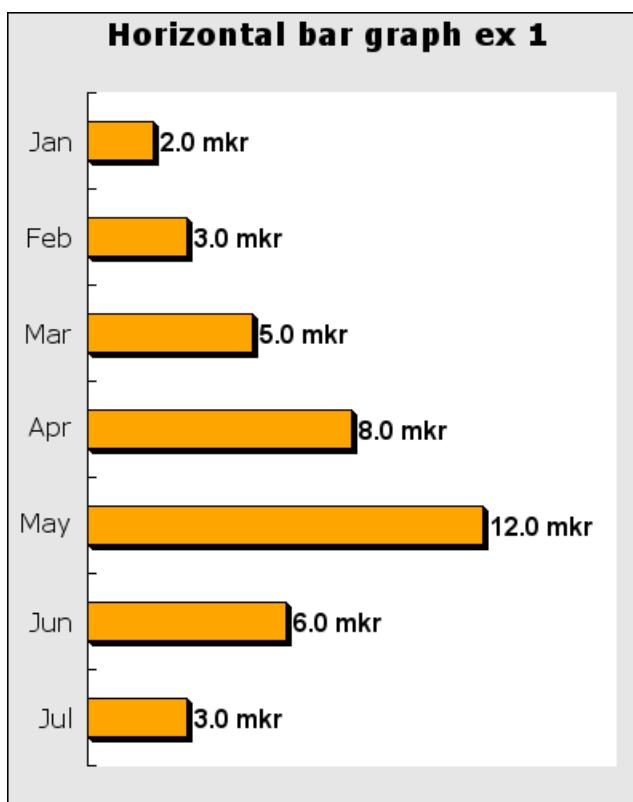
## 15.2.4. Horizontal bar graphs

If a large number of values needs to be displayed in a bar graph it is often better to rotate the bar graph 90 degree so that the bars are horizontal instead. There is no special graph type for this so this is achieved by rotating a standard vertical bar graph 90 degrees, usually with a call to

- `Graph::Set90AndMargin()`

The orientation of the labels of the axis will be automatically adjusted for bar graphs. A basic example is shown in Figure 15.44, “A basic horizontal bar graph (`horizbarex1.php`)” below

**Figure 15.44.** A basic horizontal bar graph (`horizbarex1.php`)  
[`example_src/horizbarex1.html`]

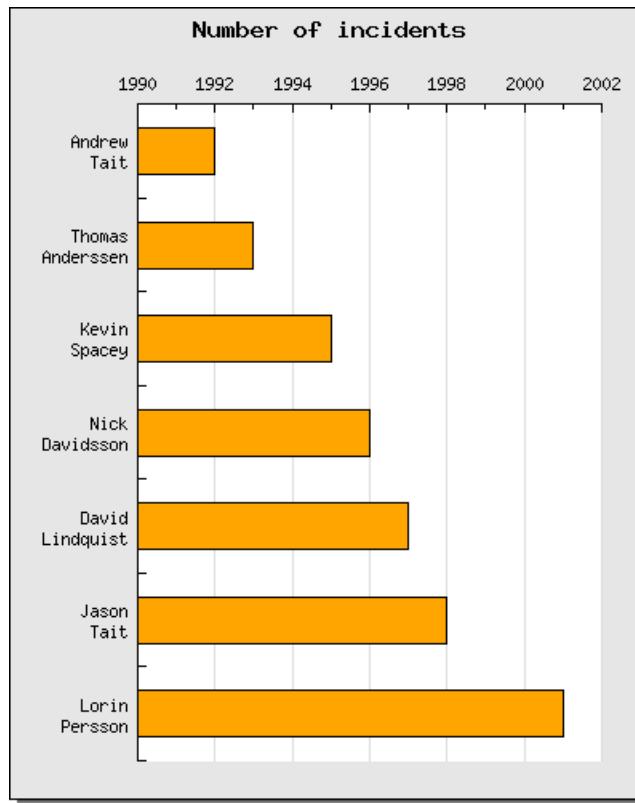


The example in Figure 15.45, “Using multiple line labels in a horizontal bar graph (`horizbarex4.php`)” shows how to use multiple lines as labels.

## Caution

Note that we have to use quotation marks ("") and not hyphens ('') in text string where we want to embed a newline character, i.e. "\n"

**Figure 15.45. Using multiple line labels in a horizontal bar graph (`horizbarex4.php`) [example\_src/horizbarex4.html]**



## 15.2.5. Adjusting the appearance of bar graphs

### Adjusting the width and colors

The width of each individual bar can be specified in either an absolute pixel size or as a fraction of the width between the major tick marks. The method used for this is

- `BarPlot::SetWidth($aWidth)`

If `$aWidth` is an integer  $> 1$  then it is interpreted as an absolute width in pixels. If it is a floating point number in the range  $[0,1]$  it will be interpreted as a fraction of the width between the tick marks.

### Using gradient fill

It is possible to use color gradient fill for the individual bars in the bar graph.

---

Different types of linear  
(cartesian) graph types

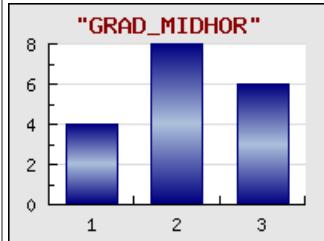
---

Color gradient fill fills a rectangle with a smooth transition between two colors. In what direction the transition goes (from left to right, down and up, from the middle and out etc) is determined by the style of the gradient fill. The library currently supports 8 different styles. All supported styles are displayed in Figure 15.46, “Supported gradient fills for bar plots” below.

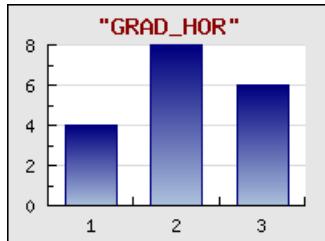
**Figure 15.47.** GRAD\_MIDVER  
([bargradsmallex1.php](#))  
[example\_src/  
[bargradsmallex1.html](#)]



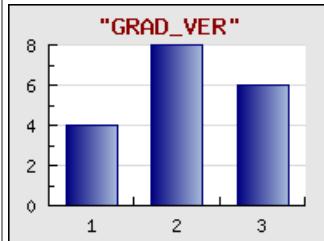
**Figure 15.48.** GRAD\_MIDHOR  
([bargradsmallex2.php](#))  
[example\_src/  
[bargradsmallex2.html](#)]



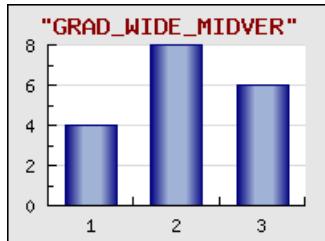
**Figure 15.49.** GRAD\_HOR  
([bargradsmallex3.php](#))  
[example\_src/  
[bargradsmallex3.html](#)]



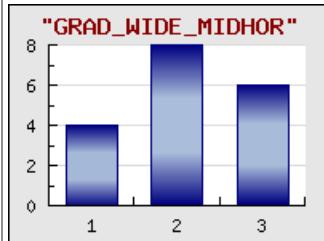
**Figure 15.50.** GRAD\_VER  
([bargradsmallex4.php](#))  
[example\_src/  
[bargradsmallex4.html](#)]



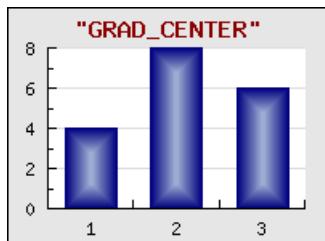
**Figure 15.51.** GRAD\_WIDE\_MIDVER  
([bargradsmallex5.php](#))  
[example\_src/  
[bargradsmallex5.html](#)]



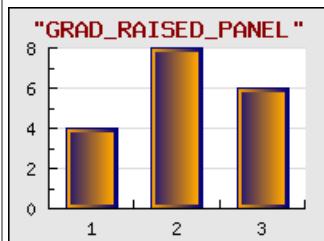
**Figure 15.52.** GRAD\_WIDE\_MIDHOR  
([bargradsmallex6.php](#))  
[example\_src/  
[bargradsmallex6.html](#)]



**Figure 15.53.** GRAD\_CENTER  
([bargradsmallex7.php](#))  
[example\_src/  
[bargradsmallex7.html](#)]



**Figure 15.54.** GRAD\_RAISED\_PANEL  
([bargradsmallex8.php](#))  
[example\_src/  
[bargradsmallex8.html](#)]



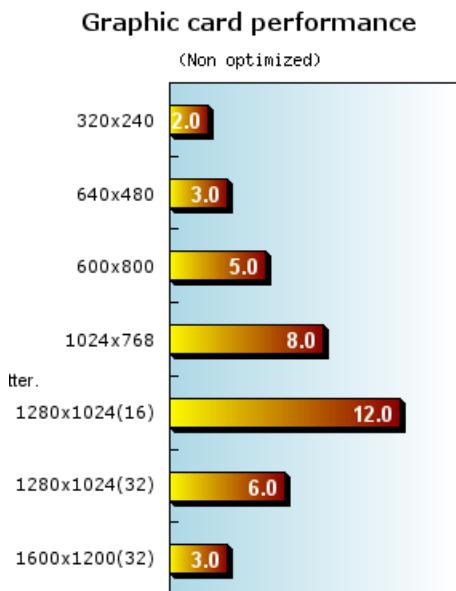
To specify a gradient fill for the bar plots you make use of the method `BarPlot::SetFillGradient()`. See the class reference for details of this function.

## Caution

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 a few times.

As a final example we show an horizontal bar graph with gradient fill on both the background and the bars in Figure 15.55, “Horizontal bar graph with gradient fill (`horizbarex6.php`) [`example_src/horizbarex6.html`]”

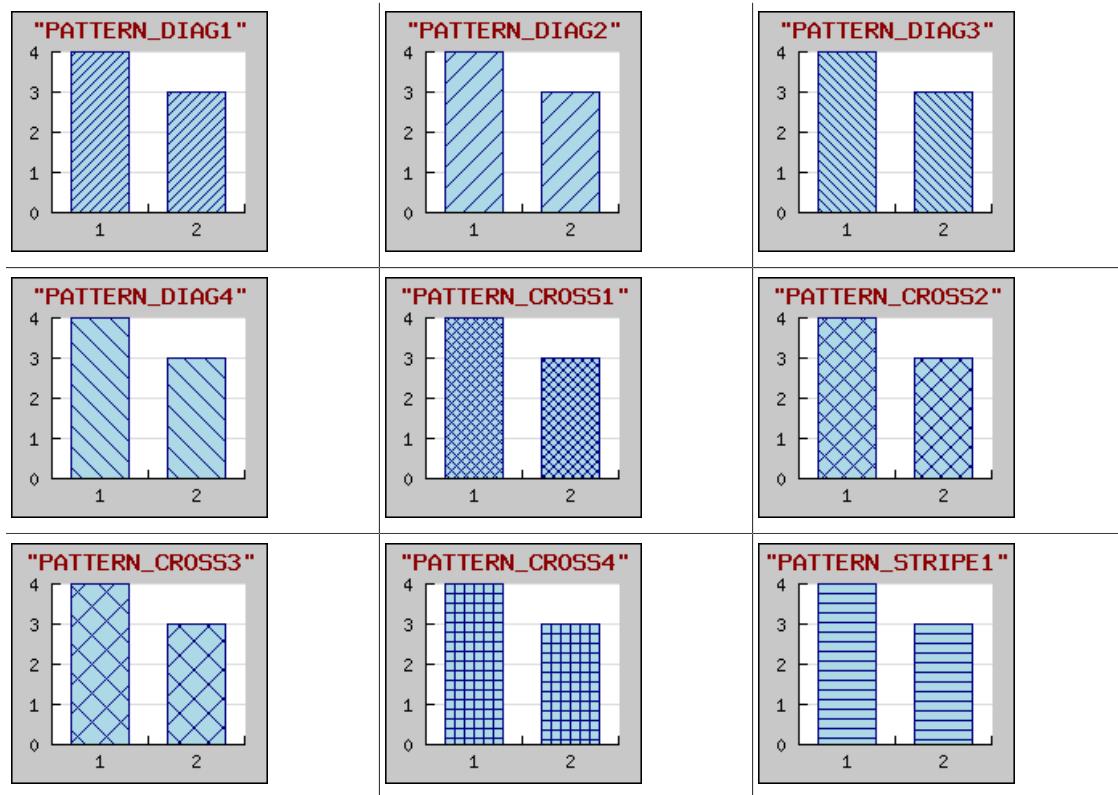
**Figure 15.55. Horizontal bar graph with gradient fill (`horizbarex6.php`) [`example_src/horizbarex6.html`]**



## Using pattern fills

As an alternative to solid and gradient fill the bars can also have a number of patterns. This is useful for the case where black and white copies needs to be printed. The library supports the following nine different patterns as shown in Figure 15.56, “Supported pattern fills for bar plots”

**Figure 15.56. Supported pattern fills for bar plots**



To specify a pattern to be used the following method is used

- `BarPlot::SetPattern($aPattern, $aColor='black')`

`$aPattern` is one of the symbolic names as shown in Figure 15.56, “Supported pattern fills for bar plots” If `$aPattern` is an array then each specified pattern will be used successively for each individual bar. If there are more bars then pattern specified then pattern used will be wrapped around.

## Displaying and formatting the value of the bar

In the same way as values could be displayed on line plots they can also be displayed on bar graphs. The formatting options for bar graphs, apart from the basic font, color and angle, also allows the specification on where on the bar the values should be displayed. This can be

- at the bottom of the bar, "bottom"
- at the middle of the bar, "middle"
- at the maximum value (but still inside the bar). "max"
- at the top of the bar (outside the bar), "top"

The position is adjusted with a call to

- `BarPlot::SetValuePos($aPos)`

using one of the above strings as value.

The value of the bar is enabled and controlled by accessing the "value" property of the bar plot. The following line will enable the value (this is done in exactly the same way as for line plots)

```
$barplot->value->Show();
```

By default the value is displayed at the top of the bar. In the same way as for line plot it is possible to adjust the formatting of the data labels by both using a format callback function as well as statically adjusting the format, for example the angle of the label.

```
// Callback function
function separator1000_usd($aVal) {
 return '$'.number_format($aVal);
}

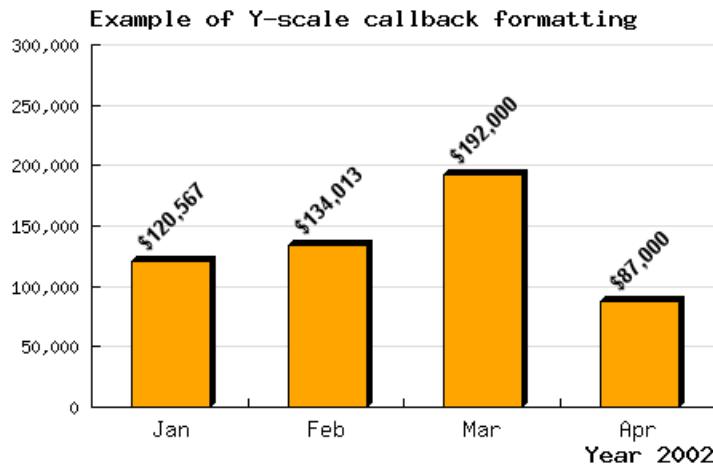
// Must use TTF fonts if we want text at an arbitrary angle
$bplot->value->SetFont(FF_ARIAL,FS_BOLD);
$bplot->value->SetAngle(45);
$bplot->value->SetFormatCallback('separator1000_usd');

// Black color for positive values and darkred for negative values
$bplot->value->SetColor('black','darkred');
$graph->Add($bplot);
```

There is one thing to take notice of here. The color of the label can be different depending on whether the bar has a positive or a negative value.

An example of using these formatting options for a bar graph is shown in Figure 15.57, “Using a callback to format the labels on a bar (barscalecallbackex1.php)”

**Figure 15.57. Using a callback to format the labels on a bar (barscalecallbackex1.php) [example\_src/barscalecallbackex1.html]**



## Adding a drop shadow to the bars

Each bar can also have a drop shadow. This is enabled by calling

- BarPlot::SetShadow(\$aColor="black",\$aHSize=3,\$aVSize=3,\$aShow=true)

\$aColor, The color of the drop shadow

\$aHSize, Horizontal size of the shadow

\$aVSize, Vertical size of the shadow

\$aShow, true = enable the shadow

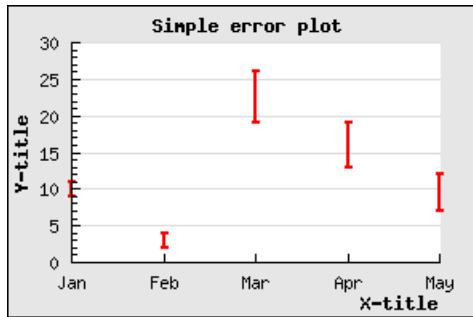
## 15.3. Error plot graphs

Error plots are used to visually indicate uncertainty in data points. This is done by specifying both a min and max value for each data point.

Before error plots can be used the module "jpgraph\_error.php" must be included.

The following example illustrates a simple error bar. We will have 5 points, so we need 10 Y-values. We also would like the error bars to be red and 2 pixels wide. All this is accomplished by creating an instance of the `ErrorPlot` class in much the same way as, for example, a normal line plot.

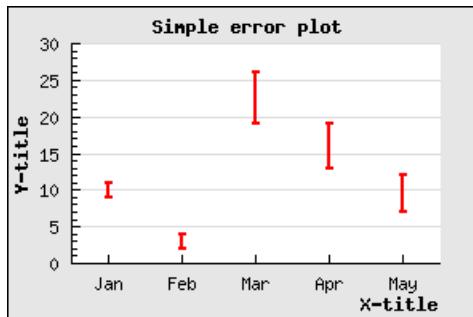
**Figure 15.58. A basic error plot (`example13.php`) [example\_src/example13.html]**



There is one displeasing esthetic 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 last bar just at the edge of the plot area.

To adjust this we can use the method `ErrorPlot::SetCenter()` which will adjust the x-scale so it does not use the full width of the X-axis.

**Figure 15.59. Making use of SetCenter() with error plots (`example14.php`) [example\_src/example14.html]**



### 15.3.1. Line error plots

A variant of the error plot graph is to use an `LineErrorPlot` instead. This is almost the same as the `ErrorPlot` but with the added feature that each data point also has an middle value which a line is drawn through. This can be thought of as a line plot combined with an error plot.

Since this also uses a line the module "`jpgraph_line.php`" must be included in addition to the error module.

To control the various properties of the line drawn the "line" property of the error line plot may be accessed. So, for example, to set the line to have weight of 2 pixels wide and with a blue color the following two lines are needed

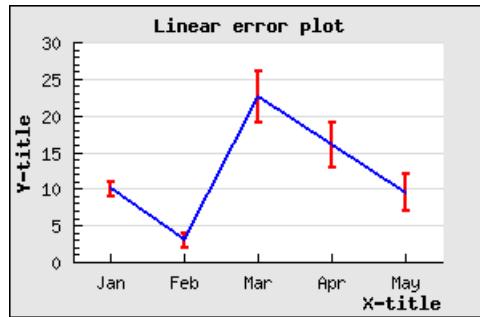
```
<?php
$elplot->line->SetWeight (2);
$elplot->line->SetColor ('blue');
?>
```

An example of this is shown in Figure 15.60, “A basic Line error plot (example15.php)”. We could now also add a legend to none, one or both of the line types(the plain line and/or the error bar). So for example if we wanted the legend "Min/Max" for the red error bars and a legend "Average" for the blue line the following lines should be added

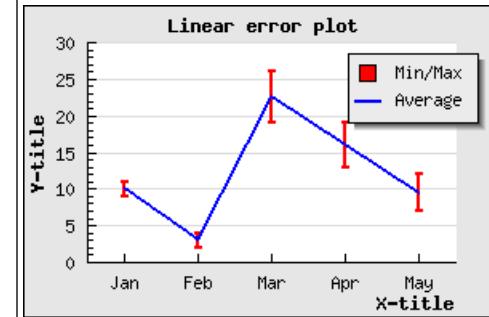
```
<?php
$errplot->SetLegend ('Min/Max');
$errplot->line->SetLegend ('Average');
?>
```

The result is shown in Figure 15.61, “A line error plot with a legend (example16.php) ”

**Figure 15.60. A basic Line error plot (example15.php)**  
[[example\\_src/example15.html](#)]



**Figure 15.61. A line error plot with a legend (example16.php)**  
[[example\\_src/example16.html](#)]



## 15.4. Stock graphs

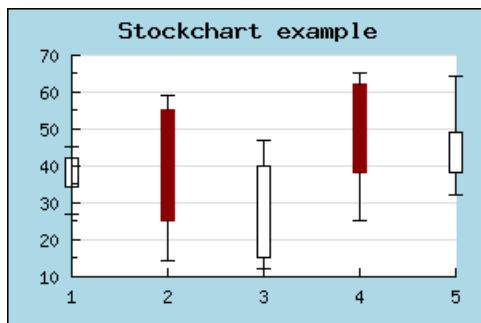
Stock charts are used to display data values where one is interested in 4 different values for each data point. This could for example be used to display a stocks open,close, min and max value during a specific day. Hence the name Stock chart (or Stock plot).

Stock plots are created as an instance of class `StockPlot` so the module "`jpgraph_stock.php`" must first be included in the script.

Figure 15.62, “A stock graph (stockex1.php) ”illustrates a sample Stock chart plot

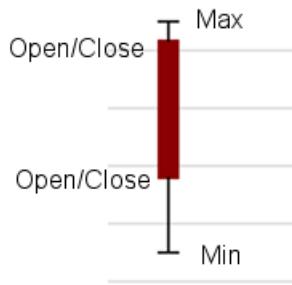
**Figure 15.62. A stock graph (`stockex1.php`)**

[`example_src/stockex1.html`]



For this type of plot the y-data array must be consist of a number of quadruples of data where each quadruple consists of (open,close,min,max). The open and close values determine the min max for the middle bar and the min,max determine the end points of the "error-lines" at the top and bottom of each bar as shown in ??.

**Figure 15.63. Explaining stock graphs**



For this type of plot the y-data array must be consist of a number of quadruples of data where each quadruple consists of an even number of (open,close,min,max).

The open and close values determine the min max for the middle bar and the min,max determine the end points of the "error-lines" at the top and bottom of each bar as shown in Figure 15.63, "Explaining stock graphs".

Note that the data follows the rules

- $\text{min} < \text{max}$
- $\text{min} < \text{min( open, close )}$
- $\text{max} > \text{max( open, close )}$

To separate the two cases where "open > close" or "open < close" two different colors are used.

These colors are specified with the method

- `StockPlot::SetColor($aFrame, $aFill='white', $aFrameNeg='darkred', $aFillNeg='darkred')`

`$aFrame, $aFill`, The frame and fill color when  $\text{open} \leq \text{close}$

`$aFrameNeg, $aFillNeg`, The frame and fill color when  $\text{open} > \text{close}$

`StockPlot::SetColor()` method. By default a positive bar ( $\text{close} > \text{open}$ ) have a fill color of white and for the negative case where ( $\text{close} < \text{open}$ ) the bars have a red color.

The final possible variation of stock plots is the option to determine whether or not the end point for the min,max lines should have the horizontal line marking the end of the line or not. This can be adjusted with a call to method

- `StockPlot::HideEndLines($aHide=true)`

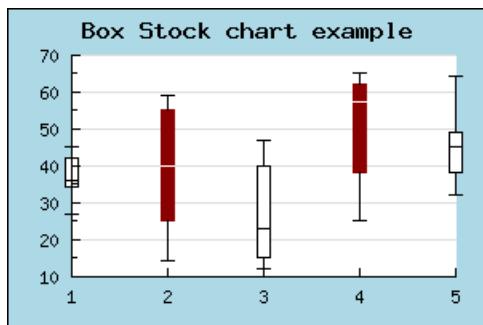
### 15.4.1. Stock plot variant: Boxplot

A minor variation of stock charts is the BoxPlot plot type. this is almost the same as StockPlot but with the very small difference that in addition to the open,close,min, max values it is also possible to specify a fifth, median value .

A box plot is created as in instance of class BoxPlot which is defined in "jpgraph\_stock.php"

The median lies between the open and close value and is illustrated as a horizontal line within the bar. An example of this is shown in

**Figure 15.64.** A typical boxplot (`boxstockex1.php`) [`example_src/boxstockex1.html`]



The color of the median line can be modified with a call to

- `BoxPlot::SetMedianColor($aColor)`

### 15.4.2. Image maps for Box and Stock charts

In the same way as for other plots it is possible to associate an image map with these plots. The "hot" area for each plot is the mid "bar" section. In the same way as other plot types the method

- `BoxPlot::SetCSIMTargets($aTargets,$aAlts='', $aWinTargets='')`

is used to set the URL:s for the hot areas

## 15.5. Scatter graphs

Scatter plots are used to display a number of data points given by there x- and y-coordinate. The difference from a line plot is that the x-coordinate must always be specified. Each point is stroked on the image with a mark as with line plots. The stroked marks can also be connected with an optional line.

To use a scatter plot the module "jpgraph\_scatter.php" must first be included in the script.

### Note

Carefully reviewing the constructor for line plots shows that it also can accept both a x- and y-array with coordinates. This means that a line plot can also be used to create a scatter plot. However scatter plots have some formatting options not available for line plots.

Even though the normal use of scatter plots is to supply both x- and y-coordinates it is still perfectly possible to use a text- or int-scale for the x-coordinates to just enumerate the points (the points will be placed along an imaginary integer scale). This is especially useful when using the "Impulse" type of scatter plot as shown below.

If no x-coordinates are specified each value will be placed at consecutive x-coordinates [1, 2, 3, ...]

A scatter plot is constructed by creating an instance of the class `ScatterPlot` supplied with the proper arguments, i.e.

```
$scatterplot = new ScatterPlot($ydata, $xdata);
```

The following example shows a very basic scatter plot

**Example 15.3. A basic scatter plot (`scatterex1.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_scatter.php");

$datax = array(3.5,3.7,3,4,6.2,6,3.5,8,14,8,11.1,13.7);
$datay = array(20,22,12,13,17,20,16,19,30,31,40,43);

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

$graph->img->SetMargin(40,40,40,40);
$graph->SetShadow();

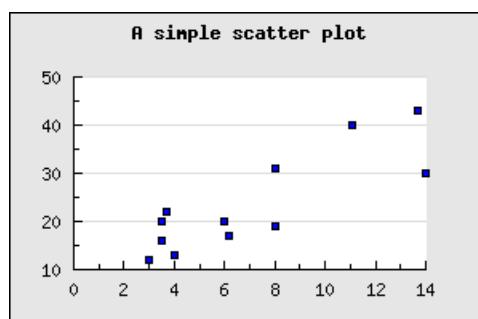
$graph->title->Set("A simple scatter plot");
$graph->title->SetFont(FF_FONT1,FS_BOLD);

$sp1 = new ScatterPlot($datay,$datax);

$graph->Add($sp1);
$graph->Stroke();

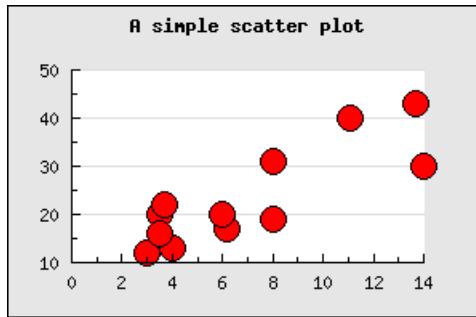
?>
```

**Figure 15.65. A basic scatter plot (`scatterex1.php`)** [example\_src/  
`scatterex1.html`]



In the same way as we adjusted the look and feel for markers for line plot (see ??) we can do the same here as shown in Figure 15.66, “Adjusting the size and color of the marker (`scatterex2.php`)”

**Figure 15.66. Adjusting the size and color of the marker (scatterex2.php)**  
[example\_src/scatterex2.html]



The marks are accessed through the instance variable `ScatterPlot::mark` (in complete analogy with line plots)

### 15.5.1. Combining marks with a line

It is possible to combine the individual data points with a line - a link. The properties for this link can be accessed by using the instance variable `ScatterPlot::link`. In order to enable the link the method `ScatterPlot::link::Show()` must first be called.

#### Note

For historical reasons there is also a utility method `ScatterPlot::SetLinkPoints()` where links can be enabled and adjusted. However, this method will be removed in future versions of the library.

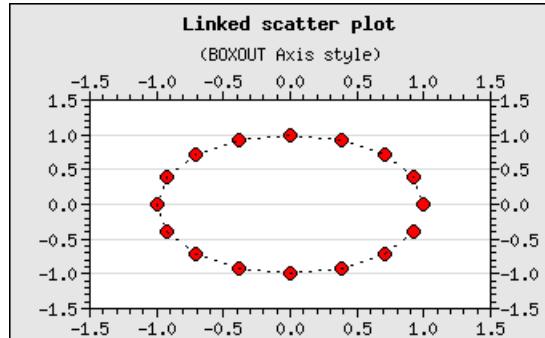
The properties that can be adjusted are the normal line properties, i.e. weight, color and line style. The following code snippet shows how to add a link line with a dotted style drawn in black to scatter plot.

```
<?php
// Enable the link lines
$scatterplot->link->Show();

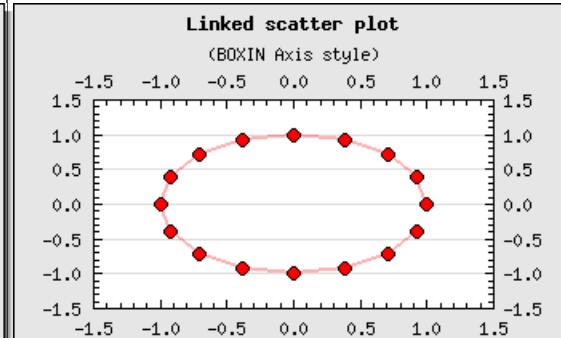
// Set the properties
$scatterplot->link->SetWeigth(1);
$scatterplot->link->SetColor('black');
$scatterplot->link->SetStyle('dotted');
?>
```

In Figure 15.67, “Combining data points with a dotted line (scatterlinkex3.php)” and Figure 15.68, “Combining data points with a red line (scatterlinkex4.php)” there are two variants of adding a link line to a scatter plot. For those figures we have also used two of the predefined scientific axis styles (see Section 14.1.4, “Predefined scientific axis setups”).

**Figure 15.67. Combining points with a dotted data line**  
(`scatterlinkex3.php`)  
[`example_src/scatterlinkex3.html`]



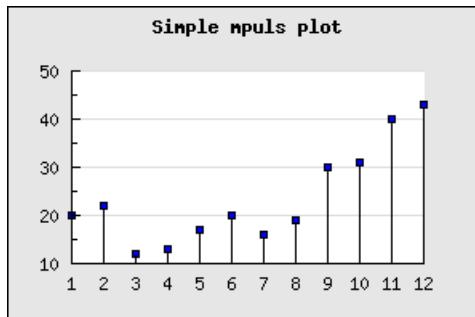
**Figure 15.68. Combining points with a red data line**  
(`scatterlinkex4.php`)  
[`example_src/scatterlinkex4.html`]



## 15.5.2. Creating impulse (or stem) - plots

A variant of scatter plots often used in electrical engineering is the stem plot as shown in Figure 15.56, “Supported pattern fills for bar plots”

**Figure 15.69. Stem plot (`impulsex1.php`)** [`example_src/impulsex1.html`]



This variant of the scatter plot is created by calling the method

```
$scatterplot->SetStem();
```

### Tip

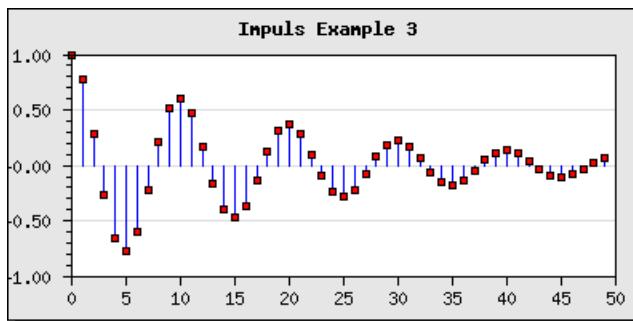
It is possible to create a stem graphs without any mark by specifying the mark type as (-1). That way only the impulse lines will be drawn.

In order to adjust the look and feel of the stems the following methods can be used

- `ScatterPlot::SetWeight($aWeight)`
- `ScatterPlot::SetColor($aColor)`

Another example of stem plot is shown in Figure 15.70, “Adjusting the overall look and feel for the stem graph (impulsex3.php)” where we have adjusted both the look and feel of the stem plot as well as the position of the x-axis.

**Figure 15.70. Adjusting the overall look and feel for the stem graph (impulsex3.php) [example\_src/impulsex3.html]**



There is also another complex impulse example shown in Figure 14.58, “Adding a static line at  $y=0$  to simulate an extra 0-axis (impulsex4.php)” where we have used a `PlotLine` to create a “virtual” x-axis.

### 15.5.3. Field plots

A variant of scatter plot is the so called Field Plots.

This is basically a scatter plot where each scatter point is an arrow with a direction between 0 to 359 degrees. This effectively allows the visualization of 3 parameters at each point ( $x, y$  and angle).

A field plot is created as an instance of `class FieldPlot` with three argument

- an array of y-coordinates
- an array of x-coordinates
- an array of angles

as the following code snippet shows

```
$fieldplot = new FieldPlot($datay, $datax, $angle);
```

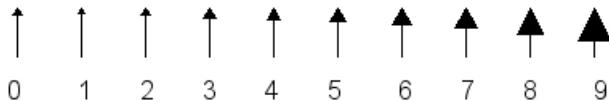
The size and color of each arrow in the field plot is controlled by accessing the property `FieldPlot::arrow` and using the two methods

- `FieldPlot::arrow->SetSize($aSize, $aArrowSize=2)`

`$aSize`, Specifies the length (in pixels) of the arrow

`$aArrowSize`, The arrow size is specified as an integer in the range [0,9]. The possible arrow sizes are shown in Figure 15.71, “Possible sizes of arrow heads for field plots”.

**Figure 15.71. Possible sizes of arrow heads for field plots**



- `FieldPlot::arrow->SetColor($aColor)`

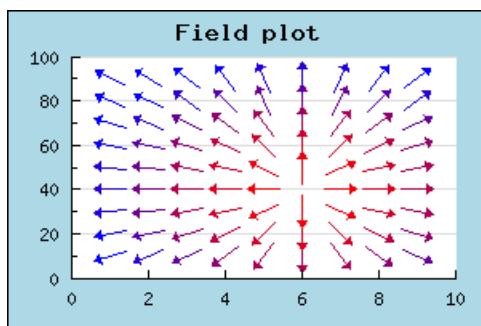
The color of the arrow

Another way to individually format each arrow is to use a callback method. The callback method must have three arguments x-, y-coordinates and angle. The callback method must return an array with three elements (color,size,arrow size). The callback method is specified by calling

- `FieldPlot::SetCallback ($aFunc)`

Figure 15.72, “A field plot (`fieldscatterex1.php`)” shows a field plot with a callback function.

**Figure 15.72. A field plot (`fieldscatterex1.php`) [example\_src/  
`fieldscatterex1.html`]**



## 15.5.4. Balloon plots

This is a variant of scatter plot where each data point is a filled circle and the size of the circle is dynamically adjusted. This makes it possible to display three values at each data point (x,y,"size"). There is no need for a specific plot type for this since these types of plots can be constructed with an ordinary scatter plot and a mark formatting callback.

In order to specify a callback for the marks the following method is used

- `ScatterPlot::mark::SetCallbackYX($aFunction)`

The argument for the callback function is y- and x-value for the data point. The return value should specify the format of the marker as an array of (*width*, *border\_color*, *fill\_color*, *filename*, *image scale*). All values must be included.

We can now create a balloon plot as follows.

1. Create a data array that specifies the data including both x- and y-coordinate, color and size of the filled circle. For example as:

```
$data = array(
 array(1,12,10,'orange'),
 array(3,41,15,'red'),
 array(4,5,19,'lightblue'),
 array(5,70,22,'yellow')
);
```

2. We must now convert this to some suitable data for a scatter plot. The only thing to keep in mind is that we need to share the data with the callback function and for that purpose we store the formatting data in an external array indexed by the x,y-value.

```
$n = count($data);
for($i=0; $i < $n; ++$i) {

 $datax[$i] = $data[$i][0];
 $datay[$i] = $data[$i][1];

 $format[strval($datax[$i])][strval($datay[$i])] = array($data[$i][2], $data[$i][3]);
}
```

The callback function can now be specified as

```
function FCallback($aYVal,$aXVal) {
 // We need to access the global format array
 global $format;
 return array($format[strval($aXVal)][strval($aYVal)][0],',
 $format[strval($aXVal)][strval($aYVal)][1],',
 $format[strval($aXVal)][strval($aYVal)][2],');
}
```

The callback function will now return the format value (size and color) we specified originally depending on the x,y - coordinate. The callback function can be installed with a call to

```
$sp1->mark->SetCallbackYX('FCallback');
```

3. The final step is to create an ordinary scatter plot with a marker that is specified as a filled circle

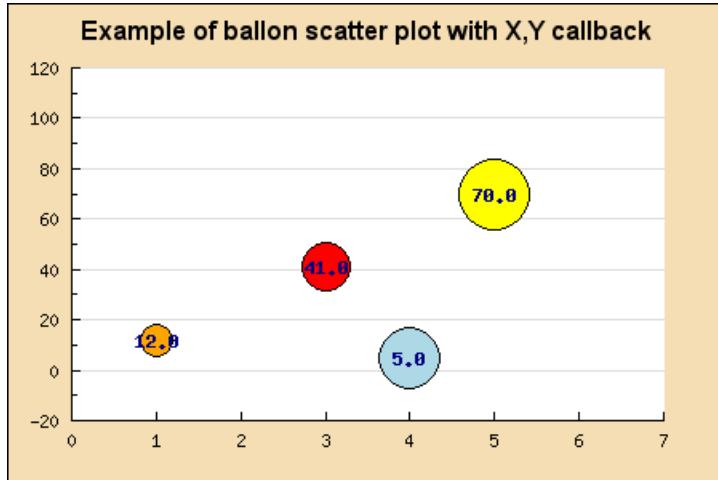
```
$sp1 = new ScatterPlot($datay,$datax);
$sp1->mark->SetType(MARK_FILLED CIRCLE);
```

As an optional addition we can also enable the display of the values at each data point by calling

```
$sp1->value->Show();
$sp1->value->SetFont(FF_FONT1,FS_BOLD);
```

Putting all these together gives the result shown in Figure 15.73, “Using format callback to create a balloon plot (balloonex2.php)”

**Figure 15.73. Using format callback to create a balloon plot (`balloonex2.php`)**  
[`example_src/balloonex2.html`]



## 15.5.5. Creating Geo-maps

Geo-maps (a.k.a. Geo-charts, push-pin graphs) is used to illustrate locations around the world by putting markers on a flat projection of the earth. A Geo-map is done by adding a Worldmap background to a standard scatter graph. The library includes a suitable background in the "Examples/" directory "wordmap1.jpg". This can then be done as the following example shows

```

require_once ("jpgraph/jpgraph_scatter.php");

DEFINE('WORLDMAP','worldmap1.jpg');

function markCallback($y,$x) {
 // Return array width
 // width,color,fill color, marker filename, imgscale
 // any value can be false, in that case the default value will
 // be used.
 // We only make one pushpin another color
 if($x == 54)
 return array(false,false,false,'red',0.8);
 else
 return array(false,false,false,'green',0.8);
}

// Data arrays
$datax = array(10,20,30,40,54,60,70,80);
$datay = array(12,23,65,18,84,28,86,44);

// Setup the graph
$graph = new Graph(400,270);

// We add a small 1pixel left,right,bottom margin so the plot area
// doesn't cover the frame around the graph.
$graph->img->SetMargin(1,1,1,1);
$graph->SetScale('linlin',0,100,0,100);

// We don't want any axis to be shown
$graph->xaxis->Hide();
$graph->yaxis->Hide();

// Use a worldmap as the background and let it fill the plot area
$graph->SetBackgroundImage(WORLDMAP,BGIMG_FILLPLOT);

// Setup a nice title with a striped bevel background
$graph->title->Set("Pushpin graph");
$graph->title->SetFont(FF_ARIAL,FS_BOLD,16);
$graph->title->SetColor('white');
$graph->SetTitleBackground('darkgreen',TITLEBKG_STYLE1,TITLEBKG_FRAME_BEVEL);
$graph->SetTitleBackgroundFillStyle(TITLEBKG_FILLSTYLE_HSTRIPED,'blue','darkgreen');

// Finally create the scatterplot
$sp = new ScatterPlot($datay,$datax);

// We want the markers to be an image
$sp->mark->SetType(MARK_IMG_PUSHPIN,'blue',0.6);

// Install the Y-X callback for the markers
$sp->mark->SetCallbackYX('markCallback');

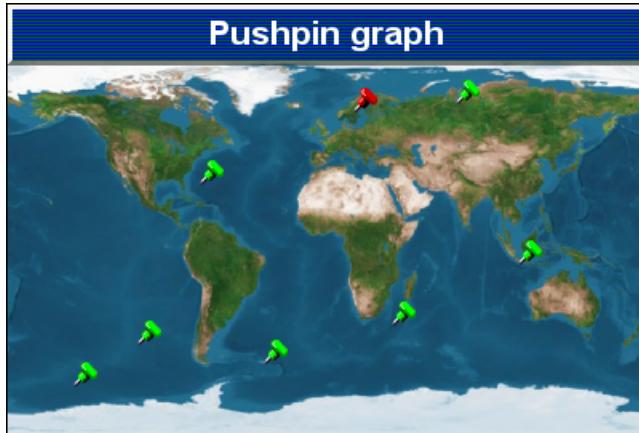
// ... and add it to the graph
$graph->Add($sp);

// .. and output to browser
$graph->Stroke();

?>

```

**Figure 15.74.** An example with geo maps (`pushpinex1.php`) [`example_src/pushpinex1.html`]



Another example of using worldmaps with Pie graphs is shown in Section 16.1.8, “Adding background images to Pie graphs”

### Note

The library does not include any conversion utility to/from Longitude/Latitude to UTM coordinates that could be used to automatically position data marks on the Mercator projection map. The options to stretch and scale the worldmap would make it rather cumbersome to create an accurate conversion. For a good overview on this kind of translation see [Converting Latitude/Longitude to Universal Transverse Mercator \(UTM\)](http://www.gpsy.com/gpsinfo/geotoutm/) [<http://www.gpsy.com/gpsinfo/geotoutm/>]

## 15.6. Contour graphs

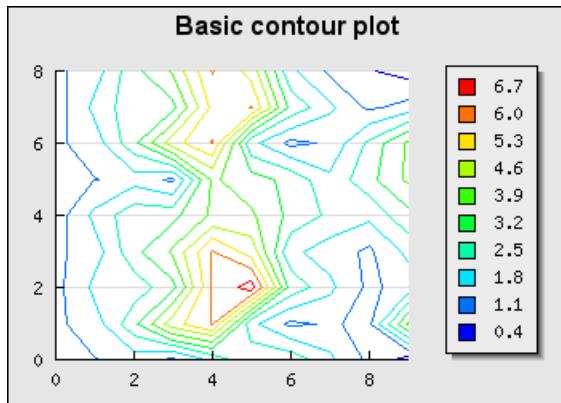
In order to create Radar graphs the module "`jpgraph_contour.php`" must first be included.

### Note

The Pro-version (>= v3.1) also includes a module "`jpgraph_contourf.php`" which is also supports filled contour plots as well as labelling in the plots of the isobar lines. The pro-version also uses a more advanced adaptive-mesh algorithm that give better resolution without manual interpolation of the original data. Both triangular and rectangular interpolation meshes are supported. The filled contour is described in Chapter 23, *Filled contour graphs*

Contour plots are used to plot the isobar lines (height curves) of a three dimensional (3D) graph. The contour plot itself is only a two-dimensional plot and looks like an ordinary topographic map. An example of a contour graph is shown in Figure 15.75, “A basic contour graph (`basic_contourex01.php`)”

**Figure 15.75. A basic contour graph (`basic_contourex01.php`)  
[[example\\_src/basic\\_contourex01.html](#)]**



A contour graph is both similar and different from other types x-y graphs.

- A contour graph uses the standard x-y class Graph as basic graph canvas
- A contour graph has all the standard graph formatting options
- The legend is automatically generated
- The input data is a matrix and not an array

There are primarily four main parameters that can be used to control the apparence of the contour plots

1. **the number of isobar lines**, by default 10 isobar lines are used
2. **the level of data grid interpolating**, i.e. to give the appearance of smoother isobar lines it is possible to tell the library to create a number of intermediate points between the given points in the input data matrix. It is important to note that this does not create any further real information it only creates smoother lines under the assumption that it is valid to assume linear interpolation between the original data points.

The level of interpolation is specified as an integer in the practical range 1 to 5. A level of 1 corresponds to just keeping the original data, a level of 2 corresponds to subdividing the original data points in two, i.e. one extra interpolated points is created in both x- and y-direction and so on. A level of 3 corresponds to further sub-divding the level 2 matrix one more time.

Even though it is theoretically possible to use an arbitrary interpolation factor the library does not allow an interpolation factor larger than 5. The reason is purely computational since the total number of data points increases very quickly.

For example if the original grid has size 10x10 (=100 data points) interpolating this grid with a factor of 5 will generate a new matrix of size 145x145 (=21025 data points).

3. **flip around the Y-axis**. By default the first row in the input matrix corresponds to y=0, however this also means that the plotted contour will be flipped compared with the input matrix since y=0 is at the bottom of the graph. If for visual appearance reason one wants the input data matrix to have the same orientation as the resulting graph it is possible to have the library interpret the last row in the input data matrix as y=0 instead
4. **the color of the isobar lines**. By default they will be assigned from the natural color spectra with pure dark blue corresponding to the lowest point in the plot and pure red corresponding to the highest isobar.

The remaining possibilities to adjust the appearance of the contour plot corresponds to the standard ways of changing the layout of the graph, for example adding titles, adjusting colors and changing the type of axis to be displayed.

The scale of the contour plot is by default the natural scale, i.e. the points are assumed to be numbered (0..n) where n is the number of points in the corresponding direction and also corresponds to each entry in the input data matrix.

## 15.6.1. Input data for contour graphs

The input data to a contour graph is a matrix. The value at each entry in the matrix represents the heights at the specified (row,col) in the matrix. In all of our example we will use the following data matrix

```
<?php
$data = array(
 array (0.5,1.1,1.5,1,2.0,3,3,2,1,0.1),
 array (1.0,1.5,3.0,5,6.0,2,1,1.2,1,4),
 array (0.9,2.0,2.1,3,6.0,7,3,2,1,1.4),
 array (1.0,1.5,3.0,4,6.0,5,2,1.5,1,2),
 array (0.8,2.0,3.0,3,4.0,4,3,2.4,2,3),
 array (0.6,1.1,1.5,1,4.0,3.5,3,2,3,4),
 array (1.0,1.5,3.0,5,6.0,2,1,1.2,2.7,4),
 array (0.8,2.0,3.0,3,5.5,6,3,2,1,1.4),
 array (1.0,1.5,3.0,4,6.0,5,2,1,0.5,0.2));
?>
```

By default the entry (0,0), i.e. row=0, col=0 will be positioned at graph position (0,0), i.e. the lower left corner. Since this will represent an inverted image compare to the data matrix. In order to have the graph oriented in the same way as the data matrix, i.e. entry (0,0) positioned in the top left corner the method

- `ContourPlot::SetInvert($aFlg=true)`

In order to get a smooth contour plot it is necessary to have adequate number of data points. What consists "adequate" number depends on the overall graph size. The larger graph the more points are needed. To help create smooth contour plot the library offers automatic linear interpolation to specified depth. It is important to note that this does not increase the accuracy. It merely creates artificial point to draw smoother curves. This technique is described in detail in Section 15.6.4, “Understanding mesh interpolation”

### Tip

If the data for the matrix is available in a file a convinient way to get hold of the dat in the file is to use the utility class `ReadFileData` to get hold of the data using the method `ReadFileData::FromMatrix($aFile,$aSeparator=' ')` which read the matrix from a file. Each row of the matrix must be a separate line and each cell is separated with the character specified as the second argument. By default a space is used as separator. All values read back are converted to floating point numbers (double precision). The following short example shows how easy this is to use

```
$data = ReadFileData::FromMatrix('matrixdata.txt');
```

## 15.6.2. Creating a contour graph

Figure 15.75, “A basic contour graph (basic\_contourex01.php)” shows the most basic contour plot. This is crated by first instantiating a graph and then creating an instance of class `ContourPlot` which is added to the graph.

The constructor for class `ContourPlot` has the following signature

- `ContourPlot::__construct($aDataMatrix, $aIsobar=10, $aFactor=1, $aInvert=false, $aIsobarColors=array())`

`$aDataMatrix`, The input data

`$aIsobar`, The number of isobars if the argument is an integer or an array specifying the exact locations of each isobar if the argument is an array

`$aFactor`, Grid interpolation factor

`$aInvert`, Invert the data matrix so the matrix entry (0,0) corresponds to the top left corner

`$aIsobarColors`, Optional specification of isobar colors

This means that the common structure to create a basic contour plot will be

```
<?php
$data = array (array (...));

// Basic contour graph
$graph = new Graph($width,$height);
$graph->SetScale('intint');

// Adjust the margins to fit the margin
$graph->SetMargin(30,100,40,30);

// Setup
$graph->title->Set('Basic contour plot');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,12);

// A simple contour plot with default arguments (e.g. 10 isobar lines)
$cp = new ContourPlot($data);

// Display the legend
$cp->ShowLegend();

$graph->Add($cp);
?>
```

Since the most common format for contour graphs is to show axis on all sides we should add the line

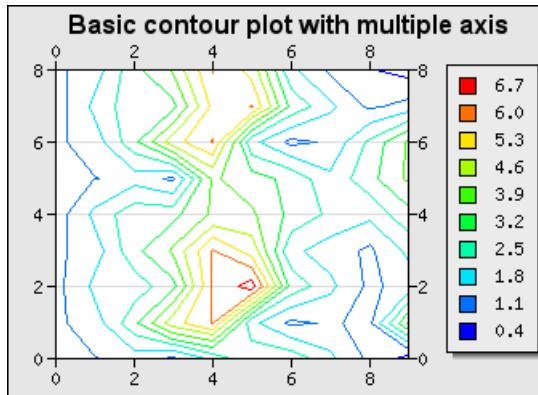
```
$graph->SetAxisStyle(AXSTYLE_BOXOUT);
```

to our previous example so we get the effect shown in Figure 15.76, “Adding axis on all sides (`basic_contourex02.php`)”. To show the effect of flipping the data around the center line we have to add the following line

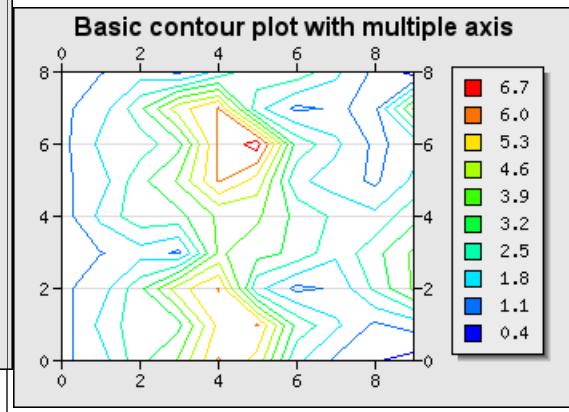
```
$cp->SetInvert();
```

and get the result of adding this line is shown in Figure 15.77, “Flipping the data around the center line (`basic_contourex05.php`)”

**Figure 15.76. Adding axis on all sides (basic\_contourex02.php)**  
[example\_src/  
**basic\_contourex02.html**]



**Figure 15.77. Flipping the data around the center line (basic\_contourex05.php)**  
[example\_src/  
**basic\_contourex05.html**]



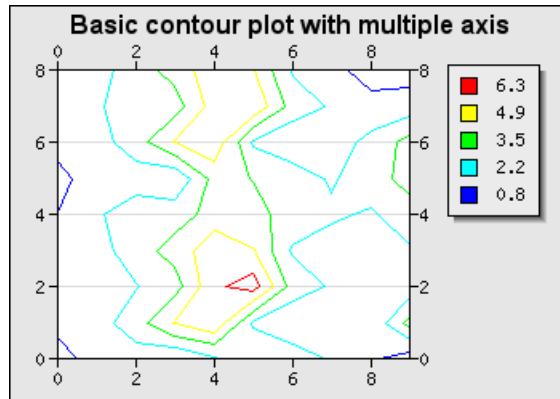
### 15.6.3. Adjusting the color and number of isobar lines

As mentioned above the library uses 10 isobar lines by default. The number of isobar lines can be adjusted by specifying a second integer argument to the constructor of the contour plot. For example to use only 5 isobar lines the construction of the contour plot would be changed to

```
$contourplot = new ContourPlot($data, 5);
```

The result of changing this in the previous example is shown in Figure 15.78, “Using only 5 isobar lines (basic\_contourex04.php)”

**Figure 15.78. Using only 5 isobar lines (basic\_contourex04.php)**  
[example\_src/basic\_contourex04.html]



The values for the isobars are determined by finding the lowest and highest point in the input data matrix and then spreading the isobars evenly between these extreme points.

In some applications however it is necessary to have better control over where exactly the isobars are placed and for this reason it is possible to tell the library exactly at what values the isobars should be.

This is done by providing an array with the values of the isobars instead of a single number in the creation of the contour plot as the following example shows

```
<?php
$isoBars = array(0.1, 0.2, 0.3, 0.4, 0.5, 0.6)
$cpt = new ContourPlot($data, $isoBars);
?>
```

The isobar colors can be adjusted by specifying a color array that specifies the color for each isobar.

However, there is some simpler changes that can be made to change the colors without having to go through the "trouble" of specifying a color array.

There are usually two reasons for changing the default color spectra.

1. to use more high contrast colors
2. to use only black and white colors

This can be accomplished with a call to the method

- `ContourPlot::UseHighContrastColor($aFlg=true, $aBW=false)`

as the following example shows

```
$cpt = new ContourPlot($data);
$cpt->UseHighContrastColor(true);
```

The above example will only use a blue to red color scale and avoid the "low-contrast" greenish middle color spectrum colors.

By also setting the second argument to the high contrast method to true the colors are restricted to only black and white. This will of course make it impossible to exactly know the value each isobar represents but could be useful for gaining some insight in the topography of a graph and would allow better printing on black and white devices.

Specifying manual colors of the isobar can be done in two ways.

1. By specifying the color array as the fifth argument to the constructor of `ContourPlot` as shown in Section 15.6.2, "Creating a contour graph"
2. By using the method `ContourPlot::SetIsobarColors($aColors)`

## Caution

The number of colors specified must match the number of isobars.

### 15.6.4. Understanding mesh interpolation

As mentioned above it is possible to use mesh interpolation to generate smoother contour graphs. In order to better understand the effect of specifying different interpolation factors we will show how what effect this have.

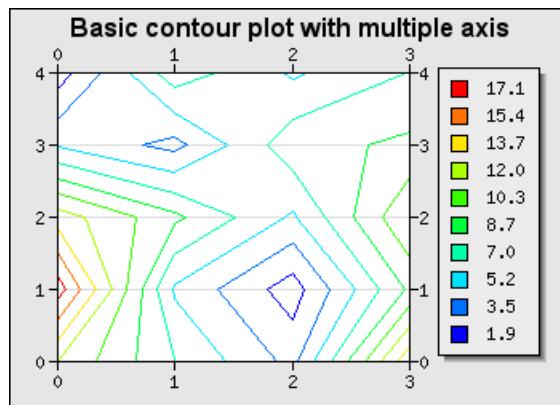
The data we will be using is given by the following data matrix

```
$data = array(
```

```
array (12,7,3,15),
array (18,5,1, 9),
array (13,9,5,12),
array (5,3,8, 9),
array (1,8,5, 7));
```

A basic contour graph using the above data is shown in Figure 15.79, “Interpolation factor=1 (basic\_contourex03-1.php)”

**Figure 15.79. Interpolation factor=1 (basic\_contourex03-1.php)  
[example\_src/basic\_contourex03-1.html]**



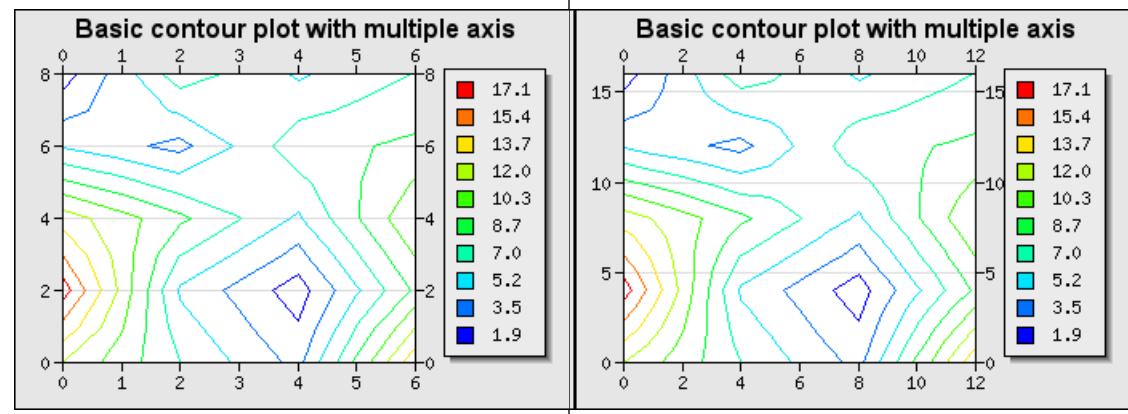
In order to specify a grid interpolation factor we need to specify this as the third argument in the constructor for the contour plot. So in order to specify a interpolation factor of 2 we would add the line

```
$contourplot = new ContourPlot($data,10,3);
```

To see the difference between different interpolation factor Figure 15.80, “Interpolation factor=2 (basic\_contourex03-2.php)” and Figure 15.81, “Interpolation factor=3 (basic\_contourex03-3.php)” shows the effect of using both a grid interpolation factor of 2 and 3 on the example in Figure 15.79, “Interpolation factor=1 (basic\_contourex03-1.php)”

**Figure 15.80. Interpolation factor=2 (basic\_contourex03-2.php)  
[example\_src/  
basic\_contourex03-2.html]**

**Figure 15.81. Interpolation factor=3 (basic\_contourex03-3.php)  
[example\_src/  
basic\_contourex03-3.html]**



We can now make two observations based on the above two figures

1. The lines get smoother the higher interpolation factor we use
2. The automatic scale has changed. This is not surprising since we have in effect increase the number of data points each time we increase the interpolation factor. The scale corresponds to the matrix size (an entry in matrix at position (r,c) corresponds to the point (x=c,y=r) in the graph.)

Initially we had a data matrix of size  $5 \times 4 = 20$  data points, after doing one interpolation (factor=2) the data matrix will have a size of  $9 \times 7 = 63$  data points and after doing one more interpolation (factor=3) the data matrix will have a size of  $17 \times 13 = 221$  data points.

If we would have done one more interpolation we would have a data matrix of size  $33 \times 25 = 825$  data points.

The exponential growth of the total number of data point for a  $10 \times 10$  matrix is shown in Figure 15.82, “The exponential growth of the data size due to grid interpolation factor (interpolation-growth.php) ”. This also makes a good example of using a logarithmic y-scale. If the graph is a straight line when plotted against a logarithmic scale this confirms our suspicion that this is indeed a exponentially growing function. Figure 15.83, “The exponential growth of the data size due to the grid interpolation factor (log scale) (interpolation-growth-log.php) ” shows a variant with a logarithmic y-scale (which confirms our suspicion.)

## Tip

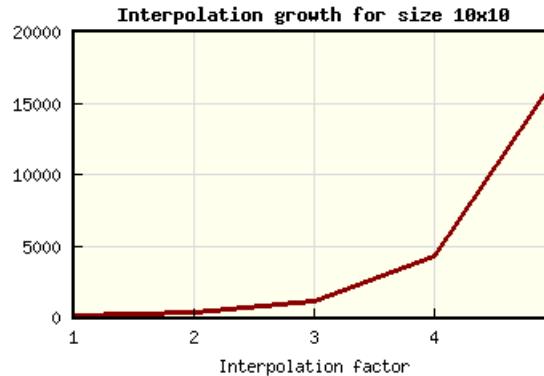
The helper class to do the actual interpolation can also be used directly in other context. In order to create a new interpolated data matrix with a specific interpolation factor the following code snippet can be used.

```
<?php
$dataMatrix = array(...);
$factor = ... ;
$grid_interpolate = new Interpolate();
$dataMatrix = $grid_interpolate->Linear($dataMatrix, $factor);
?>
```

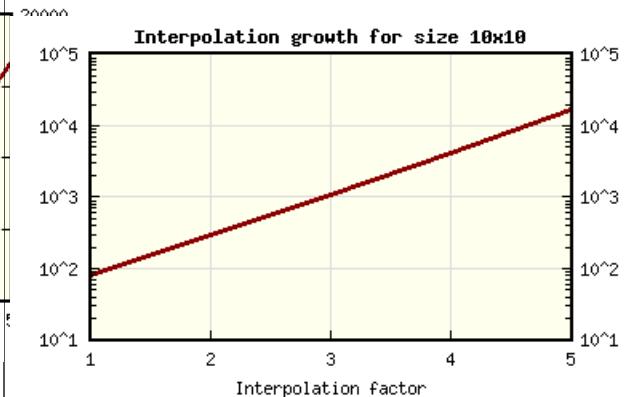
## Tip

There is also a utility function `doMeshInterpolate(&$aData, $aFactor)` that can be sued to interpolate a matrix in place. This creates the necessary class, interpolates its first argument and then fills the argument with the interpolated data. By using call by reference some data copying is avoided.

**Figure 15.82.** The exponential growth of the data size due to grid interpolation factor (interpolation-growth.php) [example\_src/interpolation-growth.html]



**Figure 15.83.** The exponential growth of the data size due to the grid interpolation factor (log scale) (interpolation-growth-log.php) [example\_src/interpolation-growth-log.html]

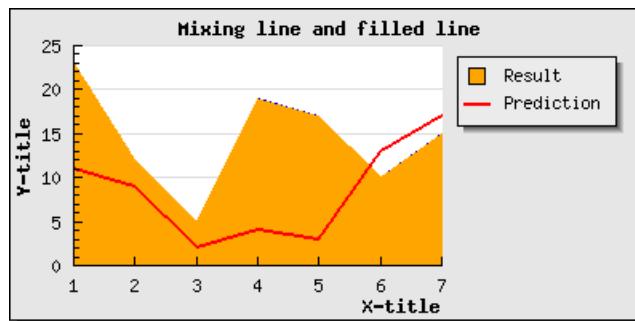


## 15.7. Combining several different plot types in the same graph

It is perfectly legal to add several different plot types to the same graph. For example to mix a line plot with a filled area plot. The different plots will be stroked to the graph in the order they are added. This means that the plot (data series) that should always be visible should be added last. In the example with the filled area and a line the line should probably be added last since it otherwise most likely will be overwritten by the filled area plot.

In Figure 15.84, “Mixing a line and area plot in the same graph (example16.1.php) ” a basic example is shown where we have combined a line plot with an area plot.

**Figure 15.84.** Mixing a line and area plot in the same graph (example16.1.php) [example\_src/example16.1.html]

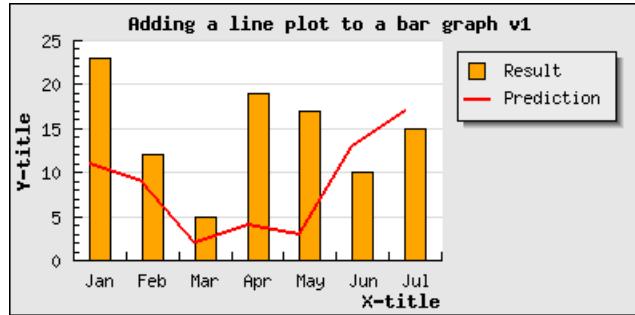


As can be seen from looking at the source to the example in Figure 15.84, “Mixing a line and area plot in the same graph (example16.1.php) ” there is nothing special in doing this other then to remember to include all necessary header files. So to add and mix scatter, line, error and stock graphs are pretty straightforward.

However, when it comes to mixing bar plots and lines (or area plot) there is a complication. By default bar plots are centered *between* the tick marks since this is the standard way of displaying bar graphs since they normally have no specific x-scale. The bars are just in an ordered sequence.

Let's just see what happens if we create a basic bar with a text scale on the x-axis and then also add a line plot to the same graph.

**Figure 15.85. Mixing a line and bar plot in the same graph (example16.3.php) [example\_src/example16.3.html]**



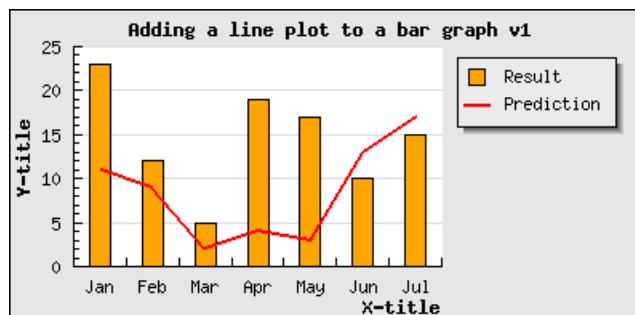
As can be seen from Figure 15.85, “Mixing a line and bar plot in the same graph (example16.3.php)” the line is adjusted so that each data point on the line coincides with the left edge of the barplot and the bars are still positioned in the middle of the tick marks as could be expected. This is the default behavior when combining a barplot with one of the other linear plots.

If we instead wanted the line to be centered in the barplot (when using a text scale) we would have to tell the line plot to position itself in the middle of the bars with the method

- `LinePlot::SetBarCenter()`

Using this method would then give the result shown in Figure 15.86, “Centering the line plot in the middle of the bar (linebarcentex1.php)” below

**Figure 15.86. Centering the line plot in the middle of the bar (linebarcentex1.php) [example\_src/linebarcentex1.html]**



Please note that the above discussion is only valid when using a text scale. if we instead would use (for example) an integer scale then both the bar and the line would be positioned at the exact scale values (i.e. both bars and lines would be left aligned) as can be seen in Figure 15.87, “Mixing bar and line using an integer x-scale (example16.4.php)”

**Figure 15.87. Mixing bar and line using an integer x-scale (example16.4.php)**  
[example\_src/example16.4.html]

### Note

As can be seen in Figure 15.87, “Mixing bar and line using an integer x-scale (example16.4.php)” it is possible to manually set the scale labels even if we use a linear scale like an integer scale. We only have to keep in mind that we ned to supply labels for all the scale labels and not just the one that are shown in case the scale is fro example only displaying a label on every second tick.

An example on how to combine a scatter plot and a line plot is shown in Section 18.1, “Linear regression analysis” where a linear regression analysis is done.

## 15.8. Creating several graphs in the same image

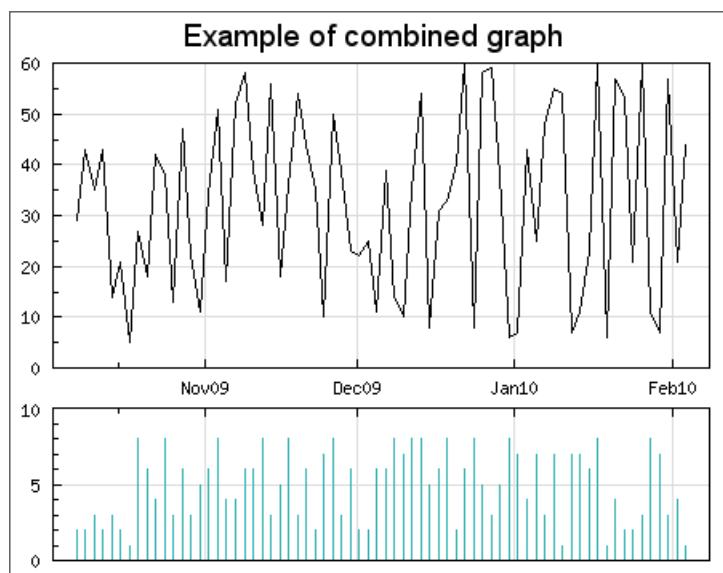
There are at least two reasons why one or several graphs should be combined into the same image.

- To be able to create overall graphs that are strongly connected and cannot be separated
- To reduce the download time by only have to download one image for several graphs
- By convention, some types of graphs are always combined

The library provides a convenience class , `class MGraph` (for Multi Graph) that assists with this. In order to use this class the module file "jpgraph\_mgraph.php" must be included.

Figure 15.88, “A combination of a line graph at top and a bar graph in the bottom (combgraphex1.php)” shows an example of how this can be used.

**Figure 15.88. A combination of a line graph at top and a bar graph in the bottom (combgraphex1.php)** [example\_src/combgraphex1.html ]



## 15.8.1. Creating a combined graph

In order to better understand what is involved in creating a combined graph we will first show how this is done manually.

This is done by creating some PHP/GD code that creates a big enough empty image with plain GD commands and then get the image handles from the graphs that should be combined. The image handle can be returned from the `Graph::Stroke()` methods in the graphs that should be included. It is then possible to use these handles together with the GD image copy command and copy the individual graph images onto the previously created empty large GD image.

The following (almost complete) code shows how this can be done

```
<?php
// Assume we would like to combine graph1,2 and 3
// create graph 1 here.....
$handle1 = $graph1->Stroke(_IMG_HANDLER);

// create graph 2 here.....
$handle2 = $graph2->Stroke(_IMG_HANDLER);

// create graph 3 here.....
$handle3 = $graph3->Stroke(_IMG_HANDLER);

// Now create the "melting graph" which should contain all three graphs
// $WIDTH1 and $HEIGHT1 are width and height of graph 1 ($handle1)
// $WIDTH2 and $HEIGHT2 are width and height of graph 2 ($handle2)
// $WIDTH3 and $HEIGHT3 are width and height of graph 3 ($handle3)
// $x2,$x3 and $y2,$y3 shift from top left of global graph (ie position of
// graph2 and graph3 in global graph)

$image = imagecreatetruecolor($WIDTH1,$HEIGHT1);
imagecopy($image, $handle1, 0, 0, 0, $WIDTH1,$HEIGHT1);
imagecopy($image, $handle2, $x1,$y1,0,0,$WIDTH2,$HEIGHT2);
imagecopy($image, $handle3, $x2,$y2,0,0,$WIDTH3,$HEIGHT3);

// Stream the result back as a PNG image
header("Content-type: image/png");
imagepng ($image);
?>
```

The advantages with this fully manual method are

- Absolute full control on a pixel level how the graphs are combined
- Full transparency on how the combination is done

but on the other hand the disadvantages are

- Requires knowledge of the GD library
- Requires re-inventing the wheel (since a lot of functionality to manipulate the GD library is already available in the JpGraph library)

It is here the class `MGraph` comes in. It will allow you to very easily create a combination of several graphs without knowing all the details on the GD library or calculating pixel positions.

This helper class abstracts away all the details about copying images as well as offering consistent interface (almost identical with the standard Graph class). A basic example will quickly reveal the elegance of using this helper class.

```
<?php
//-----
// Graph 1
//-----
$graph1 = new Graph(...);

//... [Code to create graph1] ...

//-----
// Graph 2
//-----
$graph2 = new Graph(...);

//... [Code to create graph2] ...

//-----
// Create a combined graph
//-----
$mggraph = new MGraph();
$xpos1=3;$ypos1=3;
$xpos2=3;$ypos2=200;
$mggraph->Add($graph1,$xpos1,$ypos1);
$mggraph->Add($graph2,$xpos2,$ypos2);
$mggraph->Stroke();
?>
```

The above example illustrates the most basic usage of the `MGraph` class.

Each graph is added to the combined graph with a call to `MGraph::Add()` which in addition to the graph argument also takes the target X and Y coordinates where the graph will be placed in the combined image. The target X and Y coordinates are relative to any optional specified left and top margin.

In addition to adding graphs there are some basic formatting option for the combined graph that are listed below. The core methods provided by the `MGraph` class are

- `MGraph::MGraph($aWidth,$aHeight).`

Create a new instance of the `MGraph` class. If the width and height are supplied as arguments they will override the automatically determined width and height.

- `MGraph::Add($aGraph,$aToX,$aToY,$aFromX,$aFromY,$aWidth,$aHeight).`

Add a new subgraph with destination position (`$aToX, $aToY`). By default the whole graph image is used. It is also possible to only specify that part of the graph should be used. The source rectangle is specified by (`$aFromX, $aFromY, $aWidth, $aHeight`).

- `MGraph::SetFillColor($aColor).`

Background color of the combined graphs.

Example: Setting an Orange background

```
$mgraph->SetFillColor('orange');
```

- `MGraph::SetFrame($aShow, $aColor, $aWeight)`

Adding a frame to the combined graph.

Example: To add a blue frame with a 2 pixel width

```
$mgraph->SetFrame(true, 'blue ', 2);
```

- `MGraph::SetMargin($aLeft, $aRight, $aTop, $aBottom)`

Adding a margin to the image. By default the minimum necessary size for the combined graph to hold all the added subgraphs is used. Use this method to add some space to the edges around the combined image. When the individual graphs are positioned (in the `Add()` method) the position is counted relative to the left and top margin.

Example: Use 10 pixel left and right margin, 5 pixel top and bottom margin

```
$mgraph->SetMargin(10,10,5,5);
```

- `MGraph::SetShadow($aShowShadow=true, $aShadowWidth=4, $aShadowColor='gray@0.3')`

This adds a drop shadow to the `MGraph` in the same way as for the usual graph class

In the same way as the ordinary `Graph` class the `MGraph` class also have a property `$footer` which allows the setting of a left, center and right footer (see Section 14.2.6, “Adding a footer to the graph”).

In addition the `MGraph::Stroke()` accepts a filename in the same way as the normal `Graph::Stroke()`

## 15.8.2. Adding background images

In addition to the basic usage as shown above it is also possible to add a background image in the combined graph. What is important to remember then is that all the subgraphs are copied onto the combined graph so normally they will be on top of the background image and hide it.

There are two ways of handling a background image

1. Do nothing and the background image will only "shine through" for those areas of the combined graph that is not covered by any subgraph.
2. Specify a "blend" factor for the subgraphs that specifies how much of the background image will be mixed with the subgraph. The blend factor is as usual specified as an integer in the interval 0-100. (Note: This is the same functionality as the standard `Graph::SetBackgroundImageMix()` for ordinary graphs.) A value of 0 means that 100% of the background is visible and a value of 100 means that 0% of the background is visible.

To add a subgraphs with a blend factor the method `MGraph::AddMix()` is used. This behaves exactly as the standard `MGraph::Add()` apart from the fact that the 3:rd argument is the blend factor (after the two x and y coordinates).

Example: Add a graph with 50% mix of the background

```
$mgraph->AddMix($graph,0,0,50);
```

The background image itself is added with the method

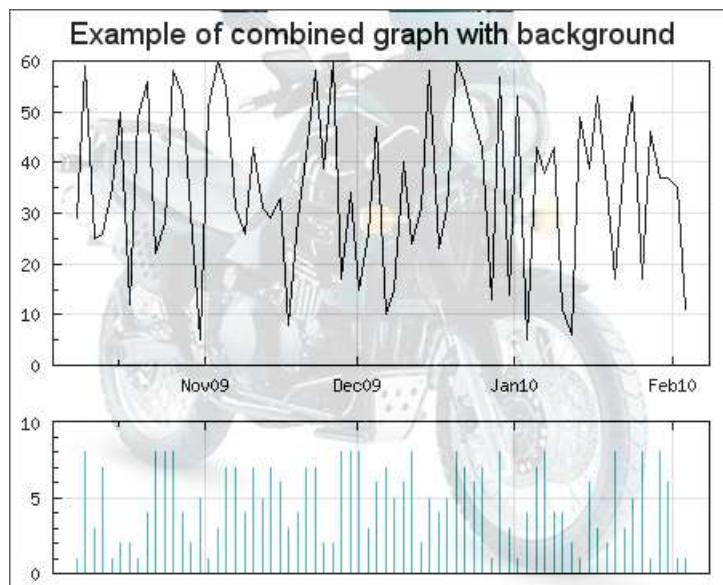
- `MGraph::SetBackgroundImage($aFile,$aCenter_aX=TRUE,$aY=NULL)`

The filename specifies the full name of the background image (including extension). Please note that the extension must be a valid image extension in order for the library to determine the format of the image file.

The second argument can be either numeric or a boolean in which case it specifies if the background image should be centered or if it should just be copied from the top left position. If it is numerical it is interpreted as the X-position for the optional X and Y coordinates specifying the position of the top left corner of the background image.

Figure 15.89, “Mixing a background image with two subgraphs. In this case the mixing factor was 85 for both subgraphs. (Note: To reduce load time the image is quite hard compressed in JPEG so there are some artifacts in high-frequency areas.) (`combgraphex2.php`)” shows a variant of Figure 15.88, “A combination of a line graph at top and a bar graph in the bottom (`combgraphex1.php`)” where a centered background image is mixed in with two subgraphs.

**Figure 15.89. Mixing a background image with two subgraphs. In this case the mixing factor was 85 for both subgraphs. (Note: To reduce load time the image is quite hard compressed in JPEG so there are some artifacts in high-frequency areas.) (`combgraphex2.php`) [example\_src/combgraphex2.html]**

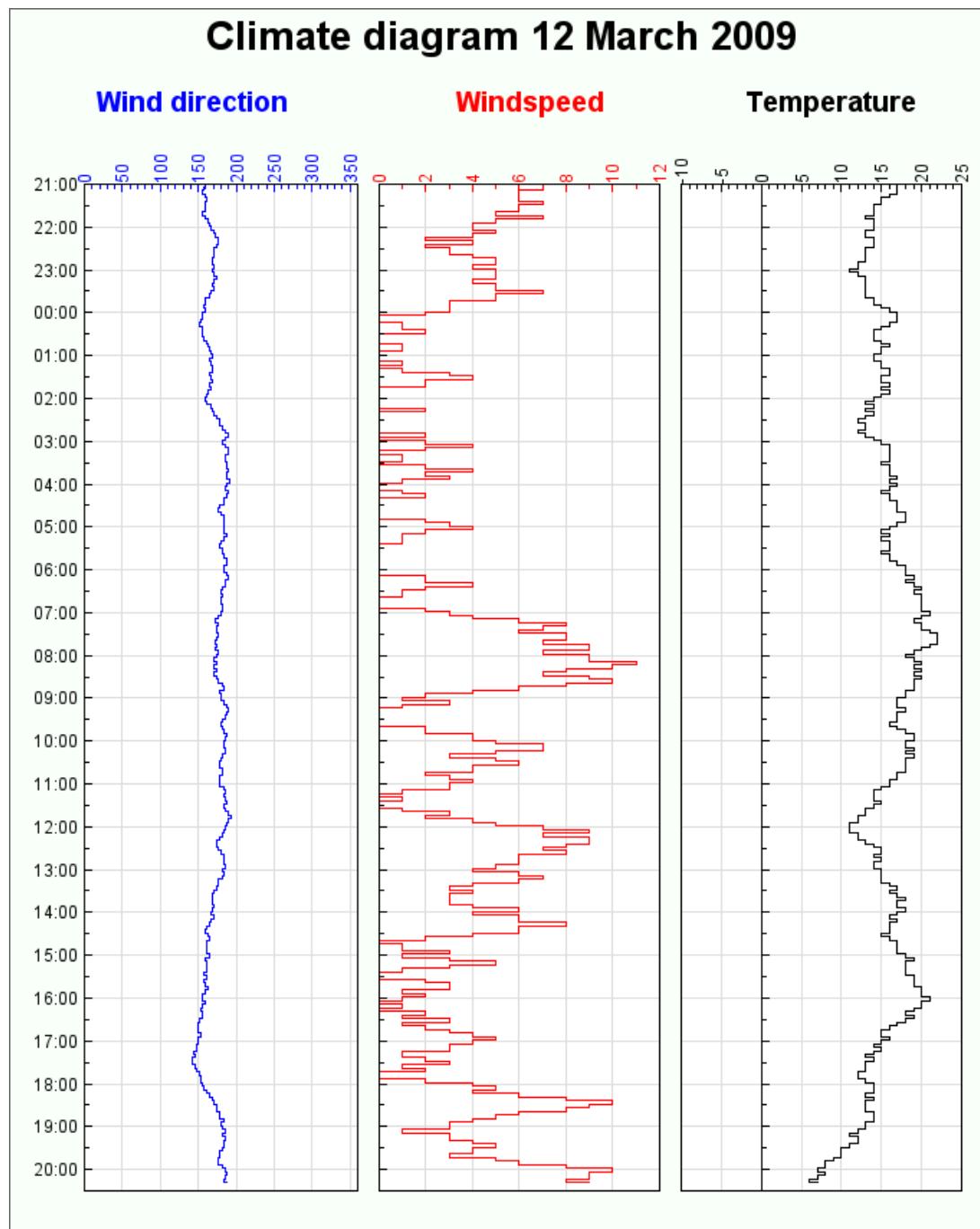


### 15.8.3. Creating rotated combined graphs

There are in principle no difference when combining graphs top/bottom or left/right. ?? shows an example of creating a graph that illustrates the changing weather conditions measure by Wind direction, speed and temperature in one graph with a detailed vertical scale showing the time.

Since this follows exactly the same principle laid out above the script should be fairly self explanatory. Note that we have suppressed the x-scale on two of the graphs to make better use of the available image size and also to create a stronger sense of "belonging" between the individual graphs.

Figure 15.90. Combining three graphs in one image  
(`comb90dategraphhex03.php`)  
[`example_src/comb90dategraphhex03.html`]



#### 15.8.4. Some caveats when using MGraph

There are some drawbacks using combined graphs

- CSIM Image maps can not be used with combined graphs.

- Creating multiple graphs in the same script will require some additional memory. Make sure that the memory limit in "php.ini" is sufficient.
- Adding background images can significantly increase the final size of the image. Using the JPEG image format when photo like background images are used gives better compressing than PNG (since JPEG is a lossy format). For example setting the JPEG quality parameter to 60 (default is 75).

```
$mgraph->SetImgFormat('jpeg' , 60);
```

### Tip

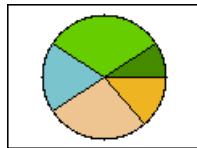
This setting can be used for ordinary Graphs as well.

- Compared with background images for ordinary graphs there is currently no possibility for automatic re-sizing of the background images.

# Chapter 16. Non-Linear graph types

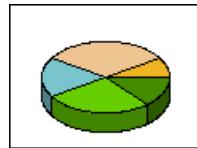
All the plots we have discussed up to now have used some kind of x-y coordinate system. The library supports a number of graph types that does not use rectangular axis or scales. The supported graphs are shown below

**Figure 16.1. Pie graphs**



(See Section 16.1.2,  
“2D-Pie plots”)

**Figure 16.2. Pie3D graphs**



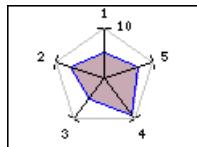
(See Section 16.1.3,  
“3D-Pie plots”)

**Figure 16.3. Ring graphs**



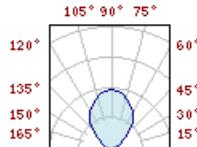
(See Section 16.1.4,  
“Ring plots”)

**Figure 16.4. Radar graphs**



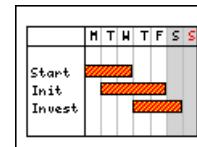
(See Section 16.2,  
“Radar graphs”)

**Figure 16.5. Polar graphs**



(See Section 16.3,  
“Polar graphs”)

**Figure 16.6. Gantt charts**



(See Section 16.4,  
“Gantt charts”)

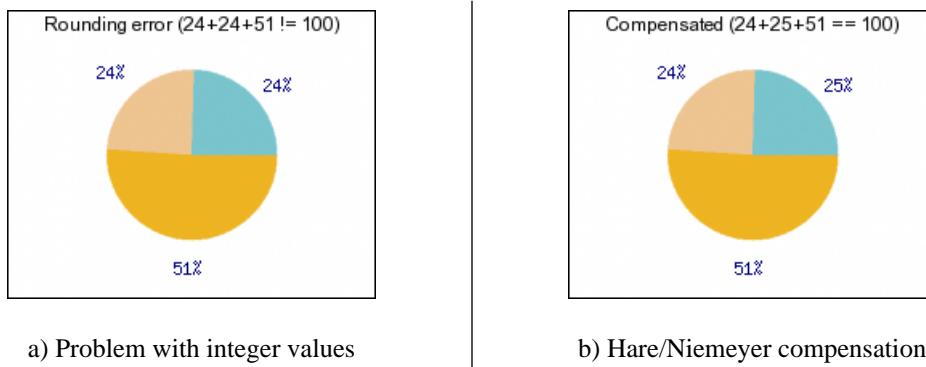
## 16.1. Pie graphs

The main difference compared to the previous discussed x-y plots is that to all pie plots are added based on the class `PieGraph` instead of the `Graph` class used for all x-y graphs.

### 16.1.1. Hare/Niemeyer Integer compensation for Pie Plots

A problem that sometimes occur with the use of integer values for Pie charts is the problem with rounding error. Even with correct rounding of the individual pie slices using only integers as display values have the problem that it sometimes doesn't sum up to 100% as illustrated in Figure 16.7, “Hare/Niemeyer pie plot integer compensation”

This is in many circumstances not acceptable. The library includes the Hare/Niemeyer compensation that can be enabled. This will adjust the integer values so that it always sum up to 100%. The result of the compensation can be seen in Figure 16.7, “Hare/Niemeyer pie plot integer compensation”

**Figure 16.7. Hare/Niemeyer pie plot integer compensation****Note**

To enable this compensation the label type should be specified as

```
$pieplot->SetLabelType(PIE_VALUE_ADJPERCENTAGE);
```

## 16.1.2. 2D-Pie plots

In order to create Pie plots the module "jpgraph\_pie.php" must first be included.

The principle for creating Pie graphs follows the same structure as for line graphs. First an instance of class `PieGraph` is created and then one or several instances of class `PiePlot` is created and added to the `PieGraph` instance as the following basic example shows

### Example 16.1. A basic Pie graph (`example26.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_pie.php");

$data = array(40,60,21,33);

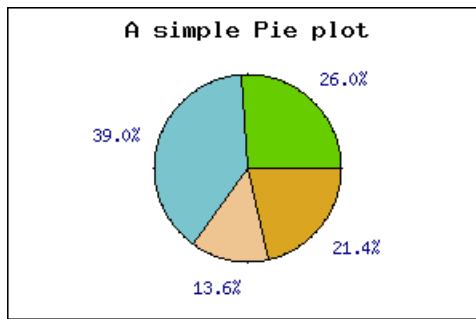
$graph = new PieGraph(300,200);
$graph->SetShadow();

$graph->title->Set("A simple Pie plot");

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

?>
```

**Figure 16.8. A basic Pie graph (`example26.php`) [[example\\_src/example26.html](#)]**



There are a few things to note here

- By default all pie slices have the percentage shown just outside the slice.
- The colors are automatically assigned to the slices.
- The pie have the edges marked by default
- The first slice start at 0 degrees (3 o'clock)

It is possible to adjust most of the aspects of a pie graph. in addition to the usual graph appearance adjustments discussed in Section 14.2, “Specifying and formatting the overall displayed graph” the following methods can be used to adjust the appearance of the individual plots.

- `PiePlot::SetStartAngle($aAngle)`

Change the angle for the first slice

- `PiePlot::ShowBorder($aFlg=true)`

Show the border around and inside the pie

- `PiePlot::SetColor($aColor)`

Specify the color of the border of the pie

- `PiePlot::SetSliceColors($aColor)`

Specify the color of the slices

- `PiePlot::setSize($aSize)`

Set the size of the pie plot. This can be specified as either a fraction of the minimum of the graph width and height or as a absolute size in pixels.

- `PiePlot::SetCenter($aX, $aY)`

Position the pie on the graph

- `PiePlot::value`

- `PiePlot::SetLabels($aLabels, $aLblPosAdj="auto")`

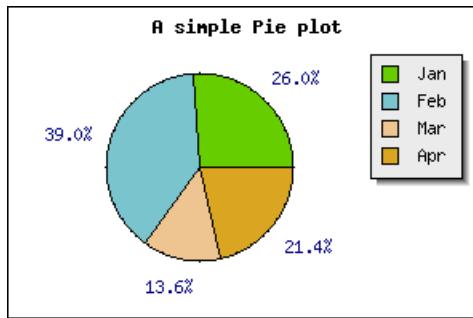
Specify the labels for each pie slice

- `PiePlot::SetLabelType($aType)`

By default the label is displayed as percentage. Using this method it is possible to change the automatic label to show the absolute value instead.

In order to show a basic addition we add a legend to the pie graph. We do this by using the `SetLegends()` method. By adding the legends to the previous example we get the result shown in Figure 16.9, “Adding a legend to a pie plot (example26.1.php)”

**Figure 16.9. Adding a legend to a pie plot (example26.1.php) [example\_src/example26.1.html]**



### Note

In the figure above we also moved the center of the pie slightly to the left to make more room for the legend box.

The text for the legends can also contain `printf()` style format strings to format a number. The number passed on into this string is either the absolute value of the slice or the percentage value. The same formatting can also be used to format the legend labels.

## Adding multiple pie plots to the same pie graph

This is done in complete analogy with the say multiple lines were added to a line graph. The following code snippet shows the principle

```
<?php
$piegraph = new PieGraph($width,$height);

$p1 = new PiePlot($data1);
$p1->SetSize(0.2);
$p1->SetCenter(0.3,0.6);

$p2 = new PiePlot($data2);
$p2->SetSize(0.2);
$p2->SetCenter(0.7,0.6);

$piegraph->Add($p1);
$piegraph->Add($p2);
?>
```

The positioning and sizing of the plots must be done manually. In the following example we show an example where four pie plots are added to a graph

**Example 16.2. Adding several pie plots to the same pie graph (`pieex3.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_pie.php");

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

// Create the Pie Graph.
$graph = new PieGraph(350,300);
$graph->SetShadow();

// Set A title for the plot
$graph->title->Set("Multiple - Pie plot");
$graph->title->SetFont(FF_FONT1,FS_BOLD);

// 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->value->SetFont(FF_FONT0);
$p1->title->Set("2001");

$p2 = new PiePlot($data);
$p2->SetSize($size);
$p2->SetCenter(0.65,0.32);
$p2->value->SetFont(FF_FONT0);
$p2->title->Set("2002");

$p3 = new PiePlot($data);
$p3->SetSize($size);
$p3->SetCenter(0.25,0.75);
$p3->value->SetFont(FF_FONT0);
$p3->title->Set("2003");

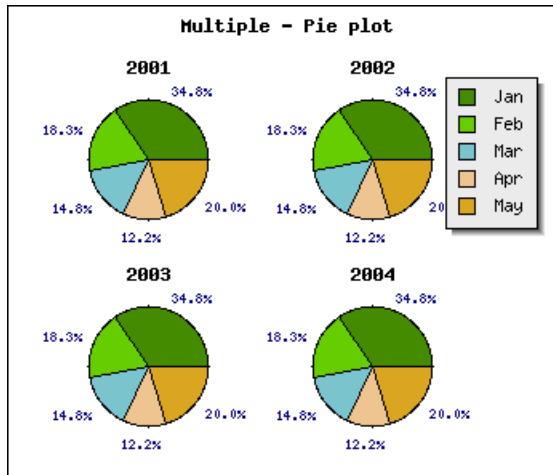
$p4 = new PiePlot($data);
$p4->SetSize($size);
$p4->SetCenter(0.65,0.75);
$p4->value->SetFont(FF_FONT0);
$p4->title->Set("2004");

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

$graph->Stroke();

?>
```

**Figure 16.10. Adding several pie plots to the same pie graph (`pieex3.php`) [[example\\_src/pieex3.html](#)]**



## Adding guide lines to Pie Plots

For very busy Pie plots it can become too little space for the labels to be printed just beside the pie slice. For this purpose it is possible to use guide lines for the labels. The library will then draw a line from the center edge of the slices to the label which will be positioned further out from the Pie Plot.

There is one method that is primarily used to handle this

- `PiePlot::SetGuideLines($aFlg=true, $aCurved=true, $aAlways=false)`

the simplest usage of this would be to add the following line to a script

```
$pieplot->SetGuideLines();
```

Figure 16.11, “Adding guide lines to a pie labels (`pielabelsex1.php`)” shows an example of this.

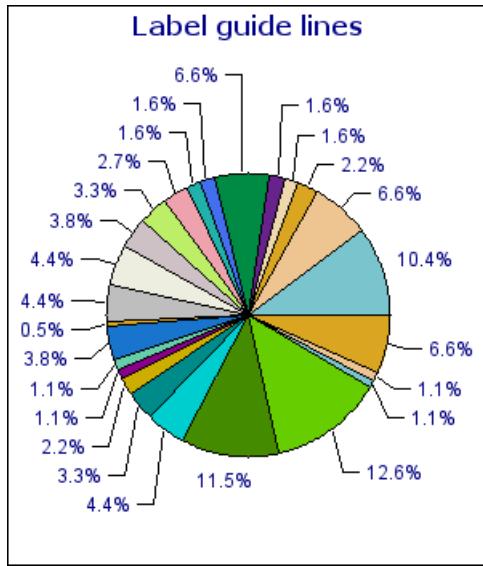
This basic use of guide lines is very similar as what is produced by other programs, e.g. Excel. In addition to the above variant it is also possible to instruct the library to line up the labels vertically in a way that we think is easier to read.

This is achieved by specifying the second parameter to the `SetGuideLines()` to 'false' as

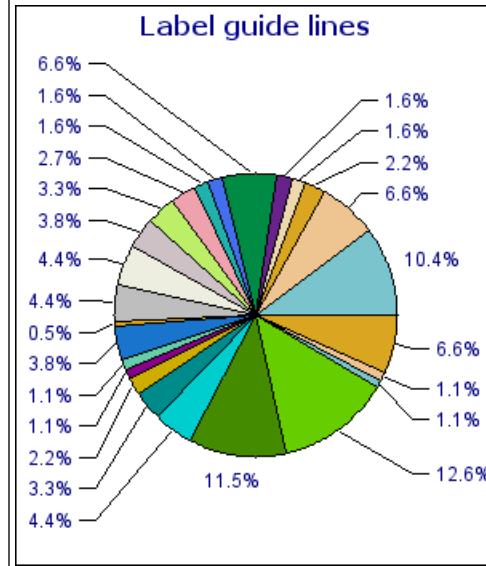
```
$pieplot->SetGuideLines(true , false);
```

The first parameter is to enable/disable the guide-lines. With the same example as above this would then produce the image shown in Figure 16.12, “Lining up guide lines vertically (`pielabelsex2.php`)”

**Figure 16.11. Adding guide lines to a pie labels (`pielabelsex1.php`)**  
[`example_src/pielabelsex1.html`]



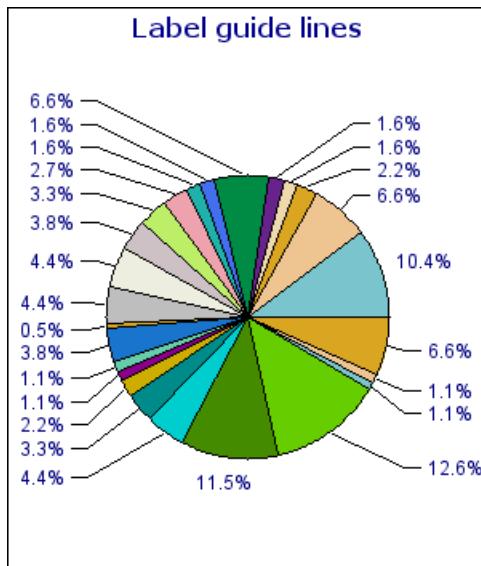
**Figure 16.12. Lining up guide lines vertically (`pielabelsex2.php`)**  
[`example_src/pielabelsex2.html`]



It is also possible to configure the vertical distance between the labels. By default the distance between the labels is roughly 40% of the labels font height. By using the method `PiePlot::SetGuideLinesAdjust()` it is possible to specify a fractional value which is interpreted as the distance between the bottom of one label to the bottom of the next. This means that specifying a value of '1.0' the labels will have no space between them and the bottom of one label will touch the top of another label. By default this value is 1.4.

By increasing or decreasing this value it is possible to make the labels become positioned more or less compact. In ?? this distance is reduced to '1.1' and as can be seen this yields much more compact labeling.

**Figure 16.13. Adjusting the distance between the labels for guide lines (pielabelsex4.php) [example\_src/pielabelsex4.html]**



### Note

Guide lines is only available for 2D Pie plots.

## 16.1.3. 3D-Pie plots

In order to create 3D Pie plots both the modules "jpgraph\_pie.php" and "jpgraph\_pie3d.php" must be included.

Creating 3D pie plots is as simple as creating normal 2D pie plots. Instead of creating the plots as an instance of `class PiePlot` the plots are created as an instance of `class PiePlot3D`

If we take the plot in Figure 16.8, “A basic Pie graph (example26.php) ” and replace the creation of an instance of `class PiePlot` with creating an instance of `class PiePlot3D`, i.e. replace

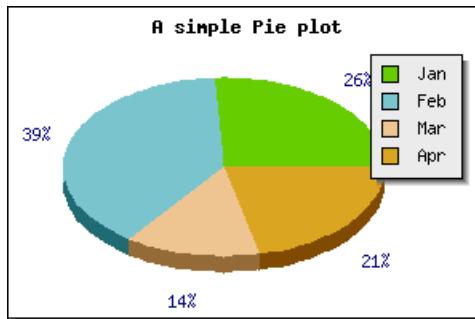
```
$p1 = new PiePlot($data);
```

with

```
$p1 = new PiePlot3D($data);
```

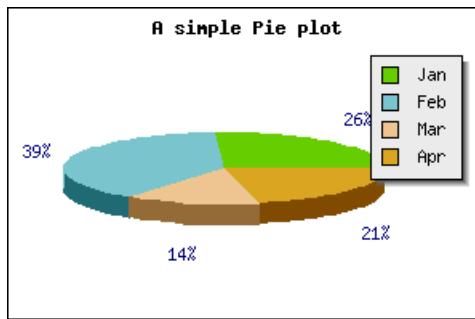
The result of this is shown in Figure 16.14, “A basic 3D pie plot (example27.php) ”

**Figure 16.14. A basic 3D pie plot (`example27.php`) [`example_src/example27.html`]**



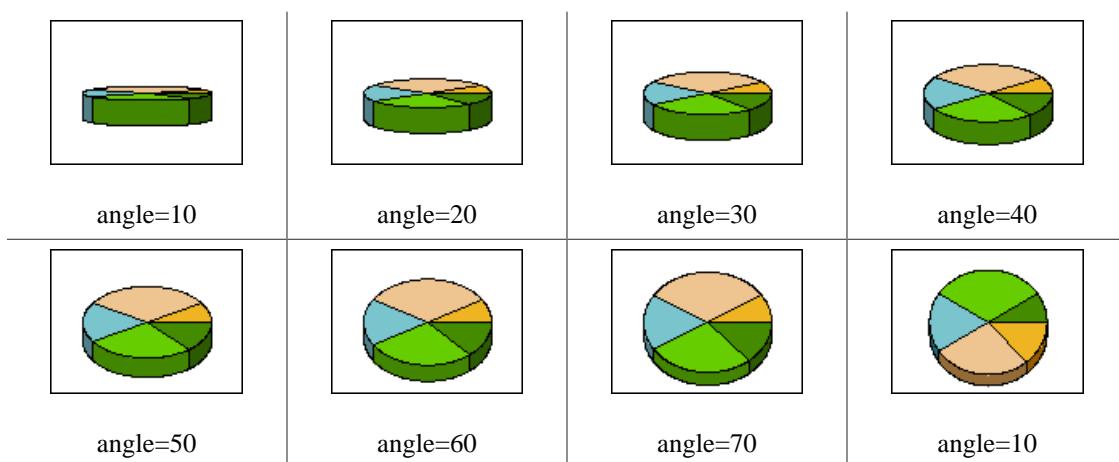
3D Pie plots have the same possibilities as the normal pie plots with the added twist of a 3:rd dimension. It is possible to adjust the perspective angle with the method `SetAngle()`. So for example to make the pie more "flat" the angle is made smaller angle. Setting the perspective angle to 20 degrees in the previous example will give the following result.

**Figure 16.15. Adjusting the perspective angle (`example27.1.php`) [`example_src/example27.1.html`]**



To give a feel for the effect of adjusting the angle Figure 16.16, “Affect of adjusting the perspective angle for a 3D pie plot” below shows a number of different angles,

**Figure 16.16. Affect of adjusting the perspective angle for a 3D pie plot**

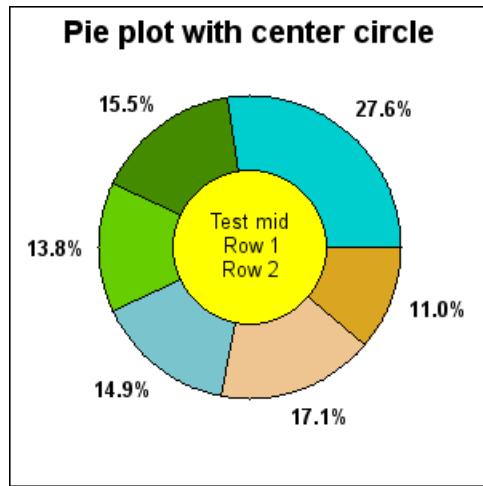


## 16.1.4. Ring plots

There are two versions of the 2D pie plots. The normal pie plot is created as an instance of class `PiePlot`. This variant is created as an instance of class `PiePlotC`.

This variant is an extension of the standard `PiePlot` in the sense that it also have a filled circle in the center. An example of this is shown in ??

**Figure 16.17. A ring plot (`piececex1.php`) [example\_src/piececex1.html]**



Since the `PiePlotC` is an extension to the basic pie plot all the normal formatting for pie plots can also be done for a "ring" plot.

The additional formatting only concerns the filled middle circle. There are options of adjusting size, fill color and all font properties. The following methods are used for this

- `PiePlotC::SetMidColor()`  
Set fill color of mid circle
- `PiePlotC::SetMidSize()`  
Set size (fraction of radius)
- `PiePlotC::SetMidTitle()`  
Set title string (may be multi-lined)
- `PiePlotC::SetMid()`  
Set all parameters in a single method call

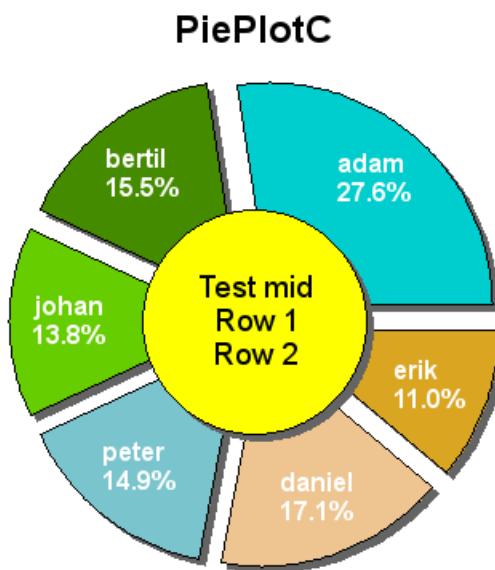
In addition to the normal CSIM for `PiePlot`:s the center area is also a CSIM hotspot. You specify the target CSIM with a call to

- `PiePlotC::SetMidCSIM()`

A more complex formatting of a ring plot is shown in Figure 16.18, "A ring graph with several formatting options adjusted (`piececex2.php`)". In this example we have:

- hidden the frame around the pie graph
- exploded all the slices
- added drop shadow to the individual slices (and the center filled circle)
- specified individual multi line labels.
- changed the font for the title to a TTF font.

**Figure 16.18.** A ring graph with several formatting options adjusted (`piecex2.php`) [[example\\_src/piecex2.html](#)]



## 16.1.5. Exploding pie slices

One way to attract attention to some specific piece of information in a pie chart is to "explode" one or more slices. Both 2D and 3D pies support exploding one or several slices.

Exploding slices is accomplished by the methods

- `PiePlot::Explode($aExplodeArr)`

This method takes an array of one or more slices index to explode

`PiePlot::ExplodeAll($aRadius)`

This explodes all slices in the pie

`PiePlot::ExplodeSlice($aSlice, $aRadius)`

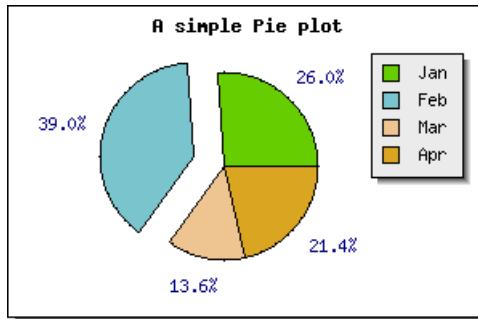
This is used to explode one specific slice

To explode one slice the default "explode" radius the following line has to be added

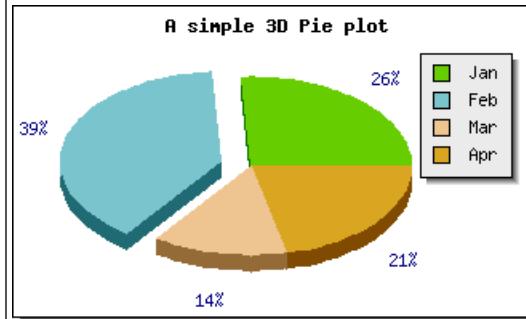
```
$pieplot->ExplodeSlice(1);
```

The above line would explode the second slice (slices are numbered from 0 and upwards) the default amount. Doing this to the previous examples would result in the following two figures

**Figure 16.19. Exploding the second slice (example27.2.php) [example\_src/example27.2.html]**



**Figure 16.20. Exploding the second slice (example27.3.php) [example\_src/example27.3.html]**



## 16.1.6. Specifying and adjusting labels on pie plots

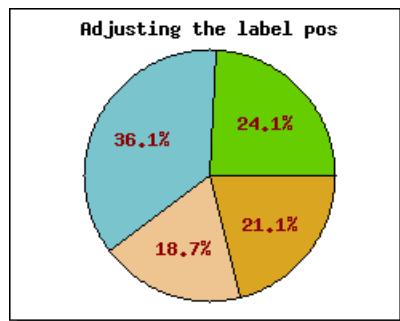
By default the values shown just outside the pie for each slice are the percentage value for each slice. If instead the absolute value should be displayed the `SetLabelType()` method must be called. To show absolute values the following line must be added

```
$pieplot->SetLabelType(PIE_VALUE_ABS);
```

Furthermore it is possible to adjust the display value by either using a `printf()` style format string (using `SetFormat()`) or by providing a formatting function callback (using `PiePlot::SetFormatCallback()`) for doing more advanced formatting.

It is also possible to adjust the position of the labels by means of the `PiePlot::SetLabelPos()` method. The argument to this method is either the fraction of the radius or the string 'auto'. In the latter case the library automatically determines the best position. The following example illustrates the effect of specifying the position to 0.5, i.e. in the middle of the radius.

**Figure 16.21. Adjusting the position of the pie labels (pieex8.php) [example\_src/pieex8.html]**



If this formatting is not enough it is also possible to "manually" specify the labels for each slice individually.

This is done by using the `PiePlot::SetLabels()` method. This will allow specifying individual text strings for each label. In each specification it is possible to add a `printf()` formatting specification for a number. The number passed on will be either the absolute value for the slice or the percentage value depending on what was specified in the call to `SetLabelType()`.

The `SetLabels()` method can also take a second parameter, the label position parameter. This is a shortcut to the `PiePlot::SetLabelPos()` as described above. By default the position will be set to 'auto' if not explicitly specified.

### Note

The alignment of the labels will be different depending on whether they are inside or outside the pie.

When the label is positioned inside the plot the center of the strings will be aligned with the center of the slice at the specified fraction of the radius. When positioned is outside the pie plot the alignment will depend on the angle to avoid that the labels inadvertently writes over the pie.

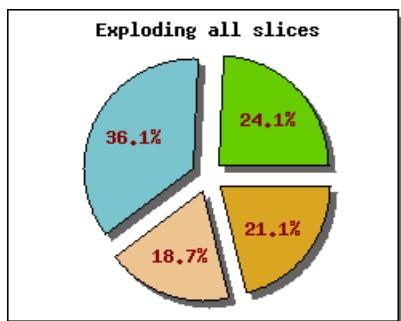
## 16.1.7. Adding drop shadows to the slices

An additional visual enhancement can be made by adding a drop shadow to the individual slices. This is accomplished by means of the

- `PiePlot::SetShadow()`

method. Adding a drop shadow is often more affective if the pie has one or more slices exploded as shown in ??

**Figure 16.22. Adding a drop shadow to exploded pie (`pieex9.php`) [[example\\_src/pieex9.html](#)]**



## 16.1.8. Adding background images to Pie graphs

In the same way as every other graphs in the library it is possible to add a background image to a Pie plot (see Section 14.15, “Adding images and country flags to the background of the graph”). An example of this where we have used multiple plots on a worldmap background is shown in Figure 16.23, “Pie plots with a background image (`piebkge1.php`)”

```

// Create the plots
for($i=0; $i < $n; ++$i) {
 $d = "data$i";
 $p[] = new PiePlot3D($data[$i]);
}

// Position the four pies
for($i=0; $i < $n; ++$i) {
 $p[$i]->SetCenter($piepos[2*$i],$piepos[2*$i+1]);
}

// Set the titles
for($i=0; $i < $n; ++$i) {
 $p[$i]->title->Set($titles[$i]);
 $p[$i]->title->SetColor('white');
 $p[$i]->title->SetFont(FF_ARIAL,FS_BOLD,12);
}

// Label font and color setup
for($i=0; $i < $n; ++$i) {
 $p[$i]->value->SetFont(FF_ARIAL,FS_BOLD);
 $p[$i]->value->SetColor('white');
}

// Show the percentages for each slice
for($i=0; $i < $n; ++$i) {
 $p[$i]->value->Show();
}

// Label format
for($i=0; $i < $n; ++$i) {
 $p[$i]->value->SetFormat("%01.1f%%");
}

// Size of pie in fraction of the width of the graph
for($i=0; $i < $n; ++$i) {
 $p[$i]->SetSize(0.15);
}

// Format the border around each slice

for($i=0; $i < $n; ++$i) {
 $p[$i]->SetEdge(false);
 $p[$i]->ExplodeSlice(1);
}

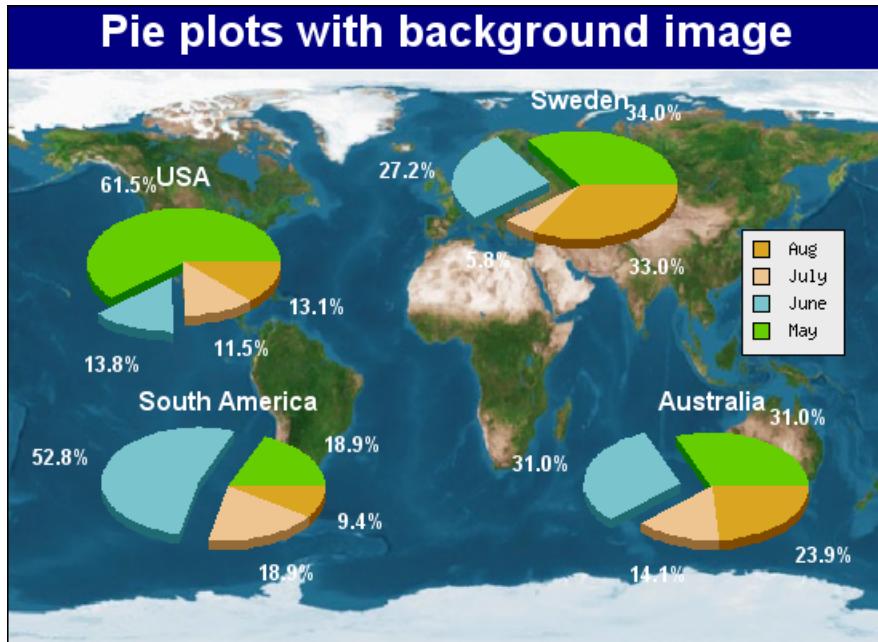
// Use one legend for the whole graph
$p[0]->SetLegends(array("May","June","July","Aug"));
$graph->legend->Pos(0.05,0.35);
$graph->legend->SetShadow(false);

for($i=0; $i < $n; ++$i) {
 $graph->Add($p[$i]);
}

$graph->Stroke();
?>

```

**Figure 16.23.** Pie plots with a background image (`piebkgex1.php`) [`example_src/piebkgex1.html`]



### 16.1.9. Specifying slice colors and using themes

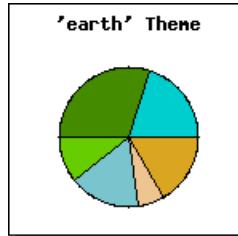
In addition to manually specifying the colors of each individual pie slice it is possible to specify a color "theme" to be used. A color theme is nothing more than a group of predefined colors that will be used for the slices. Each theme is referred to by its name (as a string) as argument for the method

- `PiePlot::SetTheme($aTheme)`

By default the library offers four different color themes listed below with a small example graph using that particular theme.

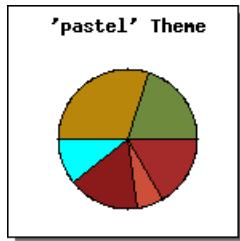
1. "earth" colors

**Figure 16.24.** (`example28.1.php`) [`example_src/example28.1.html`]



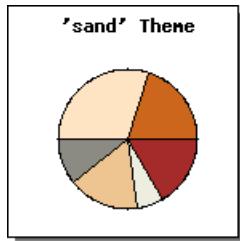
2. "pastel" colors

**Figure 16.25.** (example28.2.php) [example\_src/example28.2.html]



3. "sand" colors

**Figure 16.26.** (example28.php) [example\_src/example28.html]



4. "water" colors

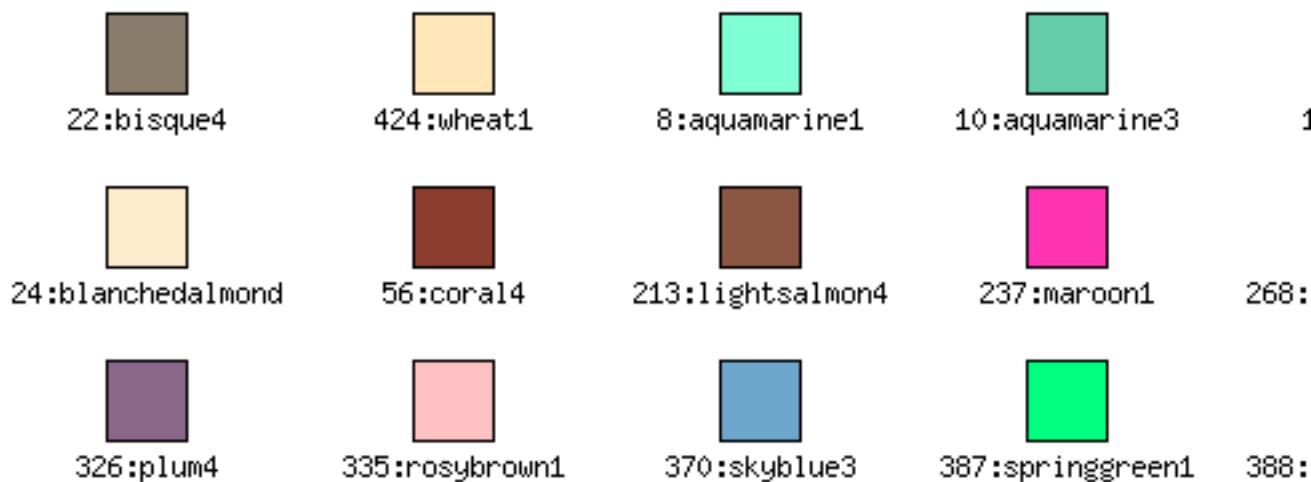
**Figure 16.27.** (example28.3.php) [example\_src/example28.3.html]



Color chart for each available theme is shown below.

**Figure 16.28. Earth theme**

**Figure 16.29. Pastel theme****Figure 16.30. Sand theme**

**Figure 16.31. Water theme**

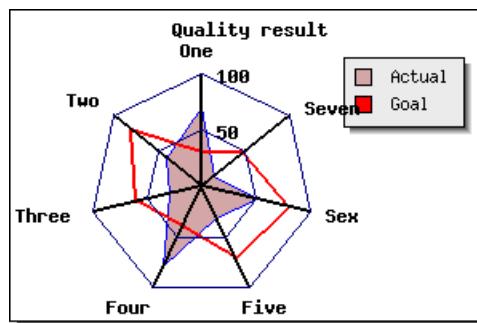
## 16.2. Radar graphs

In order to create Radar graphs the module "jpgraph\_radar.php" must first be included.

Radar 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). Radar plots are not suitable if high accuracy readings from the graph are needed since, by its nature, it can be difficult to read out very detailed values.

Figure 16.32, "A typical radar graph with two radar plots added (radarex7.php)" shows a basic radar graph with two radar plots added.

**Figure 16.32. A typical radar graph with two radar plots added (radarex7.php)**  
[[example\\_src/radarex7.html](#)]



Radar graphs are created as an instance of class `RadarGraph` which inherits all common formatting options from the standard class `Graph` such as titles, background colors, background titles, etc.

### Note

Tabbed titles cannot be used for radar graphs since there is no applicable concept of a plot area

The following facts applies to a radar graph

- There is one axis for each data point
- Each axis may have an arbitrary title which is automatically positioned
- A radar plot may be filled or open
- The first axis is always oriented vertical and is the only axis with labels
- Several radar plots can be combined in one radar graph
- Axis can have either linear or logarithmic scale on the axis

In addition all the normal formatting of labels, background colors, grid lines, plot colors, fill colors etc. can be adjusted. The rest of this sections will show some of the typical formatting options available.

### 16.2.1. Adding radar plots to a radar graph

The principle is exactly the same as for line plots but instead of an x-y coordinate each data point represents a value on one of the axis. The number of axis will automatically be the same as the number of data points. Even though the library itself does not impose any restrictions on the number of axis there is a practical limit so it is probably best to keep the number of data points limited to around 10-12.

Typical data set for a radar plot is then given as

```
$data = array(axis<1>_value, axis<2>_value, ..., axis<n>_value);
```

The plot points are assigned to the axis in a counter-clockwise direction starting at 12 o'clock. If clockwise order is needed the input data must be reversed.

Creation of a radar graph follows the now familiar steps of first creating an instance of `class RadarGraph` and then adding one or several instances of `class RadarPlot` as the following code snippet shows

```
<?php
// Some data
$data1 = array(...);
$data2 = array(...);

$width = ... ;
$height = ... ;

// Setup a basic radar graph
$graph = new RadarGraph($width,$height);
$graph->SetScale('lin');

// Add a title to the graph
$graph->title->Set('Quality result');

// .. add any other common graph formatting

// Create the first radar plot with formatting
$pplot1 = new RadarPlot($data1);
$pplot1->SetLegend('Goal');
```

```
$plot1->SetColor('#red', 'lightred');

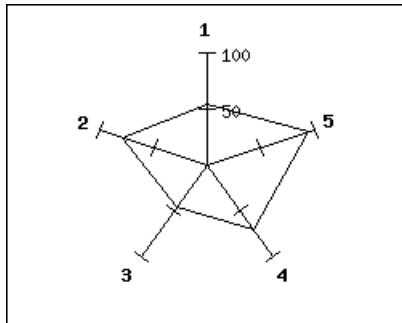
// Create the second radar plot
$plot2 = new RadarPlot($data2);
$plot2->SetLegend('Actual');
$plot2->SetColor('blue', 'lightred');

// Add the plots to the graph
$graph->Add($plot2);
$graph->Add($plot1);

// and display the graph
$graph->stroke()
?>
```

As an illustration two very basic radar graphs with one radar plot each are displayed below. The left figure uses only default values and the right figure uses the same data but adjusts a few properties to make the graph a bit more interesting.

**Figure 16.33. A basic radargraph with no formatting (`radarex1.php`)**  
[[example\\_src/radarex1.html](#)]



**Figure 16.34. A basic radargraph with minimal formatting (`radarex2.php`)**  
[[example\\_src/radarex2.html](#)]



Each radar plot can also have a legend. As usual the text for the legend is created by using the method

- `RadarPlot::SetLegend($aText)`

The legend box is accessed through the "\$legend" property of the graph and this can be used to adjust the size, position and layout of the legend box (as described in Section 14.4, "Adjusting the position and layout of the legend").

## 16.2.2. Adding plot-marks to radar plots

In exactly the same way as for line graphs it is possible to add plot marks in radar plots. The mark property is accessed through the instance variable

- `RadarPlot::mark`

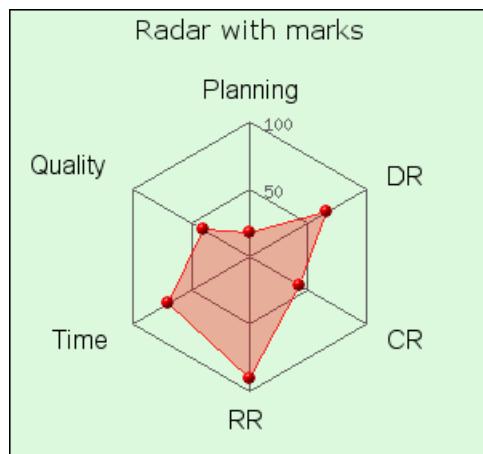
(All available plot marks are listed in Appendix E, *Available plot marks*)

For example to add a red ball marker the following line must be added

```
$radarplot->mark->SetType(MARK_IMG_SBALL, 'red');
```

In Figure 16.35, “Adding plot marks to a radar graph (`radarmarkex1.php`)” we show a radar graph with added plot-marks. In addition we have also chosen to hide the tick marks on the axis and adjust some of the default colors.

**Figure 16.35. Adding plot marks to a radar graph (`radarmarkex1.php`) [`example_src/radarmarkex1.html`]**



### 16.2.3. Client Side Image maps

(See Chapter 10, *Using CSIM (Client side image maps)* for a full description on the usage of CSIM together with the library)

If markers are shown for the polar plot (by setting the mark property of the plot) each marker can be a hot spot in a client side image map. The target URL are as usual specified with the `SetCSIMTargets()` as the following short code snippet shows

```
<?php
// Start by specifying the proper URL targets
$targets = array("#1" , "#2" ,);
$radarplot = new RadarPlot($data);
$radarplot->mark->SetType(MARK_SQUARE);
$radarplot->SetCSIMTargets($targets);
$graph->Add($radarplot);
$graph->StrokeCSIM();
?>
```

### 16.2.4. Adjusting the overall properties of the radar graph

The scale of the axis can be either a linear or a logarithmic scale and is specified with the method

- `RadarGraph::SetScale($aScale, $aYMin=1, $aYMax=1)`

The scale is specified as a string and can be either "lin" or "log"

The size and position of the radar graph are adjusted by the two methods

- `RadarGraph::SetSize($aSize)`

The size is specified as a fraction of min(\$width,\$height) and indicates the length of the axis.

- `RadarGraph::SetCenter($aXPos, $aYPos=0.5)`

The center of the graph can be specified as either a fraction of the width/height or as an absolute position.

The background color can be adjusted by

- `RadarGraph::SetColor($aColor)`

## 16.2.5. Adjusting the axis formatting

All axis have the same formatting and the axis is access through the "\$axis" property of the RadarGraph class. For example make the axis 2 pixels wide the following lines would be needed

```
$radargraph = new RadarGraph($width,$height);
$radargraph->axis->SetWeight(2);
```

The axis supports all the same formatting options as for standard x-y graphs. Some of the more commonly used methods for the axis formatting are

- `Axis::SetColor($aColor,$aLabelColor=false)`

Set the color of the axis and (optional) the labels

- `Axis::SetWeight($aWeight)`

Specifies width in pixels of the axis

- `Axis::SetLabelFormatString($aFormStr,$aIsDateFormat=false)`

Specify the format string to be used for the labels

- `Axis::SetLabelFormatCallback($aFuncName)`

Set the callback function to be used for the label formatting

- `Axis::SetFont($aFamily,$aStyle,$aSize)`

Sets the font property for the labels of the axis. Note that this will **not** effect the font property of the title of the axis.

In order to specify the titles for each axis the method

- `RadarGraph::SetTitle($aTitleArray)`

The argument is an array with as many entries as there are axis and where each entry specifies a text string which is the title of the axis.

### Example 16.4. Having the name of the months as title of the axis

The easiest way to have the month name as titles is to use the library global variable "\$gDateLocale" as follows

```
$titles = $gDateLocale->GetShortMonth();
$radargraph->SetTitle($titles);
```

In order to adjust the property of the title for the axis the "\$title" property of the axis is used. For example, to set the font and color for the titles of the axis the following two lines are needed

```
<?php
$radargraph->axis->title->SetFont(FF_ARIAL,FS_BOLD,12);
$radargraph->axis->title->SetColor('darkred');
?>
```

The tick marks can also be adjusted with the following methods

- RadarGraph::HideTickMarks(\$aFlag=true)
  - Hide all tick marks
- RadarGraph::ShowMinorTickmarks(\$aFlag=true)
  - Enable/Disable tick marks
- RadarGraph::SetTickDensity(\$aDensity=TICKD\_NORMAL)
  - Adjust the tick density

## 16.2.6. Adjusting grid lines for the radar graph

Each major tick mark can also be connected together to create a grid in the graph. The grid is accessed through the '\$grid' property of the graph. To enable the grid and set the line style to "dotted" the following two lines must be added

```
$radargraph->grid->Show();
$radargraph->grid->SetLineStyle('dashed');
```

An example of dashed grid lines are shown in Figure 16.36, “Enabling a dashed grid line (radarex4.php) ”. By default the grid lines have a gray color and are drawn behind the radar plot and hence part of the grid lines are obscured.

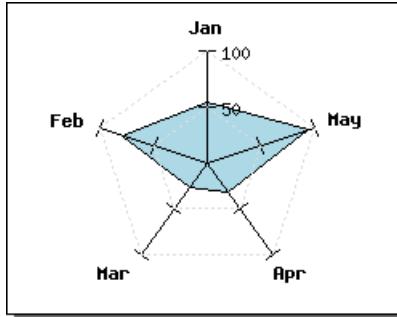
The property of the grid lines are adjusted by accessing the "\$grid" line. The available methods to format grid lines are:

- RadarGraph::grid::Show(\$aShowMajor=true)
  - Enable grid lines
- RadarGraph::grid::SetColor(\$aMajColor)
  - Set the color of the grid lines
- RadarGraph::grid::SetWeight(\$aWeight)
  - Set the weight of the grid line
- RadarGraph::grid::SetLineStyle(\$aType)
  - Set the line style, can be one of "dotted","dashed","long-dash","solid"

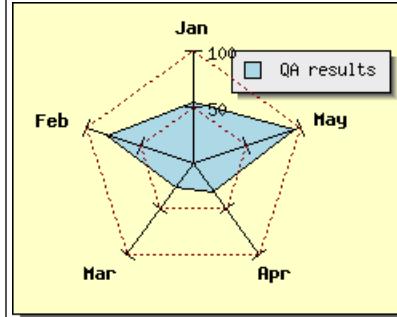
To make the grid lines in Figure 16.36, “Enabling a dashed grid line (radarex4.php) ” more visible lets change the color by using the SetColor() method on the grid. In addition we can also adjust the

background color. The result of this is shown in Figure 16.37, “Enabling a dashed grid line with red (radarex6.php)”

**Figure 16.36. Enabling a dashed grid line (radarex4.php) [example\_src/radarex4.html]**



**Figure 16.37. Enabling a dashed grid line with red (radarex6.php) [example\_src/radarex6.html]**



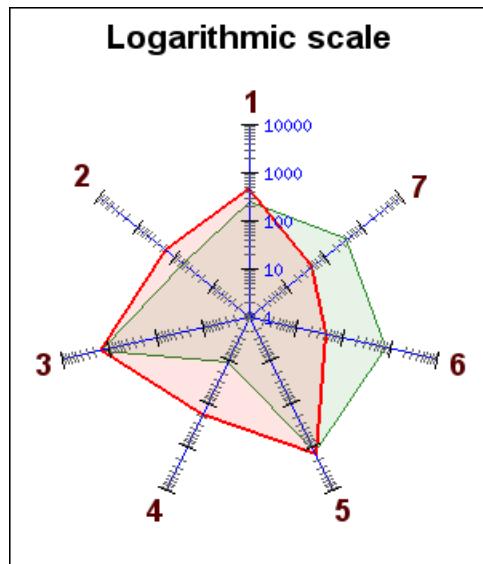
## 16.2.7. Using a logarithmic scale

In order to change the (default) linear scale of the axis to use logarithmic scale the following line must be added:

```
$graph->SetScale('log');
```

Figure 16.38, “Using a logarithmic scale (radarlogex1.php)” shows an example of a logarithmic radar graph

**Figure 16.38. Using a logarithmic scale (radarlogex1.php) [example\_src/radarlogex1.html]**



## 16.2.8. Enabling anti-aliasing for radar graphs

Radar graphs are an excellent example where enabling anti-alias can make a large visual difference. In order to enable anti-aliasing the method

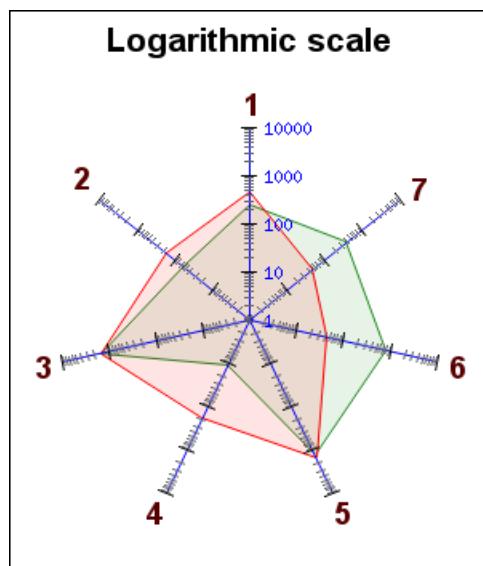
- RadarGraph::img::SetAntiAliasing(\$aFlg=true)

must be called, for example as in

```
$radargraph->img->SetAntiAliasing();
```

To give an example of this Figure 16.39, “Enabling anti-alias for the logarithmic radar example (radarlogex1-aa.php) ” shows an anti-aliased version of Figure 16.38, “Using a logarithmic scale (radarlogex1.php) ”. (Another comparative example of enabling anti-aliasing is also given in Figure 14.80, “Plain radar plot (radarex8.php) ” and Figure 14.81, “Anti-aliased radar plot (radarex8.1.php) ”.)

**Figure 16.39. Enabling anti-alias for the logarithmic radar example (radarlogex1-aa.php) [example\_src/radarlogex1-aa.html]**



## Caution

Remember the limitation when using anti-aliasing that lines will be given line weight=1 regardless of the specified weight.

### 16.2.9. A final example

As a final example we shown a radar graphs were we have adjusted many of the available properties, such as specifying a manual scale, to achieve the result shown in Figure 16.40, “A more complex example of a radar graph with a manual scale (fixscale\_radarex1.php) ”

**Example 16.5. A more complex example of a radar graph with a manual scale  
(fixscale\_radarex1.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_radar.php');

$graph = new RadarGraph(300,300);
$graph->SetScale('lin',0,50);
$graph->yscale->ticks->Set(25,5);
$graph->SetColor('white');
$graph->SetShadow();

$graph->SetCenter(0.5,0.55);

$graph->axis->SetFont(FF_FONT1,FS_BOLD);
$graph->axis->SetWeight(2);

// Uncomment the following lines to also show grid lines.
$graph->grid->SetLineStyle('dashed');
$graph->grid->SetColor('navy@0.5');
$graph->grid->Show();

$graph->ShowMinorTickMarks();

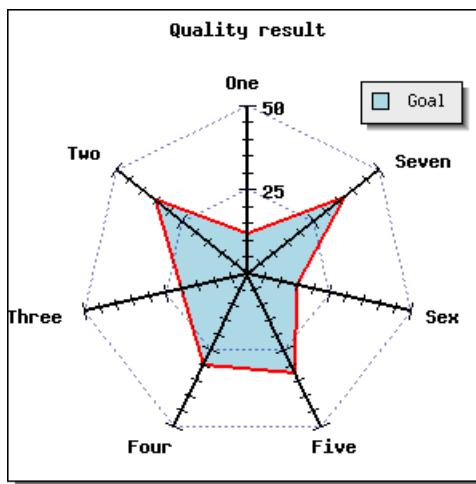
$graph->title->Set('Quality result');
$graph->title->SetFont(FF_FONT1,FS_BOLD);
$graph->SetTitles(array('One','Two','Three','Four','Five','Six','Seven','Eight','Nine'));

$pplot = new RadarPlot(array(12,35,20,30,33,15,37));
$pplot->SetLegend('Goal');
$pplot->SetColor('red','lightred');
$pplot->SetFillColor('lightblue');
$pplot->SetLineWeight(2);

$graph->Add($plot);
$graph->Stroke();

?>
```

**Figure 16.40.** A more complex example of a radar graph with a manual scale (`fixscale_radarex1.php`) [example\_src/  
`fixscale_radarex1.html`]



## 16.3. Polar graphs

In order to create Radar graphs the module "`jpgraph_polar.php`" must first be included.

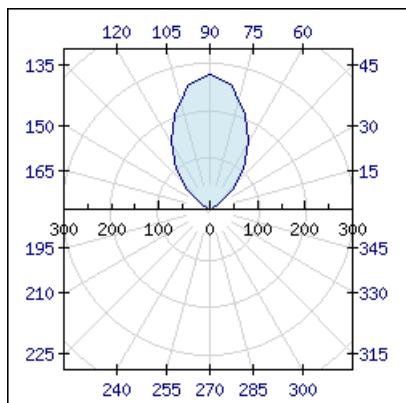
Each data point in a polar plot is represented by a tuple consisting of a radius and an angle. The polar plot itself can be either outlined or filled. In addition each point may have a standard marker (the same as for line and scatter plots).

The scale for the radius can be either linear or logarithmic.

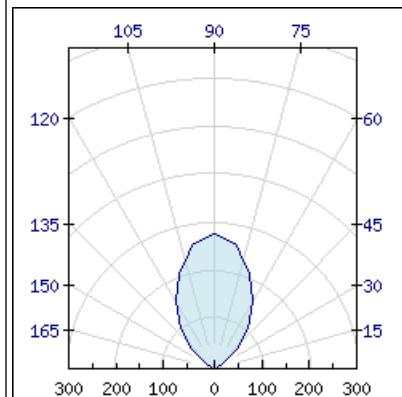
A polar graph is created by creating an instance of `class PolarGraph`. The polar graph type inherits all the capabilities of ordinary x,y graphs, i.e they can have background images, background gradients, formatted titles, using tabbed titles etc.

There are two types of polar graphs, full 360 degree view or just 180 degree view. The figures below show the difference between these two graph types.

**Figure 16.41.** A full 360 degree polar graph (polarex0.php) [example\_src/polarex0.html]



**Figure 16.42.** A 180 degree (half) polar graph (polarex0-180.php) [example\_src/polarex0-180.html]



The choice is controlled with the method

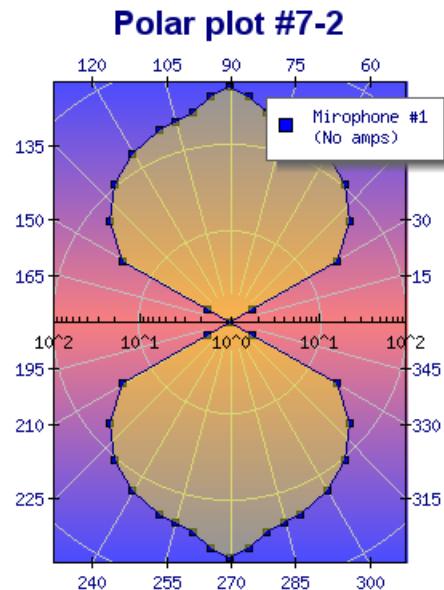
- `PolarGraph::SetType($aType)`

\$aType is specified with the symbolic defines

- `POLAR_360`, The default
- `POLAR_180`

As an example of using some more of the available formatting that is discussed in later sections Figure 16.43, “A 360 polar plot with background gradient and alpha blending (polarex7-2.php)” shown a full 360 degree plot with both background gradient as well as using alpha blending for the fill of the polar plot.

**Figure 16.43.** A 360 polar plot with background gradient and alpha blending (`polarex7-2.php`) [[example\\_src/polarex7-2.html](#)]



### 16.3.1. Rotating the polar graph

#### Caution

There is a bug in versions before v3.0.4 so that a rotated polar graph must always have equal margins on all sides to work as expected.

By default the start angle (`angle=0`) is located at the 3'o clock position. By calling the method `Graph::Set90AndMargin()` it is possible to rotate the graph so that the 0-angle position is at the 6'o clock position (straight down). This is mostly used for full 360 degree graphs.

### 16.3.2. Changing the angle direction

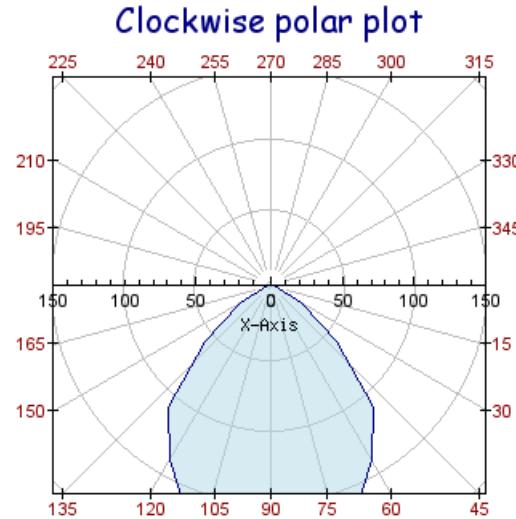
#### Note

This is only available in v3.0.4 of the library and above.

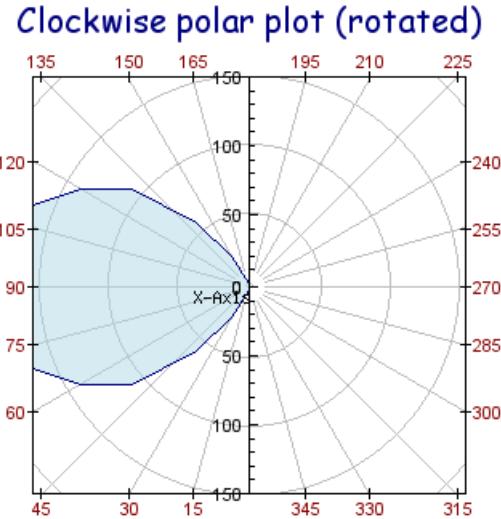
By default the angle is counted counter-clockwise as is custom in most applications of polar plots. However, it is also possible to have the angles counted clockwise. This is done by calling the method `Graph::SetClockwise()`.

In the following two examples we show a full 360 degree polar plots with clockwise ordering in both a standard configuration and also as a 90 degrees rotated graph.

**Figure 16.44. Clockwise polar graph (polarclockex1.php)**  
 [example\_src/  
 polarclockex1.html]



**Figure 16.45. Rotated clockwise polar graph (polarclockex2.php)**  
 [example\_src/  
 polarclockex2.html]



### 16.3.3. Adding polar plots to a polar graph

The principle is exactly the same as for line plots but instead of an x-y coordinate each point is represented by a angle and a radius to specify its position. The data is given in the format

```
$data = array(angle1, radius1, angle2, radius2, ...)
```

Creation of a single polar plot follows the now familiar structure as shown in the following code snippet

```
<?php
...
// Setup the graph
$graph = new PolarGraph($width,$height);

// Set a logarithmic scale with 100 as the maximum value, i.e. 10^2
$graph->SetScale('log',100);

// Use a full 360 degree polar graph
$graph->SetType(POLAR_360);

// Create a plot with marks
$polarplot = new PolarPlot($data);
$polarplot->mark->SetType(MARK_SQUARE);

// Add it to the graph
$graph->Add($polarplot);
...
?>
```

Adding multiple polar plots follows the same pattern. Each polar plot is created as an instance of class `PolarPlot` and then added to the graph.

### 16.3.4. Adding plot-marks to polar plots

In exactly the same way as for line graphs it is possible to add plot marks in radar plots. The mark property is accessed through the instance variable

- `PolarPlot::mark`

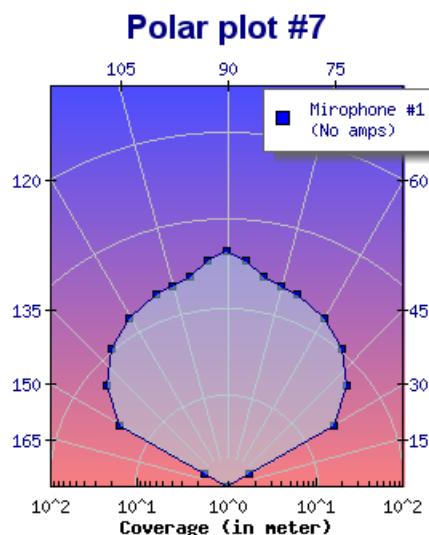
(All available plot marks are listed in Appendix E, *Available plot marks*)

For example to add a square marker (with the default color) the following line must be added

```
$polarplot->mark->SetType(MARK_SQUARE);
```

In Figure 16.46, “Adding plot marks to a polar graph (`polarex7-1.php`)” we show a polar graph with added plot-marks. In addition we have added a gradient background.

**Figure 16.46. Adding plot marks to a polar graph (`polarex7-1.php`)**  
[[example\\_src/polarex7-1.html](#)]



### 16.3.5. Client Side Image maps

(See Chapter 10, *Using CSIM (Client side image maps)* for a full description on the usage of CSIM together with the library)

If markers are shown for the polar plot (by setting the mark property of the plot) each marker can be a hot spot in a client side image map. The target URL are as usual specified with the `SetCSIMTargets()` as the following short code snippet shows

```
<?php
// Start by specifying the proper URL targets
$targets = array("#1" , "#2" ,);
$polarplot = new PolarPlot($data);
$polarplot->mark->SetType(MARK_SQUARE);
$polarplot->SetCSIMTargets(targets);
```

```
$graph->Add($polarplot);
$graph->StrokeCSIM();
?>
```

### 16.3.6. Adjusting the radius scale

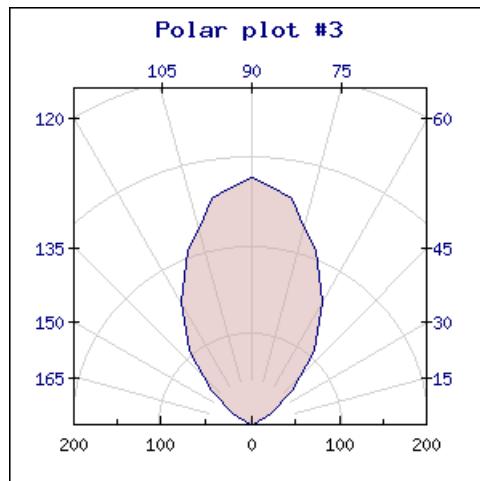
The radius axis can be shown in either a linear or logarithmic scale. This is controlled by a call to

- `PolarGraph::SetScale($aScale,$aRadiusMax=0)`

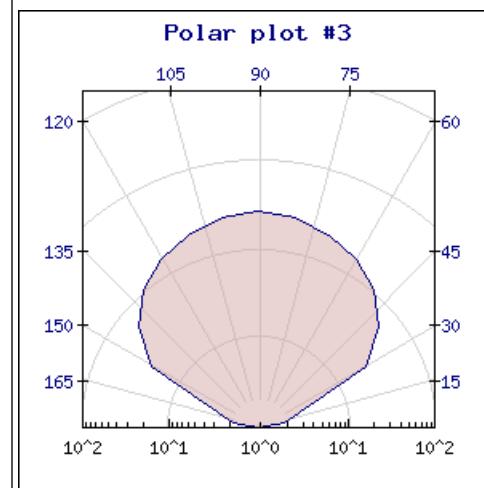
Supported scale types for `$aScale` are the strings "lin" or "log". The second argument is an optional manual setting of the maximum scale value for the radius.

The two examples below show the same plot in either linear or logarithmic scale for the radius

**Figure 16.47. Linear scale for radius (polarex3-lin.php) [example\_src/polarex3-lin.html]**



**Figure 16.48. Logarithmic scale for radius (polarex3.php) [example\_src/polarex3.html]**



Please note that the maximum values of the scales are different.

By default the scale will be auto scaled depending on the data. It is also possible to specify a manual scale by supplying an extra argument to the `SetScale()` method.

The only difference from the manual scaling with the standard x-y-graphs is that for polar graph only the maximum value is manually specified. The minimum will always be 0 for the linear scale and a scaled value of 10 (i.e 1, 0.1, 0.001 and so on) for the logarithmic scale.

#### Note

The plot is always clipped to the plot area.

### 16.3.7. Adjusting the grid lines

The graph allows several formatting option for the grid lines. For polar plots there are two types of grid lines, the angle and the radius grid lines.

Grid lines format is controlled by

- `PolarAxis::ShowGrid($aMajor=true,$aMinor=false,$aAngle=true)`

Decide if the major,minor and angle grid line should be displayed

- `PolarAxis::SetGridColor($aMajorColor,  
$aMinorColor=' ', $aAngleColor=' ')`

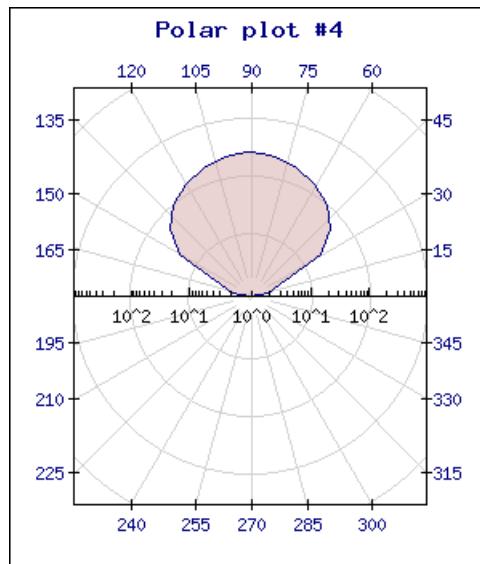
Specify the color for the major,minor and angel grid

- `PolarAxis::SetAngleStep($aStep)`

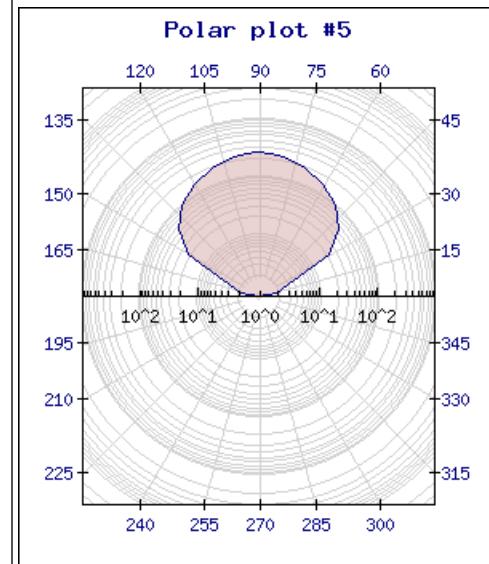
This specifies the angle distance between two consecutive angel scale marks in degrees. By default the step size is 15 degrees.

The two example below shows a logarithmic plot with either just major grid lines or both minor and major grid lines.

**Figure 16.49. Logarithmic scale with only major grid lines (polarex4.php) [example\_src/polarex4.html]**



**Figure 16.50. Logarithmic scale with both major and minor grid lines (polarex5.php) [example\_src/polarex5.html]**



### 16.3.8. Adjusting the labels

It is possible to individually specify different fonts and colors for the angle and the radius labels.

- `PolarAxis::SetFont()`

Adjust the radius font

- `PolarAxis::SetAngleFont()`

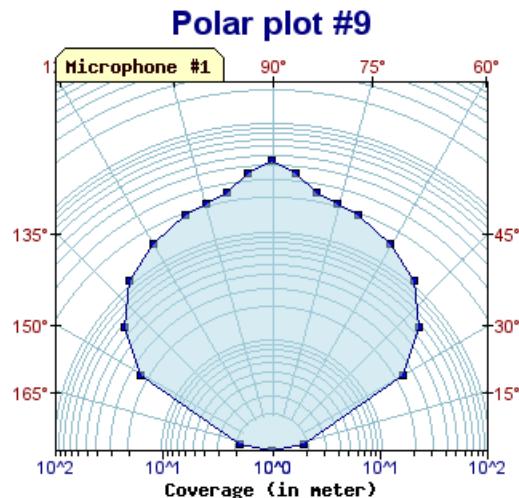
Adjust the radius font

- `PolarAxis::SetColor()`

Adjust the radius color

The following example specifies different color for the labels. it also shows how to add both a radial axis title as well as a tabbed title. In this example we have also chosen not to show the frame around the edge of the plot.

**Figure 16.51. Different colors for labels, specifying both a tabbed title as well as a axis title (`polarex9.php`) [[example\\_src/polarex9.html](#)]**



### Note

As can be seen from the previous examples the angle labels have a degree mark "°" by default if the font used is a TTF font (the degree symbol is not available for bitmap fonts) after each label. It is possible to select if this degree mark should be displayed or not with a call to the method `PolarAxis::SetAngleDegreeMark()`.

### Note

For the radius labels all standard formatting that can be done to the x-y axis such as format string or format callbacks are supported.

### Note

A common modification for polar plots is to disable the display of the last label when using a 360 degree plot since the last label will "collide" with the plot box around the plot area. It is possible to disable the last label with a call to `Axis::HideLastTickLabel()`. As can be see this has been used in some of the examples in this chapter.

## 16.4. Gantt charts

Gantt charts are used to give an easy overview of the extension in time of one or several activities (possibly grouped). In addition the gantt chart can show an ordinal relation between one or several activities such as "*activity A needs to be finished before activity B can start*".

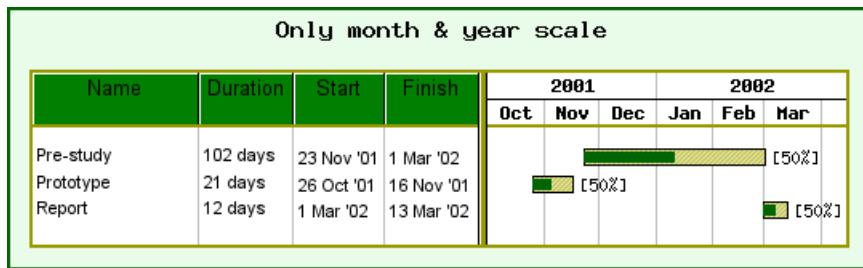
The shape of each activity can also be adjusted as well as color and size. Some of the capabilities of the Gantt module in the library are listed below.

- Overall gantt graph features

- Visualization of constraints between activities, start-to-start, start-to-end, end-to-start and end-to-end
  - Unlimited number of activities (up to memory and time constrains of PHP)
  - Full support for independent CSIM for both labels and activity bars
  - Support for visualization of grouped activities
  - Gantt charts can be automatically sized according to the number of bars and scale used.
  - Supports title and subtitle with user specified font, size and color
  - Supports vertical marker lines with text
  - Full support for CSIM (or drill down graphs)
  - Alternate row colors
- **Scale features**
    - Both automatic and fully automatic scale
    - Flexible scale with options to display up to 6 lines of scale headers (year,month,week,day,hour,minute)
    - Supports platform independent Week number calculation according to ISO:8601
    - Scales grids are intelligent not to overwrite smaller resolution scales
    - Scale headers can be localized
    - Full user configurable scales
    - Each scale header is configurable in terms of font, size, color, background, grid lines etc.
  - **Activity bars**
    - can have multiple patterns and colors
    - can have progress indicators
    - can have drop shadow
    - titles can have individual fonts, colors and backgrounds
    - can have captions
    - can have specified left- and right end markers
    - heights can be specified in absolute pixels or in percent of the activity line width
  - **Milestones**
    - can have user selectable shape and color
    - titles can have individual fonts, colors and backgrounds
    - can have captions

An example of a small Gantt chart is shown in Figure 16.52, “A typical small Gantt chart ([ganttmonthyearex2.php](#))”

Figure 16.52. A typical small Gantt chart (`ganttmonthyearex2.php`)  
 [example\_src/ganttmonthyearex2.html]



The remainder of this section will be used to discuss most of the formatting options for Gantt charts.

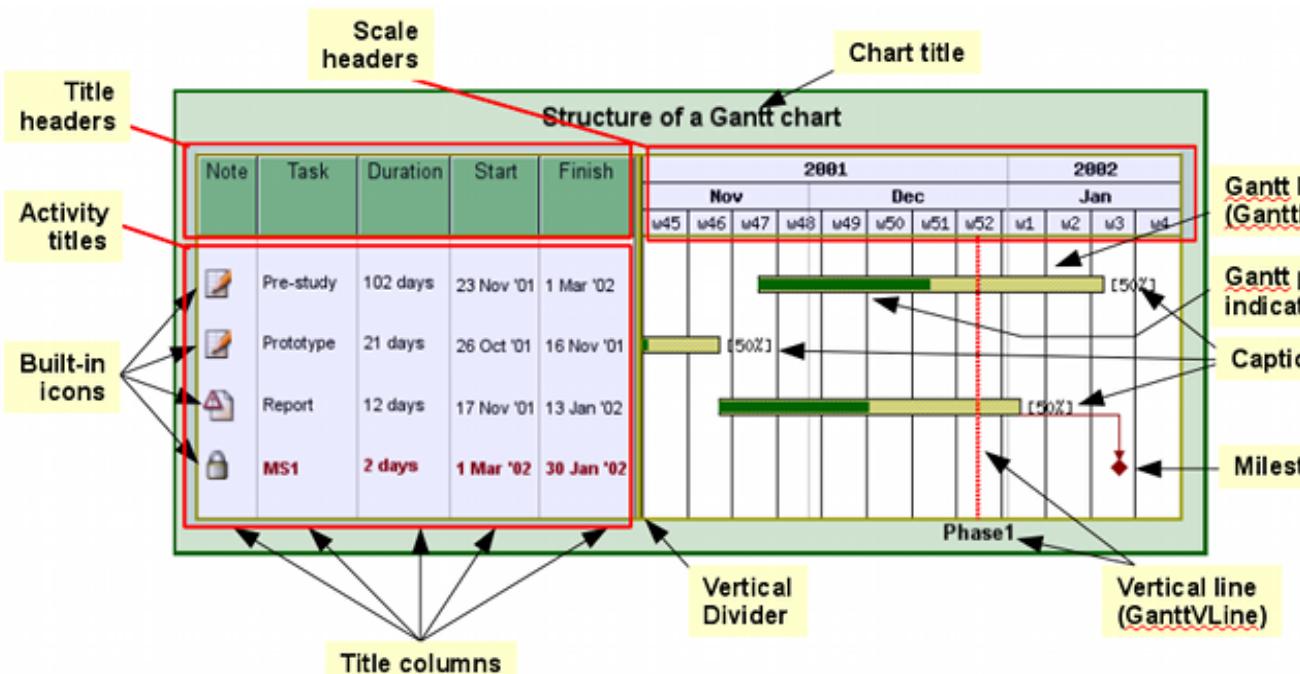
## Note

There is one restriction of the date scale and that is that it cannot have any "holes" the dates must be continuous.

#### **16.4.1. The structure of a Gantt chart**

To understand the terminology used for Gantt chart Figure 16.53, “Building block of a Gantt chart” shows a typical chart with indications of the name of each main building block of the chart.

**Figure 16.53.** Building block of a Gantt chart



In addition to the specific Gantt formatting that will be discussed in the following sections all the previously explained graph embellishment like the options of adding texts, icons, changing font and color of titles, adding footers etc. are also available for Gantt charts.

A Gantt chart is made up of four distinct areas:

1. On the left side there is the activity title column.
2. On the top there is the scale headers (up to six headers may be displayed)
3. The actual plot area where all the activity Gantt bars and markers are placed
4. The margin area, where for example the titles are shown

The steps to crate a Gantt charts is similar to creating a cartesian plot. First an instance of the main graph canvas is created (as an instance of class `GanttGraph`) and then one or more "plots" are created and added to the graph. For Gantt graph the "plots" that can be added are typically

- Activity bars (we will also use the name Gantt bars as synonym) that indicates the length (and possible progress) of one activity. An activity bar is created as an instance of class `GanttBar`
- Milestone marks, this can be thought of as a special activity bar with length = 0 and is often used to indicate a milestone or a deadline in the Gantt chart. A milestone is created as an instance of class `Milestone`
- A phase divider, this is a vertical line that can be added at specific dates and is often used to mark the end and beginning of phases in a project. A divider is created as an instance of class `GanttVLine` (for Gantt Vertical Line)
- A background pattern for a specific date range, this is often used to indicate holidays, public holidays or periods of special interest in the project
- An arbitrary icon, this is either a predefined image or one of the built in markers in the library. This is created as an instance of class `IconImage`

All these objects may be extensively modified in terms of formatting., colors (both fill- and frame color), size, titles, style and patterns etc. All objects have basic default values so it is not strictly speaking necessary to adjust them. However, the basic default values will give the charts a very simple look.

## 16.4.2. Creating a Gantt graph

In order to create a Gantt graph the module "`jpgraph_gantt.php`" must be included together with the core module "`jpgraph.php`"

A Gantt graph is created as an instance of class `GanttGraph` which inherits much of the same formatting options available for standard x-y graphs, for example titles, backgrounds, adding icons, adding texts and so on.

There is one crucial difference between all the other graph types and Gantt charts and that is the fact that for all other graphs both the height and width of the graph must be specified. For Gantt graphs this is not true.

Gantt graphs can be either

- fully automatically sized
- have the width specified but the height automatically determined by the number of activities added to the graph
- fully specified with both width and height

This means that all the following gantt graph creations are valid

```
<?php
// Fully automatic
$ganttgraph = new GanttGraph();

// Semi automatic (automatically determined height)
$ganttgraph = new GanttGraph(800);

// Fully manual
$ganttgraph = new GanttGraph(800,500);
?>
```

### Note

If the specified width and height is too small to have room for all the activities specified the activities will be clipped to the specified date range.

The creation of a full Gantt graph follows the now familiar pattern of

1. Create the graph (as shown above) and specify the overall formatting options (e.g. titles, colors etc)
2. Create the plot objects that should be added to the graph (e.g. activity bars, milestones, icons, texts etc) and format them as wanted
3. Add the object to the graph, with a call to `GanttGraph::Add()`, i.e. the activities, milestones etc.
4. Send the graph back to the client or save it directly to a file, with a call to `GanttGraph::Stroke()`

### Note

Even though there is no limit for the size of the Gantt chart (apart from available memory and execution time limit specified in "php.ini") the library defines two constants `MAX_GANTTIMG_SIZE_W` and `MAX_GANTTIMG_SIZE_H` that sets the maximum allowed image size for a Gantt chart. This is primarily meant to discover scripts gone wrong that tries to make very large images (perhaps by some non-properly terminating loops). If the overall image size becomes larger than these limit an error message will be shown.

## 16.4.3. Adjusting the scale headers

A gantt chart must always have at least one scale header and at most six scale headers (see Figure 16.53, “Building block of a Gantt chart”). Usually it is not advisable to use more than three scale headers at the same time. Having multiple scale headers allow the specification of the same date range but with different resolutions.

The library makes six different scale headers available which is identified by a symbolic constant that is logically combined to specify the wanted headers using the method

- `GanttGraph::ShowHeaders($aHeaderSelection)`

The `$aHeaderSelection` argument is a binary combination of one or more of the following specifiers that indicates the interval used in the header

1. `GANTT_HMIN`, Minute interval header (See the section called “Minute scale” for label format options)
2. `GANTT_HHOUR`, Hour interval header (See the section called “Hour scale” for label format options)

3. GANTT\_HDAY, Day interval header (See the section called “Day scale” for label format options)
4. GANTT\_HWEEK, Week interval header (See the section called “Week scale” for label format options)
5. GANTT\_HMONTH, Month interval header (See the section called “Month scale” for label format options)
6. GANTT\_HYEAR, Year interval header (See the section called “Year scale” for label format options)

For example to show a year, month and week header the following line would be used

```
$graph->ShowHeaders(GANTT_HYEAR | GANTT_HMONTH | GANTT_HWEEK);
```

Any combination of the listed headers above can be used. The scale headers will always be drawn with the larger header rage on top of a header with smaller range.

Scale headers week, day, hour , minute have a minimum span of 1 unit. This means that if, for example, the week header is displayed the minimum width of the overall scale is one week.

## Caution

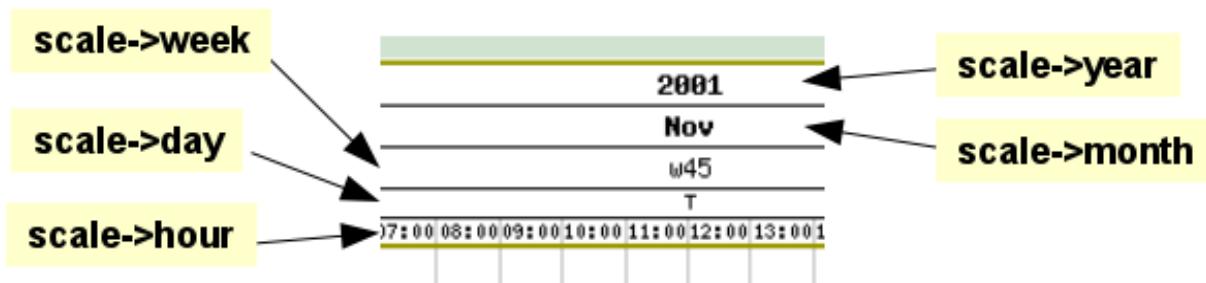
The overall minimum size of the scale regardless of what scale headers are displayed is one day.  
This means that it is not possible to just create a Gantt graph with , say 6 hour.

Specifying the wanted headers is the first step in controlling the header. The next step is to specify the format that should be used to print the date label in each of the selected headers. For example the week interval header will have a label at the start of every week. This could for example be indicated as week number, day of month, full date and so on. In order to adjust the headers the appropriate header instance variable must be access. The header instance variables are

1. Graph::scale::minute
2. Graph::scale::hour
3. Graph::scale::day
4. Graph::scale::week
5. Graph::scale::month
6. Graph::scale::year

The connection with the scale headers are shown in Figure 16.54, “The Gantt scale properties” which shows a cut out part of a larger gantt chart

**Figure 16.54. The Gantt scale properties**



All the headers are an instance of class `HeaderProperty` and supports the following formatting methods

- `HeaderProperty::SetFont($aFontFamily,$aFontSize,$aFontStyle)`

Specify the font to be used for the label

- `HeaderProperty::SetTextColor($aColor)`

Specify the font color to use

- `HeaderProperty::SetStyle($aStyle)`

The style depends on the actual header and all available styles for each header are shown below. The style specifies the format used for the scale header label

- `HeaderProperty::SetBackgroundColor($aColor)`

Set the header background color

- `HeaderProperty::SetFrameWeight($aWeight)`

Specify the weight of the frame around the scale header

- `HeaderProperty::SetTitleVertMargin($aMargin)`

Specifies the margin between this header and the next header (usually there is no need to adjust this)

- `HeaderProperty::SetInterval($aInterval)`

Specifies the interval between each scale label. For example specifying

```
$graph->scale->hour->SetInterval(6);
```

Will step the hour label 6 hours for each label

So for example to set the font of the month header the following line would be used

```
$ganttgraph->scale->month->SetFont(FF_ARIAL,FS_NORMAL,10);
```

In addition to these methods each scale also has the property '`$grid`' which determines the appearance of grid lines for that specific scale. It is possible to adjust the appearance of the grid lines by the "normal" line methods, i.e.

- `SetColor($aColor)`, Set the grid color
- `SetWeight($aWeight)`, Set the grid line weight
- `SetStyle($aLineStyle)`, Set the grid line style, i.e. "solid", "dotted", "dashed", "long-dashed"
- `Show($aFlg=true)`, Enable the grid line

So for example to enable the week grid line and set it to red color the following lines would be needed

```
$graph->scale->week->grid->Show();
$graph->scale->week->grid->SetColor('red');
```

The automatic grid lines have some "intelligence" so that higher resolution scales will not cut through part ways of scale headers with lower resolution (such as a year grid line cutting through the middle of a week).

## Tip

It is possible to specify a zoom factor for the scale that adjusts how wide the automatic sized header should be. See the section called "Adjusting the scale zoom factor" for more details.

## Minute scale

Minute scale is enabled by adding the GANTT\_HMIN in the `GanttGraph::ShowHeaders()` call, for example as the following line shows

```
$graph -> ShowHeaders (GANTT_HDAY | GANTT_HHOUR | GANTT_HMIN);
```

The `SetStyle($aStyle)` method supports the following label styles

- MINUTESTYLE\_MM

This will display minutes as a two digit number with a leading zero if necessary

- MINUTESTYLE\_CUSTOM

This will let you specify your own custom minute style by making a call to `HeaderProperty::SetFormatString()`

The format string is specified as a format string for the `date()` function (See PHP Manual [<http://php.net/manual/en/function.date.php>])

## Hour scale

Minute scale is enabled by adding the GANTT\_HHOUR in the `GanttGraph::ShowHeaders()` call, for example as the following line shows

```
$graph -> ShowHeaders (GANTT_HDAY | GANTT_HHOUR | GANTT_HMIN);
```

The `SetStyle($aStyle)` method supports the following label styles

- HOURSTYLE\_HM24

Will display the only the hour in military time 0-24 , for example 13:00

- HOURSTYLE\_H24

Will display the hour with both hour and minute in military time 0-24, for example 13

- HOURSTYLE\_HMAMPM

Will display the hour and minutes with a suitable am/pm postfix, for example 1:30pm

- HOURSTYLE\_HAMPM

Will display only the hour with a suitable am/pm postfix, for example 1pm

- HOURSTYLE\_CUSTOM

Custom defined format as specified with a call to `HeaderProperty::SetFormatString()`

The format string is specified as a format string for the `date()` function (See PHP Manual [<http://php.net/manual/en/function.date.php>])

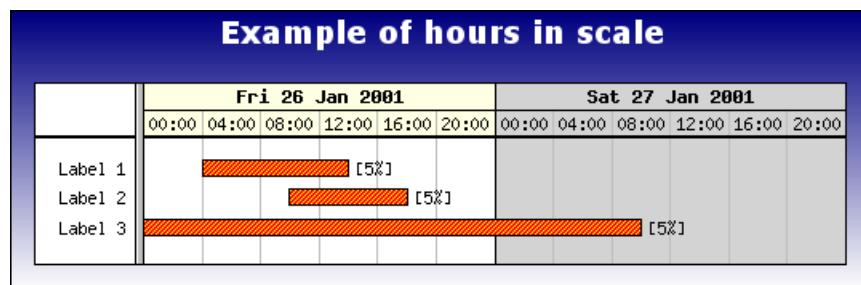
For hours it is possible to specify the interval in either of two ways. With an integer, e.g. 6, or as time interval, e.g. "1:30" which makes the interval one and a half hour. The only restriction is that the interval must be a divisor of 24 since one day is the smallest possible interval to show. This means that it is allowed to use, for example 2,4,6,"1:30" or "0:45" as intervals but not 7, "2:45".

The code snippet below shows how to set up a hour scale to with 45 minutes interval and custom colors

```
<?php
$graph->scale->hour->SetBackgroundColor('lightyellow:1.5');
$graph->scale->hour->SetFont(FF_FONT1);
$graph->scale->hour->SetStyle(HOURSTYLE_HMAMPM);
$graph->scale->hour->SetIntervall('0:45');
?>
```

The example in Figure 16.55, “Gantt chart with day and hour scale enabled (`gantthourex1.php`)” below shows a gantt chart with the day and hour scale enabled. In this example we have also added a gradient background to show some formatting options.

**Figure 16.55. Gantt chart with day and hour scale enabled (`gantthourex1.php`)** [[example\\_src/gantthourex1.html](#)]



## Day scale

The `SetStyle($aStyle)` method supports the following label styles

- `DAYSTYLE_ONELETTER`,

On letter week day. Example "M"

- `DAYSTYLE_LONG`,

Full week day. Example "Monday"

- `DAYSTYLE_LONGDAYDATE1`,

Day with date. Example "Monday 23 Jun"

- `DAYSTYLE_LONGDAYDATE2`,

Day with date+year. Example "Monday 23 Jun 2003"

- DAYSTYLE\_SHORT,  
Short date. Example "Mon"
- DAYSTYLE\_SHORTDAYDATE1,  
Short date+date. Example "Mon 23/6"
- DAYSTYLE\_SHORTDAYDATE2,  
Short date+date. Example "Mon 23 Jun"
- DAYSTYLE\_SHORTDAYDATE3,  
Short date+date. Example "Mon 23"
- DAYSTYLE\_SHORTDATE1,  
Short date. Example "23/6"
- DAYSTYLE\_SHORTDATE2,  
Short date. Example "23 Jun"
- DAYSTYLE\_SHORTDATE3,  
Short date. Example "Mon 23"
- DAYSTYLE\_SHORTDATE4,  
Short date. Example "23"
- DAYSTYLE\_CUSTOM,  
Custom specified formatting string. Example "%A"

The format string is specified as a format string for the `strftime()` function (See PHP Manual [<http://php.net/manual/en/function.strftime.php>])

## Caution

Note that the day format string is a format string for `strftime()` while the format string for hour and minutes are given as a format string for the `date()` function. This inconsistency purely exists for historic reasons and is kept not to break existing scripts.

### Example:

```
$graph->scale->day->SetStyle (DAYSTYLE_LONG);
```

The graphical formatting possibilities for days allow the possibility to specify a different color for the weekend background and also for the Sunday.

- `HeaderProperty::SetWeekendColor($aColor)`  
Set the background color for weekends. (Defaults to light gray)
- `HeaderProperty::SetSundayFontColor($aColor)`

The Sunday font color. (Defaults to red)

In addition to this there is also a possibility to choose whether or not the weekend background should be extended vertically down over the plot area which is the default. Since that is a property more of the whole plot this behavior is modified with a call to the method

- `GanttScale::UseWeekendBackground($aFlg=true)`

**Example:**

```
// Don't use background for weekend in the plot area
$graph->scale->UseWeekendBackground(false);
```

## Note

The actual text displayed is dependent on the Locale used. See Section 16.4.12, “Localizing the Gantt chart scale”

## Week scale

Week scales, if enabled, by default shows the week number in range 1 to 53 (as defined by ISO-8601)

## Note

It might be worth pointing out here that the week number calculation is carried out within the library and does not rely on the underlying OS date libraries. This makes the behavior consistent over several OS:s (at least MS Windows does not comply to ISO-8601 or supply any way of doing this through the normal libraries, e.g. `strftime()`)

The `SetStyle($aStyle)` method supports the following label styles

- `WEEKSTYLE_WNBR`

Show week number. To further modify the formatting of the actual week number it is possible to optionally supply a format string with a call to

- `HeaderProperty::SetLabelFormatString($aFormat)`

The format for this string is the same format used for the `sprintf()` function and formats the week number given as an integer.

## Caution

Note that the day format string is a format string for `sprintf()` while the format string for hour and minutes are given as a format string for the `date()` function. This inconsistency purely exists for historic reasons and is kept not to break existing scripts.

- `WEEKSTYLE_FIRSTDAY`

Show date of first day in week.

- `WEEKSTYLE_FIRSTDAY2`

Show date of first day in week and short month

- WEEKSTYLE\_FIRSTDAYWNBR  
Show week number of first day in week.
- WEEKSTYLE\_FIRSTDAY2WNBR  
Show week number of first day in week and month

**Example:**

```
$graph->scale->week->SetStyle(WEEKSTYLE_FIRSTDAY);
```

**Note**

The actual text displayed for month is dependent on the Locale used. See Section 16.4.12, “Localizing the Gantt chart scale”

## Month scale

The `SetStyle($aStyle)` method supports the following label styles

- MONTHSTYLE\_SHORTNAME  
Display the month name in its locale specific short form, i.e Jan, Feb etc
- MONTHSTYLE\_SHORTNAMEYEAR2  
Display the month name in its locale specific short form together with a 2 digit year , i.e Jan '01, Feb '01 etc
- MONTHSTYLE\_SHORTNAMEYEAR4  
Display the month name in its locale specific short form together with a 4 digit year , i.e Jan 2001, Feb 2001 etc
- MONTHSTYLE\_LONGNAME  
Display the month name in its locale specific long name, i.e. January, February
- MONTHSTYLE\_LONGNAMEYEAR2  
Display the month name in its locale specific long name together with a 2 digit year , i.e January '01, February '01 etc
- MONTHSTYLE\_LONGNAMEYEAR4  
Display the month name in its locale specific long name together with a 4 digit year , i.e January 2001, February 2001 etc
- MONTHSTYLE\_FIRSTLETTER  
The first letter of the month name

**Example:**

```
$graph->scale->month->SetStyle(MONTHSTYLE_LONGNAME);
```

## Year scale

Year scale has no extra formatting possibilities it is always displayed as a four digit number , e.g. "2009"

### 16.4.4. Adding gantt objects to the chart

Gantt objects are primarily instances of one of two classes

1. class GanttBar

This is the main activity added to a Gantt chart to show the extension in time of one activity, a gantt bar

2. class Milestone

This activity is a special case of an activity with zero extension, usually indicating a milestone or a deadline

3. class VLine

This is similar so a millstone as it has no extension in time. Instead this object is visually represented as a vertical line that crosses the entire graph plot area. This is often used to mark, for example, phases in a project.

To add a Gantt objects there are two compulsory parameters that must be set. These parameters specify on what row and at what date the activity should start at.

Bars and Milestones need both a vertical position and a horizontal position. The horizontal start position is specified as a date, e.g. "2001-06-23", and the vertical positions are specified as a number [0,1,2,3,...]. This vertical number indicates the position from the top where the object should be placed. To understand this one could imagine a number of "invisible" horizontal bands with a certain height. If the vertical position is specifies as 0 the bar will be placed in the first band, specify 3 and the bar will be placed in the fourth band and so on.

All these "invisible bands" have the same height (equ-spaced). The height of each band is automatically determined and depends on both the method of layout (as specified by (GanttChart::SetLayout( )) and the individual heights of the individual bars and titles. The rules are quite simple:

1. If layout=GANTT\_FROMTOP (the default and most common) the height will equal the height (+ a margin) of the highest gantt bar. The height calculation of each bar takes into account both the actual bar, the title, and any left- right-marks (more about that later) that may be present.

The name "fromtop" refers to that when the height is explicitly specified the bars will usually be added from band 0 and onwards and hence being added from the top. (This might leave empty space at the bottom of the plot area in the graph if the height of the graph has been explicitly specified).

2. If layout=GANTT\_EVEN the bars are evenly (hence the name) spread out over the available height in the gantt chart and no consideration is taken of the individual bars heights. Note that if you use automatic sizing even layout cannot be used. (It just doesn't make sense). Even layout is for those cases when a large area is specified and the bars should be evenly distributed using the full height.

So in summary each object must have two position parameters.

1. **Which row the gantt object should be drawn on.**

This should be a positive integer in the range (0, ...), It is perfectly legal to specify a large vertical position with no other object above as shown in Figure 16.58, "Specifying a large vertical position (ganttex03.php)"

## 2. Which start and end date (or dates) the object should have.

Start of bars are given as a date string. The format depends on the current locale. Examples of valid date strings are

- "2001-10-22"
- "2001-10-22"
- "22 Oct 2001"

Even if several format are supported it is recommended to use all numeric dates, i.e in the form "2001-10-22".

## Caution

Watch our for the locale used since, for example, "2008-05-10" can have two meanings.

Specifying the end position may be done in two different ways, either by the end date in the same way as for the start date. The other way is to specify the length of the activity in number of days (and fractions thereof). Examples of valid end dates are:

- "2001-11-15"
- "15 Nov 2001"
- 22, (specifies duration of 22 days)
- 22.7, (specifies duration of 22.7 days)

Please note that duration must be specified as numerical values and not as a string.

Usually at least one or more of the following parameter is also specified

1. The title of the gantt object
2. The caption of the object. This is a text string that is drawn beside the object on the gantt chart
3. Color and any optional patterns to separate different activities
4. When applicable the state of a progress indicator (see the section called “Adding progress indicators”)

## Adding Gantt activity bars

Instances of class `GanttBar` is created with the constructor

- `GanttBar::__construct($aPos, $aLabel, $aStart, $aEnd, $aCaption= "", $aHeightFactor=0.6)`  
\$aVPos The vertical position for the bar, [0..n]  
\$aTitle Title for the activity  
\$aStart Start date for the activity given as string, e.g "2001-09-22"  
\$aEnd End date for activity given as either a date (a string) or as the duration (in days) of the activity, e.g both "2001-10-15" and 20.5 are valid inputs

\$aCaption Text string (caption) to appear at the end (right side) of the bar

\$aHeight Height of bar given as either a value in range [0,1] in which case this is interpreted as what fraction of the vertical position should the bar occupy. The height can also be given in absolute pixels [1..200]

In order to illustrate this we will create a the most basic (and simple ) Gantt chart possible. This will consist of just a chart with one activity bar.

### Example 16.6. The simplest possible Gantt graph (`ganttex00.php`)

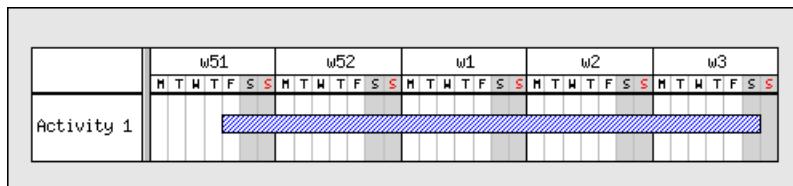
```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_gantt.php");

// A new graph with automatic size
$graph = new GanttGraph();

// A new activity on row '0'
$activity = new GanttBar(0,"Activity 1","2001-12-21","2002-01-19");
$graph->Add($activity);

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

**Figure 16.56. The simplest possible Gantt graph (`ganttex00.php`)**  
**[`example_src/ganttex00.html`]**



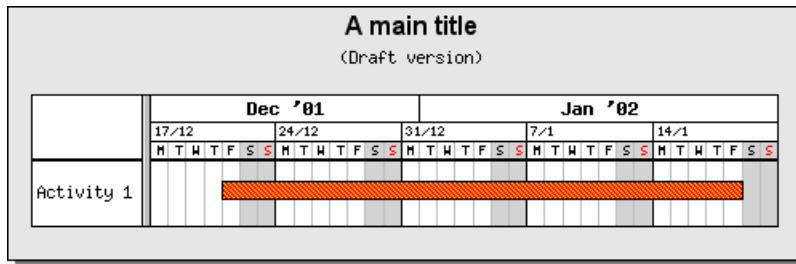
The example above will make use of just the default settings for all formatting parameters but still manage to create a perfectly readable Gantt chart with only 4 lines of real code. We can note a couple of things

- By default two scale headers are used, week and day resolution
- Weekends will have a gray background
- Sunday scale header uses red for the "Sunday"

Lets now take the above simple graph and make a few small alterations.

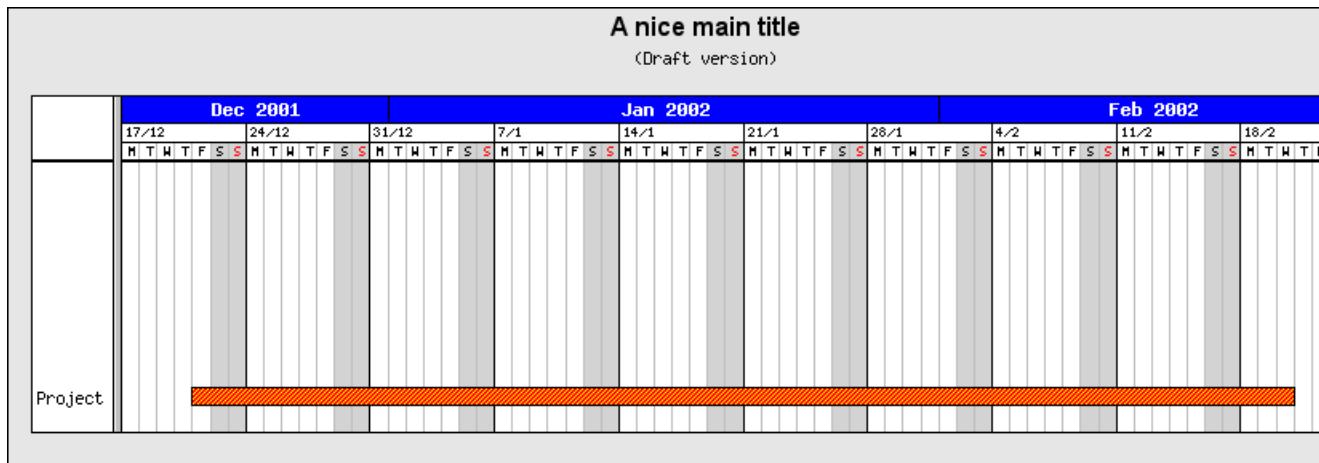
- We will add graph titles
- We will change the pattern and color of the activity bar
- We will adjust the scale headers so er have three headers and use the date of the start day for each week (in the week scale)

**Figure 16.57. Making some minor alterations to the Gantt graph (`ganttex01.php`) [`example_src/ganttex01.html`]**



To show the effect of row positioning Figure 16.58, “Specifying a large vertical position (`ganttex03.php`)” shows the effect of adding a bar to row 7. This will as can be seen create graph that is mostly empty sine row 0-6 have no specified gantt objects.

**Figure 16.58. Specifying a large vertical position (`ganttex03.php`) [`example_src/ganttex03.html`]**



The height of the bars can also be adjusted. The horizontal spacing between each bar is based on the highest single bar including the size of the title. By default the height of the bar is 60% of the overall vertical size allocated to each row (all the rows have the same height).

The height of the bar can be specified as either as an absolute number of pixels or as a fraction of the row height. Since by default the bar height is 60% this means that if any single line has, for example, a large title all the rows will be adjusted to the same size and hence the bars will also be adjusted to fill 60% of the new width.

## Adding milestones

Instances of class `Milestone` is created with the constructor

- `MileStone::__construct($aVPos, $aLabel, $aDate, $aCaption = '')`

`$aVPos`, The vertical position for the bar, [0..n]

`$aTitle`, Title for the activity

`$aDate`, Date for the milestone

\$aCaption, Text to the right of the milestone

Valid milestones are for example

```
$ms4 = new MileStone(3, 'Code complete', '2001-12-01');
$ms5 = new MileStone(3, 'Ready for test', '2002-01-06');
```

By default milestones are rendered as a filled "Diamond" shape. This may be optionally modified. The actual shape is specified by the 'mark' property of milestone which is an instance of class PlotMark. See the section called "Adding markers to the Gantt bars".

To change the shape of a milestone to, say a triangle, you use the SetType() method as in

```
$ms4->mark->SetType(MARK_DTRIANGLE);
```

An example of adding a milestone to a gantt graph is shown in Figure 16.59, "Adding a milestone marker to a gantt graph (ganttex04.php)"

**Figure 16.59. Adding a milestone marker to a gantt graph (ganttex04.php)**  
[\[example\\_src/ganttex04.html\]](#)



We note that;

- By default the title of a milestone is set in a red color. To change the title the "\$title" property of the milestone must be accessed. For example

```
$milestone->title->SetColor('black');
```

- A milestone has a caption in exactly the same way as an ordinary gantt bar
- Milestones are added to the graph with the usual GanttGraph::Add()

## Adding vertical lines (GanttVLine)

Instances of class GanttVLine is created with the constructor

- GanttVLine::\_\_construct(\$aDate,\$aTitle='',\$aColor='darkred', \$aWeight=2,\$aStyle='solid')

\$aDate Date for the milestone

\$aTitle Title for the line. The title is displayed at the bottom of the line

`$aColor` Color of the line

`$aWeight` Line width

`$aStyle` Line style, specified as a string "dashed", "dotted" and so on

By default a GanttVLine will cover the entire Gantt area from the top to the bottom. It is also possible to restrict the area that the line is spanning by specifying the start and stop row for the line with a call to the method

- `GanttVLine::SetRowSpan($aStart, $aEnd)`

If the end row is left out the line will go all the way to the bottom.

## Note

Since there is a small margin from the top header row to the first Gantt row this means that specifying a start row of 0 (the very first gantt row) will leave a small margin between the line and the header row. If this is not desirable the start row can be specified as -1 in which case the line will go all the way up to the header row.

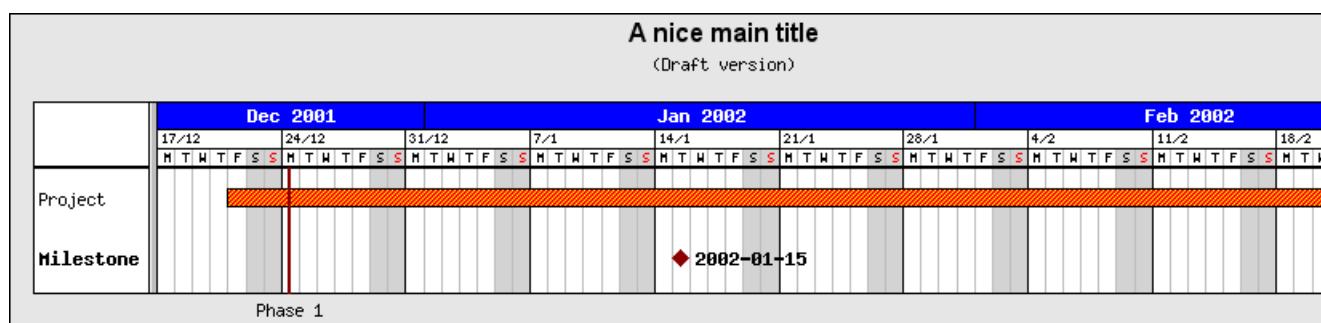
Valid creations of lines are for example

```
$vline1 = new GanttVLine('2001-12-24', 'Phase 1');
$vline2 = new GanttVLine('2001-12-28', 'Phase 1', 'darkred', 5, 'dotted');
$vline2->SetRowSpan(3); // Start at row index=3 (fourth row) and go all the way to the bottom
```

In Figure 16.60, “Adding a vertical line in the Gantt graph (`ganttex06.php`)” we have created a line with

```
$vline = new GanttVLine('2001-12-24', 'Phase 1');
$graph->Add($vline);
```

**Figure 16.60. Adding a vertical line in the Gantt graph (`ganttex06.php`) [[example\\_src/ganttex06.html](#)]**



We note the following

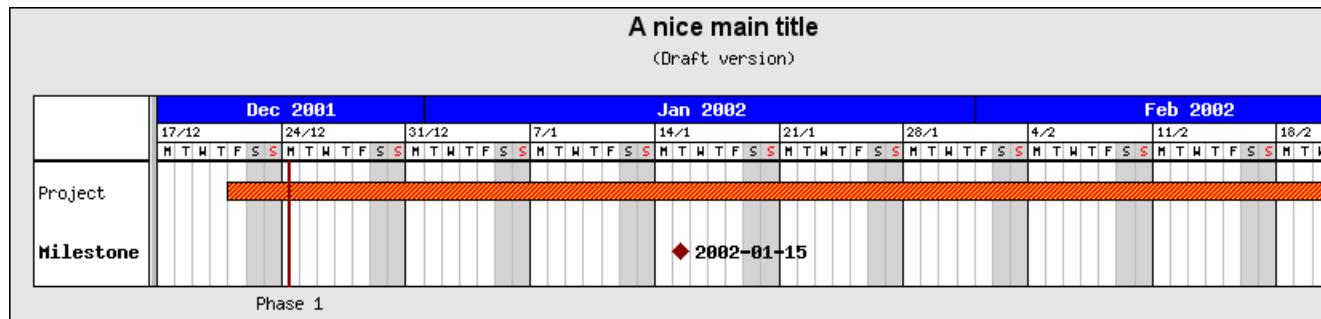
- The title of the vertical line is rendered at the bottom of the graph in the margin.
- By default the line is drawn at the beginning of the day of the specified date and in a "dashed" style. This can be modified so that the line is drawn/aligned anywhere in the specified day. This is done by using the method `GanttVLine::SetDayOffset()` with an argument specifying the fraction of the width of a single day where the line should be positioned.

For example, if we want to display the line in the middle of the day we need to add the line

```
$vline->SetDayOffset (0.5);
```

Adding this line to the example in Figure 16.60, “Adding a vertical line in the Gantt graph (ganttex06.php) ” gives the result shown in Figure 16.61, “Adjusting the position of the vertical line within the day (ganttex07.php) ”

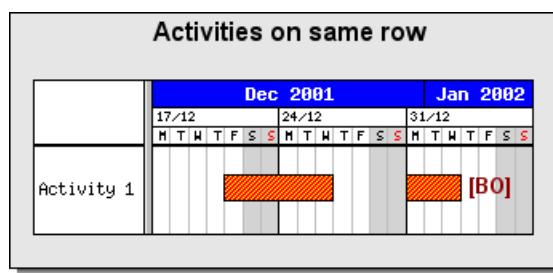
**Figure 16.61. Adjusting the position of the vertical line within the day (ganttex07.php) [example\_src/ganttex07.html]**



## Illustrating break in activities

Since the position of each activity is specified by row and date there is nothing that prevents adding multiple activity bars to the same row. This observation makes it possible to illustrate breaks in activities during a certain period. Figure 16.62, “Adding several activity bars on the same row (gantt\_samerowex1.php) ” shows an example of this

**Figure 16.62. Adding several activity bars on the same row (gantt\_samerowex1.php) [example\_src/gantt\_samerowex1.html]**



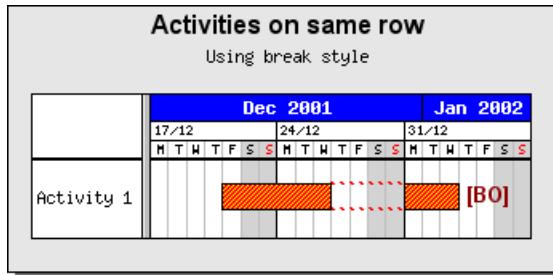
A common variant is to have some connection between the parts in the broken activity. This can be accomplished by using a special variant of the gantt activity available just for this purpose.

Gantt bars have the following method available

- `GanttBar::SetBreakStyle($aFlg=true, $aLineStyle='dotted', $aLineWidth=1)`

This will draw the gantt bar almost hollow. It will just consist of the top and bottom line of the bar. The idea to illustrate a break is to have a bar of this style during the break. Figure 16.63, “Adding a hollow “break” bar (gantt\_samerowex2.php) ” shows an example of this.

**Figure 16.63. Adding a hollow "break" bar (`gantt_samerowex2.php`) [`example_src/gantt_samerowex2.html`]**



## 16.4.5. Additional formatting for activity bars (Gantt bars)

### Adding markers to the Gantt bars

Each gantt bar can have a marker either at the left or to the right of the bar. The markers are accessed through the properties

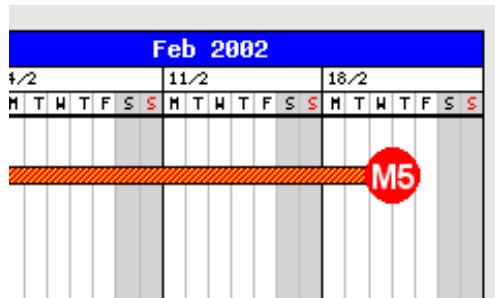
- `GanttBar::leftMark`
- `GanttBar::rightMark`

Since the markers are instances of class `PlotMark` all the features of this class is available as described in Section 15.1.3, “Adding marks to the plot (a.k.a. plot marks)” and in Appendix E, *Available plot marks*.

To give a practical example of the usage of marks we will add a solid filled circle with a title text to the right end of a activity bar by using the following lines

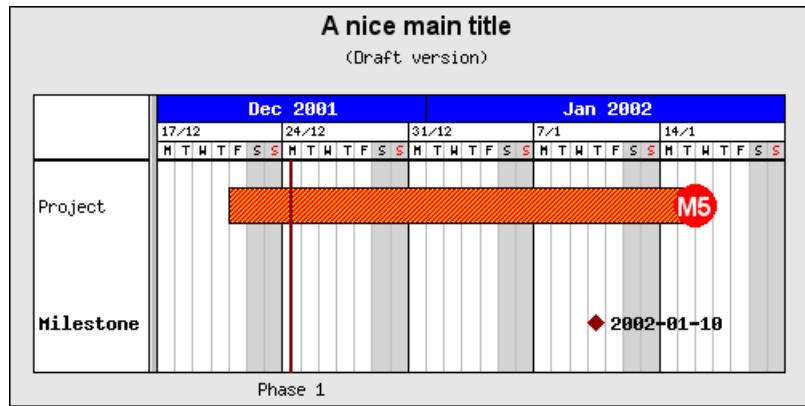
```
<?php
// Add a right marker
$activity->rightMark->Show();
$activity->rightMark->SetType(MARK_FILLEDIRCLE);
$activity->rightMark->SetWidth(13);
$activity->rightMark->SetColor('red');
$activity->rightMark->SetFillColor('red');
$activity->rightMark->title->Set('M5');
$activity->rightMark->title->SetFont(FF_ARIAL,FS_BOLD,12);
$activity->rightMark->title->SetColor('white');
?>
```

This would give the effect shown in Figure 16.64, “Example of adding a right marker to the activity bar”

**Figure 16.64. Example of adding a right marker to the activity bar**

## Caution

Since the bar height is 60% of the row height this means that a large marker will adjust the row size (since it must be large enough to make room for the mark) and it might be necessary to compensate for this by calling the `GanttBar::SetHeight($aHeight)`. Remember that the reserved height is the maximum height needed by any line. Figure 16.65, “A large marker will force the row to become larger since it by default always fills 60% of the allocated height for each row (`ganttex08.php`)” shows what happens with the bar height if it is not adjusted.

**Figure 16.65. A large marker will force the row to become larger since it by default always fills 60% of the allocated height for each row (`ganttex08.php`) [`example_src/ganttex08.html`]**

## Note

There are two special markers `MARK_LEFTTRIANGLE` and `MARK_RIGHTTRIANGLE` not normally available that are used to format the visual indication of a group bar header. See Section 16.4.7, “Grouping activities”

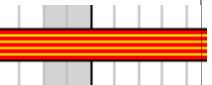
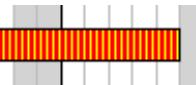
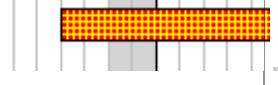
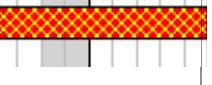
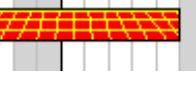
## Specifying a fill pattern for the activity bar

The pattern and color for an activity bar is specified with the method

- `GanttBar::function SetPattern($aPattern,$aColor="blue",$aDensity=95)`

The available patterns together with the symbolic names are given in Table 16.1, “Gantt bar patterns”.

**Table 16.1. Gantt bar patterns**

			
GANTT_LDIAG	GANTT_RDIAG	GANTT_HLINE	GANTT_VLINE
			
GANTT_SOLID	GANTT_HVCROSS	GANTT_DIAGCROSS	GANTT_3DPLANE

**Note**

In Table 16.1, “Gantt bar patterns”.we have used different value for the \$aDensity parameter to better show the pattern. Depending on the pattern we have used a density in the range 85 to 95.

**Adding captions to activities**

The caption text is a text string that is shown to the right of the activity bar. It can have different usages, for example a common use is to show the initials of the person (or persons) responsible for completing a specific activity.

The caption is accessed through the "\$caption" property of the bar. This property is an instance of the Text class and hence inherits all the common text formatting options.

The following line sets the caption to the string " [ AG ] "

```
<?php
$activity->caption->Set(' [AG] ')
$activity->caption->SetFont(FF_FONT2,FS_BOLD);
?>
```

Figure 16.66, “Adding a caption to a Gantt bar” shows a small cut out from a gantt chart that shows the typical appearance of a gantt bar caption.

**Figure 16.66. Adding a caption to a Gantt bar**

In addition to specifying the caption as shown above the caption text can also be specified directly when creating a gantt bar as the fifth parameter as the following example shows

```
$activity = new GanttBar(0, 'Activity 1', '2001-11-21', '2001-12-20', '[BS,ER]');
```

In order to specify the distance between the Gantt object and the caption text the method

- GanttPlotObject::SetCaptionMargin(\$aMargin)

is used. For example, to increase the margin to 20 pixels for a gantt bar the following line must be added

```
$activity->SetCaptionTitle(20);
```

## Adding progress indicators

To indicate the progress of a specific activity it is also possible to add a progress indicator to each bar. This progress indicator consists of a smaller bar within the bar. By default this progress bar is black and 70% of the height of the bar. These parameters can all be changed.

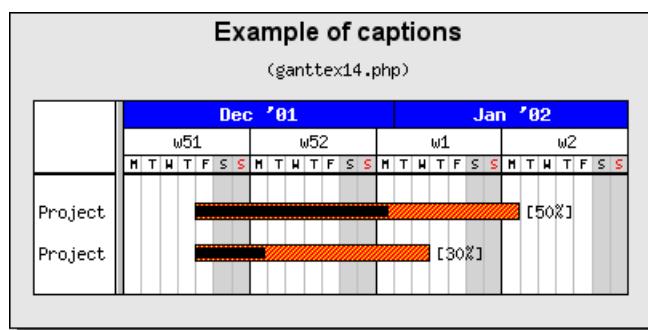
The properties for the progress indicator are accessed through the 'progress' property and its methods.

To set the progress for a specific activity you only specify the percent as a fraction (0-1). As in

```
$activity->progress->Set(0.4)
```

In each activity uses the default format for the progress indicator, a solid bar. To make it clearer we have also modified the caption to reflect the displayed progress. (At the same time we also modified the scale headers just to illustrate some more formatting options).

**Figure 16.67. Adding progress bars of the gantt chart (ganttex14.php) [example\_src/ganttex14.html]**



To specify a different format for the progress the following method is used

- Progress::SetPattern(\$aPattern,\$aColor="blue",\$aDensity=98)

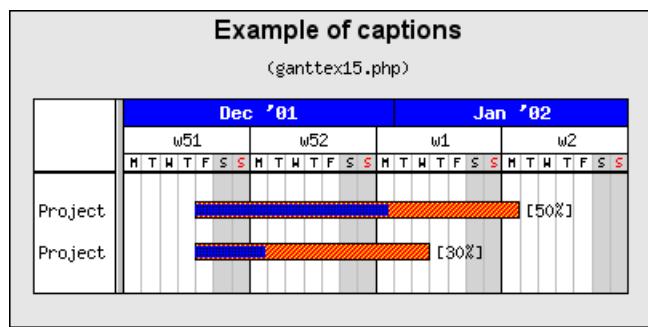
The parameters follow the same structure as patterns for the activity bars

We can now modify the progress bar above, for example by adding the line

```
$activity->progress->SetPattern(BAND_RDIAG , "blue");
```

and we then get the result shown in Figure 16.68, “Modifying the format for the progress pattern (ganttex15.php)”

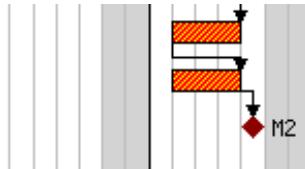
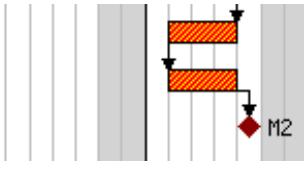
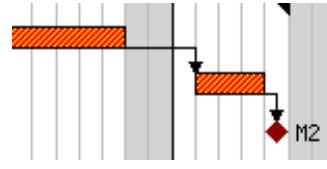
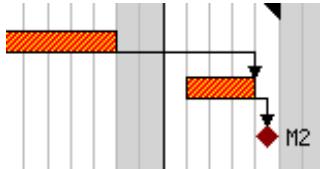
**Figure 16.68. Modifying the format for the progress pattern (ganttex15.php) [example\_src/ganttex15.html]**



## 16.4.6. Adding visual indication of constraints between gantt objects

With Gantt charts there is often the need to illustrate constraints between one or several activities. One of the most common constrain is that on activity can't start before some other activity has finished. The constrain is visualized as an arrow between two gantt objects.

The library supports visualization of the following four types of constraints

<b>Figure 16.69. Start to end constraint</b>  Start to End	<b>Figure 16.70. Start to start constraint</b>  Start to Start
<b>Figure 16.71. End to start constraint</b>  End to Start	<b>Figure 16.72. End to end constraint</b>  End to End

To visualize a constrain the method `SetConstrain()` is called on the first Gantt object that is the first part of the constraint. The signature for this method is

- `GanttObject::SetConstrain($aRow, $aType, $aColor='black', $aArrowSize=ARROW_S2, $aArrowType=ARROWWT_SOLID)`

`$aRow`, The target row for the constrain

`$aType`, The type of constraints. Can be one of the following symbolic defines

- `CONSTRAIN_STARTSTART`
- `CONSTRAIN_STARTEND`
- `CONSTRAIN_ENDSTART`
- `CONSTRAIN_ENDEDN`

`$aColor`, The color of the constraint arrow

`$aArrowSize`, The size of the constraint arrow. Can eb one of the following symbolic defines

- `ARROW_S1`
- `ARROW_S2`

- ARROW\_S3
- ARROW\_S4
- ARROW\_S5

\$aArrowType, specifies the visual appearance of the arrow. Can be one of the following symbolic defines

- ARROWT\_SOLID, Using a solid arrow and arrow head
- ARROWT\_OPEN, Using an outline arrow and outline arrow head

If we assume that the gantt chart have two activity bars (Gantt bars) defined as

```
$activity1 = new GanttBar(0,'First activity','2001-01-01','2001-03-01');
$activity2 = new GanttBar(1,'Second activity','2001-02-01','2001-04-15');
```

and that we need to visualize a end-to-start constraint from \$activity1 to \$activity2 we must add the following statement

```
$activity1->SetConstrain(1,CONSTRAIN_ENDSTART);
```

### Note

The actual path followed by the arrow is controlled by some heuristics to make it clear what the constrain is. It has been a design decision that in order to keep the API simple the user has no further detailed controlled on the actual path followed.

## 16.4.7. Grouping activities

It is common to group activities. There is no special type for activity bars that are used to illustrate grouping. The common way of illustrating this (as have been used above) is to add "half" a triangle marker at both ends of the bar.

The library provides two special types of marks that are handled slightly different than other markers just to cater for illustrating groups of activities. If the left or right marker (see the section on adding left and right markers) are of type MARK\_LEFTTRIANGLE or MARK\_RIGHTTRIANGLE those triangles will be drawn under the bars to give the effect show in the examples above.

It is also a good idea to make the grouping bars have slightly less height than normal activity bars since the end triangles will visually "grow" the bar. Remember that the size of the bar can be adjusted by adding an extra parameter in the creation of the bar and can be specified as either an absolute or as a relative (0-1) value, see the section called "Adding Gantt activity bars"

So to get the effect we want for a group bar we have to use the two lines

```
<?php
$groupbar->leftMark->SetType(MARK_LEFTTRIANGLE);
$groupbar->leftMark->Show();
$groupbar->rightMark->SetType(MARK_RIGHTTRIANGLE);
$groupbar->rightMark->Show();
?>
```

to add the typical group markers at the left and right end of an otherwise standard activity bar as can be seen in Figure 16.73, "Group markers"

**Figure 16.73. Group markers**



### Note

Since there is not yet any formatting support to accomplish the effect of indentation for the titles so this is accomplished by using a fixed width font and adding spaces in front of the title.

## 16.4.8. Simplifying creation of basic Gantt charts

As we have shown in the previous examples constructing a Gantt chart consists of a number of repetitive tasks; Create the individual activity bars, add possible constraints, format them and finally add them to the graph.

Since many basic Gantt charts that doesn't need a very high degree of customization and has an almost identical (but repetitive structure) the library offers a "wrapper" method that takes a structured specification of the activities and implements most of these repetitive tasks which will make the graphs scripts much simpler.

The only drawback is that in order to keep the specification simple (which is the whole purpose) there are very limited options to format the activities.

The wrapper method has the following signature

- `GanttGraph::function CreateSimple($aData, $aConstraints=array(), $aProgress=array())`
- `$aData`, Specification of activities described below
- `$aConstraints`, (optional) List of constraints
- `$aProgress`, (optional) List of progress values

The specification of the activities used for this wrapper method has the following structure

```
$spec = array(
 array(<act1_row>, <act1_type>, <act1_title>, <act1_start>, <act1_end>, <act1_>
 array(<act2_row>, <act2_type>, <act2_title>, <act2_start>, <act2_end>, <act2_>
 ...
 array(<actN_row>, <actN_type>, <actN_title>, <actN_start>, <actN_end>, <actN_>
```

Each array within the specification creates one activity. The fields have the following meaning

- `<act_row>`, Which row in the gantt chart this activity shall be drawn on
- `<act_type>`, What type of activity is this. The type is one of the following symbolic defines
  - `ACTYPE_NORMAL`, A standard activity bar
  - `ACTYPE_MILESTONE`, A milestone activity

- ACTYPE\_GROUP, Create a grouping activity bar by adding start and end group marks
- <act\_title>, The title for the activity, can be either a text string or an array to specify several title when several title columns have been specified
- <act\_start>, <act\_end>, Start and end date for the activity. For ACTYPE\_MILESTONE only the <act\_start> shall be specified.
- <act\_caption>, The optional caption text for the activity

The (optional) constraints specification has the following structure

```
$constraints = array(
 array(<act1_row>, <act2_row>, <type>),
 array(<act1_row>, <act2_row>, <type>),
 ...
 array(<act1_row>, <act2_row>, <type>));
```

- <act1\_row>, <act2\_row>, Start and end row for this constraint
- <type>, Type of constraint, as described in Section 16.4.6, “Adding visual indication of constraints between gantt objects”

Finally the (optional) progress specification has the structure

```
$progress = array(
 array(<act1_row>, <act1_progress>),
 array(<act2_row>, <act2_progress>),
 ...
 array(<act2_row>, <act2_progress>));
```

- <act\_row>, The row that has the activity with the specified progress
- <act\_progress>, The progress value for the activity at the specified row

This does perhaps not look like a simplification but the following example will show that it really is.

The following example will create a gantt graph with two activities, one milestone and all grouped together by a group bar. The specification for these activities will be

```
$data = array(
 array(0 , ACTYPE_GROUP , "Phase 1" , "2001-10-26" , "2001-11-23" ,
 array(1 , ACTYPE_NORMAL , "Label 2" , "2001-10-26" , "2001-11-13" ,
 array(2 , ACTYPE_NORMAL , "Label 3" , "2001-11-20" , "2001-11-22" ,
 array(3 , ACTYPE_MILESTONE , "Phase 1 Done" , "2001-11-23" , "M2"));
```

Note that we use spaces to get the titles indented according to the group structure.

We now only need to create a very basic gantt chart with some selected scale headers and perhaps also a title. We then call the `CreateSimple()` method with the data specification above as the argument and that is all we must do.

```
<?php
// Create a basic graph and set a title
$graph = new GanttGraph();
```

```

$graph->title->Set("Gantt Graph using CreateSimple() ");

// Setup a scale
$graph->ShowHeaders(GANTT_HYEAR | GANTT_HMONTH | GANTT_HDAY | GANTT_HWEEK);
$graph->scale->week->SetStyle(WEEKSTYLE_FIRSTDAY);

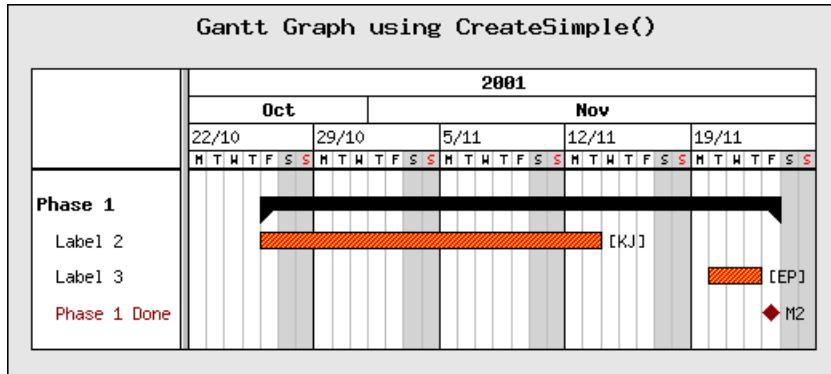
// Add the specified activities
$graph->CreateSimple ($data);

// .. and stroke the graph
$graph->Stroke ();
?>

```

Compare this to all the `$act = new GanttBar(...)` etc. we would have had to add and format using the "manual" method of adding activities to gantt graph. A complete example of this is shown in Figure 16.74, "Using the CreateSimple() wrapper method (`ganttsimpleex1.php`)"

**Figure 16.74. Using the CreateSimple() wrapper method (`ganttsimpleex1.php`) [example\_src/ganttsimpleex1.html]**



As we mentioned above the format options are very limited when using `CreateSimple()` since that is the whole idea. However, there are two methods to at least partially affect the format and the visual appearance of the activities.

- `GanttGraph::SetSimpleFont($aFont, $aSize)`
  - `$aFont`, Font family for titles
  - `$aSize`, Font size for titles
- `GanttGraph::SetSimpleStyle($aPattern, $aColor, $aFillColor)`
  - `$aPattern`, Specifies the pattern (if any) to use for filling the activity bars
  - `$aColor`, The pattern color
  - `$aFillColor`, The fill color of the activity bar

## 16.4.9. Using multiple title columns

It is often of interest not only to show one title for a gantt bar but often one wants to show, title, start date, end date, duration or effort and so on. Up until now we have, to keep things simple only shown a single

title for each activity. We will now show how to use an arbitrary number of columns/titles for each gantt activity (gantt bar or gantt milestone).

**Figure 16.75. Using multiple columns as titles for activities  
(ganttmonthyearch3.php)  
[example\_src/ganttmonthyearch3.html]**



To use multiple columns there are two steps needed.

1. The number of columns to be used as titles together with the headings and the headings properties like text, font and color are specified
2. The second step is to add the proper titles to each activity to be displayed in the specified columns

To set the columns the "Activity information" property of the scale must be accessed. To specify the headers of the title column the following method is used

- `ActivityInfo::SetColTitles($aHeaderArray, $aMinWidths)`

The following code excerpt shows an example on how to do this

```
$graph->scale->actinfo->SetColTitles(
 array('Note', 'Task', 'Duration', 'Start', 'Finish'),
 array(30 , 100));
```

Furthermore it is possible to modify the background colors and the style and colors of the vertical dividing grid lines. In the previous image we used the lines

```
<?php
$graph->scale->actinfo->SetBackgroundColor('green:0.5@0.5');
$graph->scale->actinfo->SetFont(FF_ARIAL , FS_NORMAL , 10);
$graph->scale->actinfo->vgrid->SetStyle ('solid');
$graph->scale->actinfo->vgrid->SetColor ('gray');
?>
```

The style for the grid lines is as usual specified with one of the strings be "solid","dashed","dotted" or "longdashed" as in other line formatting contexts within the library.

It is also possible to specify if a small "3D" effect should be used in the titles. By default this is enabled. You can easily turn this off with a call to

```
$graph->scale->actinfo->SetStyle(ACTINFO_2D);
```

To adjust the colors of the vertical dividing lines in the title the method `SetColor()` is used as in

```
$graph->scale->actinfo->SetColor('navy');
```

Once the format of the title column is set up the data in the columns (the texts) needs to be entered.

This is done when the activity bars are added to the graph. By default only a single column is used as a title and then the title of the activity is specified as a string. When multiple columns are used then the title of each activity is specified as an array of texts, each entry in the array corresponding to one column.

Specifying two column titles could be done with the following creation of the activity bar

```
$title1 = '...';
$title2 = '...';
$bar = new GanttBar(0 ,array($title1 , $title2) , "2003-11-23" , "2003-12-05");
```

Apart from specifying the titles as separate entries in an array there is actually one additional (older) way to specify the text that should go into each column and that is to separate each text string with a "\t" tab character. So for example the lines above could also have been written as

```
$title1 = '...';
$title2 = '...';
$bar = new GanttBar(0 , $title1 . "\t" . $title2 , "2003-11-23" , "2003-12-05");
```

Note the quote character used to encode the "\t" character. There is however a very crucial semantic difference between these two ways of specifying the texts and that is that with the method of tab characters the columns will not be automatically resized to fit the text string. This means that when the columns are setup they must have a big enough width to cater for all the strings.

## Adjusting the individual fonts for each column

*Note: This feature is only available in versions >= 3.0.3 of the library.*

When using the method `GanttBar::title::SetFont()` the same font will be applied to all columns for that particular activity. To specify individual fonts for each column the method `GanttBar::title::SetColumnFonts()` must be used.

This method accept an array of font arrays where each font array specifies the font family, style and size for that particular column. If fewer fonts specification than columns are given then the remaining columns will be using the default fonts specification (as defined by the `SetFont()` method).

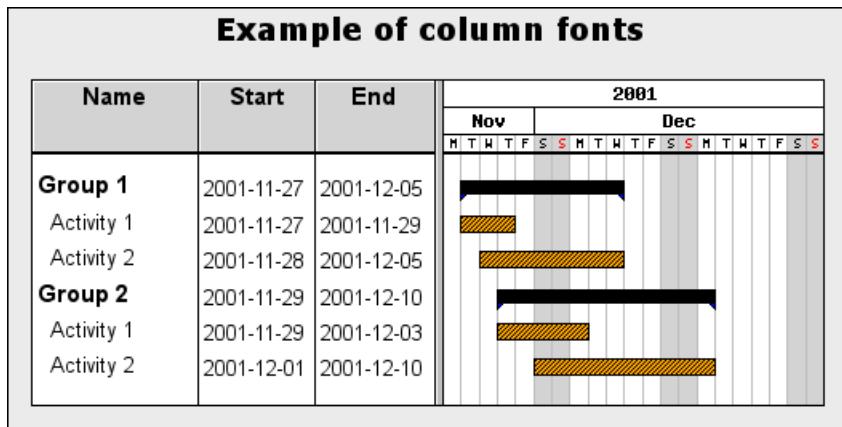
For example the two lines of code

```
<?php
$bar->title->SetFont(FF_ARIAL,FS_NORMAL,10);
$bar->title->SetColumnFonts(array(array(FF_ARIAL,FS_BOLD,11)));
?>
```

will cause the first column to be set in a bold font and the remaining columns falling back to the default non-bold font style.

A complete example on how to use this is shown in Figure 16.76, “Using different fonts for individual columns (ganttcolumnfontsex01.php)” below.

**Figure 16.76. Using different fonts for individual columns (ganttcolumnfontsex01.php) [example\_src/ganttcolumnfontsex01.html]**

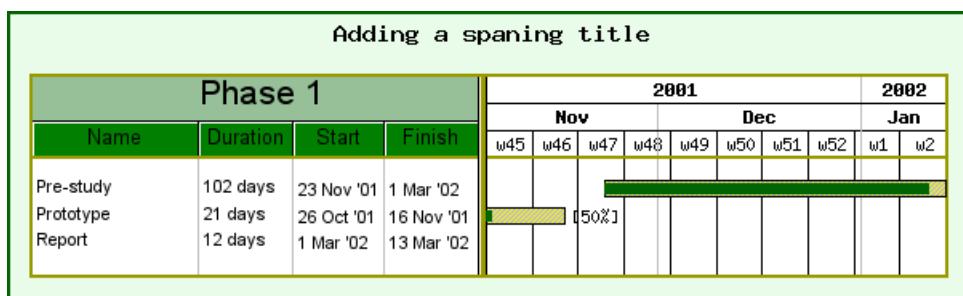


## Adding an overview title for all columns

It is possible to add a title that spans all the defined title columns. An example of this is shown in Figure 16.77, “Adding a spanning title over all title columns (ganttmonthlyarex4.php)”. This title is specified with the property “\$tableTitle” of the scale. Specifying a table title will automatically adjust the height of the column titles to fit the table title. The small code snippet below shows how to add a title.

```
<?php
$graph->scale->tableTitle->Set('Phase 1');
$graph->scale->tableTitle->SetFont(FF_ARIAL , FS_NORMAL , 12);
$graph->scale->SetTableTitleBackground('darkgreen@0.6');
$graph->scale->tableTitle->Show(true);
?>
```

**Figure 16.77. Adding a spanning title over all title columns (ganttmonthlyarex4.php) [example\_src/ganttmonthlyarex4.html]**



## Caution

Remember that the overall height available for both title column header and the spanning column title is limited to the height of the specified scale header. This means that having only a single scale header will not leave enough room to add a spanning title.

## 16.4.10. Built-in icons for use in activity titles

To assist in getting visual clues to how to interpret activities in the Gantt chart it is possible to add icons in the title columns. These icons can also act as hot-spots in CSIM graphs. The available built-in icons are listed in Figure 16.78, “Built-in icons for Gantt charts” together with there symbolic names.

In order to add an icon in a title column first an instance of `class IconPlot` is created and it is then added in exactly the same was as a text string. The signature for the constructor is

- `IconPlot::__construct($aIcon,$aScale=1)`

`$aIcon`, is either one of the symbolic names in Figure 16.78, “Built-in icons for Gantt charts” or a text string which is a filename of an arbitrary image to use as icon.

`$aScale`, is the initial scaling of the image

For example the following code snippet adds a "folder open image" in the first column and scaling it to be 60% of its original size

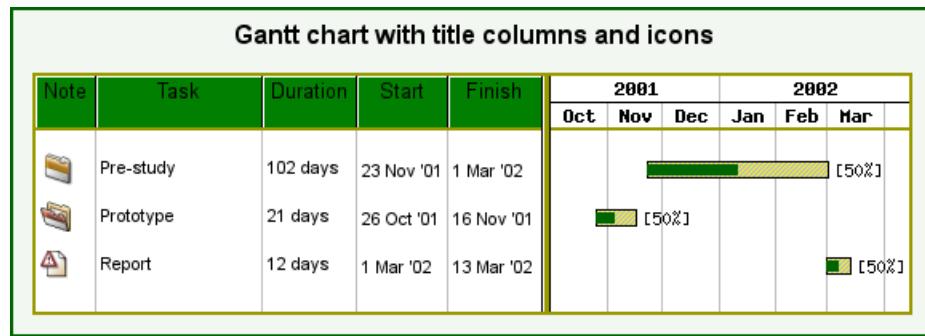
```
<?php
$iconopen = new IconImage (GICON_FOLDEROPEN , 0.6);
$title2 = '...';
$bar = new GanttBar (0 ,array($iconopen , $title2) , '2003-11-23' , '2003-12-05
?>
```

To instead use an arbitrary image as icon the code would have to be changed to

```
<?php
$iconopen = new IconImage ('myiconimage.jpg' , 0.6);
$title2 = '...';
$bar = new GanttBar (0 ,array($iconopen , $title2) , '2003-11-23' , '2003-12-05
?>
```

**Figure 16.78. Built-in icons for Gantt charts**

An example of using icons in the titles is shown in Figure 16.79, “Adding built in icons in titles (`gantticonex1.php`)”

**Figure 16.79. Adding built in icons in titles (`gantticonex1.php`) [`example_src/gantticonex1.html`]**

### 16.4.11. More general Gantt formatting

In this section we will show a few more ways by which you may customize the gantt chart itself. This include among other thing

- Adding a table title (not to be confused with the graph title)

- Adjusting appearance of the various lines in the bar chart
- Adjusting the zoom factor for the width when using automatic scaling

## Adjusting the scale zoom factor

By default the scale will be just wide enough to fit the chosen scale headers with some small margins on each side. It is possible to adjust the width when using automatic graph sizing by setting a zoom factor for the scale. The default width corresponds to a zoom factor of 1.0.

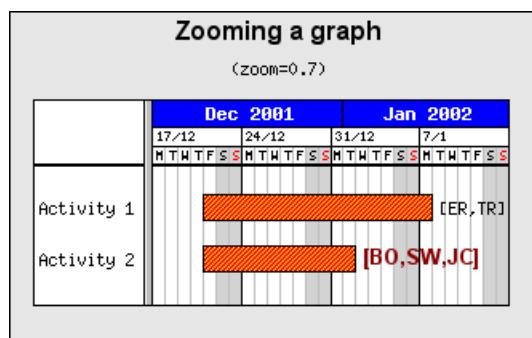
The zoom factor is adjusted by the method

- `GanttGraph::SetZoomFactor($aZoomFactor)`

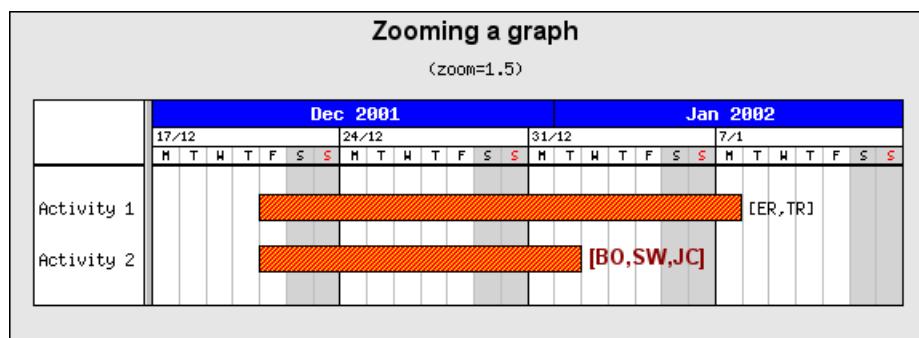
`$aZoomFactor`, A real number that specifies the zoom of the date scale. A zoom factor of 2.0 will double the default horizontal spacing.

In the following two examples the same Gantt chart is first shown with a scale factor of 0.7 (=70% of the original width) and in Figure 16.81, “A zoom factor of 1.5 (ganttex13-zoom2.php) ” a zoom factor of 1.5 (=150%) is used.

**Figure 16.80. A zoom factor of 0.7 (ganttex13-zoom1.php) [example\_src/ganttex13-zoom1.html]**



**Figure 16.81. A zoom factor of 1.5 (ganttex13-zoom2.php) [example\_src/ganttex13-zoom2.html]**



### Note

There is no limit for the zoom factor but from a practical point of view the useful range is [0.5,3.0]

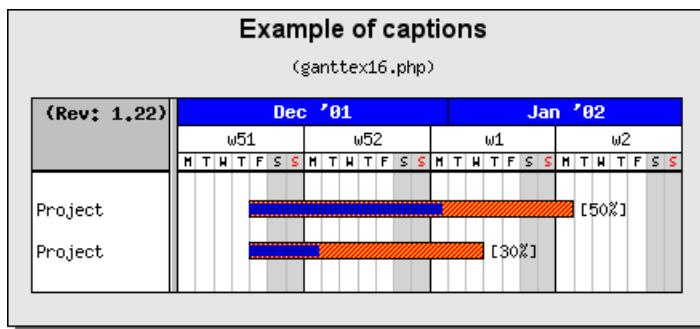
## Adding a table title

The (default) white area in the top left of the gantt table may have a title. This is accessed by the 'tableTitle' property of the gantt scale. Using this is straightforward as the following code snippet shows.

```
<?php
$graph->scale->tableTitle->Set('(Rev: 1.22)');
$graph->scale->tableTitle->SetFont(FF_FONT1 , FS_BOLD);
$graph->scale->SetTableTitleBackground('silver');
$graph->scale->tableTitle->Show ();
?>
```

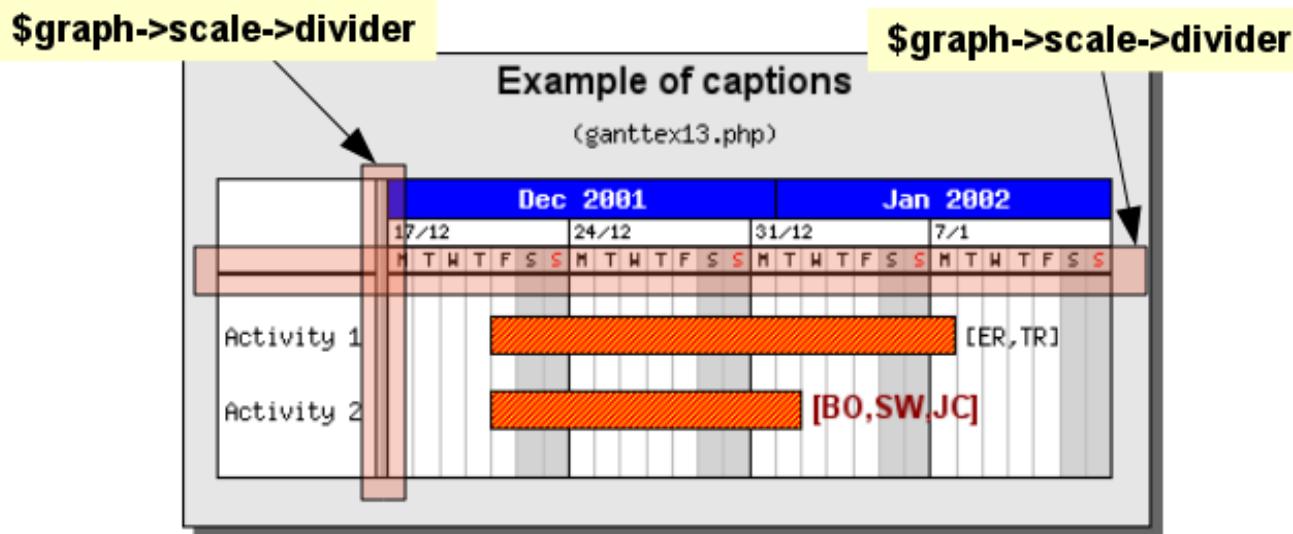
The example lines above also changes the default white background to silver. An example of this is shown in Figure 16.82, “Adding a table title in the top left corner (`ganttex16.php`)”. As can be seen the width of the left column which holds all the titles has been adjusted to make it wide enough to fit the table title.

**Figure 16.82. Adding a table title in the top left corner (`ganttex16.php`)**  
`[example_src/ganttex16.html]`



## Modifying the divider lines

The vertical and horizontal lines between the activity titles and the plot area and the bars can be modified by accessing the vertical divider 'divider' and the horizontal divider 'dividerh' properties of the scale. Figure 16.83, “Gantt divider lines” shows the exact location of the divider lines.

**Figure 16.83. Gantt divider lines**

Again, this is straightforward as the following code snippet shows.

```
<?php
$graph->scale->divider->SetWeight(3);
$graph->scale->divider->SetColor('navy');
$graph->scale->dividerh->SetWeight(3);
$graph->scale->dividerh->SetColor('navy');
?>
```

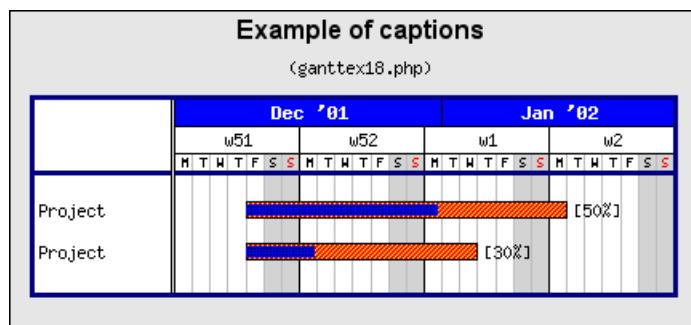
An example of this can be seen in Figure 16.84, “Adjusting the plot box around the gantt chart (ganttex18.php)”

## Modifying the box around the plot

In a similar manner to the other plots in the library the Box around the plot can be modified with the gantt graph method 'GanttGraph::SetBox()' . The following line will result in a thicker plot box around the plot area as can be seen in Figure 16.84, “Adjusting the plot box around the gantt chart (ganttex18.php)”

```
$graph->SetBox(true, 'navy', 3);
```

**Figure 16.84. Adjusting the plot box around the gantt chart (ganttex18.php)**  
[example\_src/ganttex18.html]



## Horizontal grids and alternating line colors

In order to make large charts easier to read it is possible to specify an horizontal grid and optional alternating line colors in the background for Gantt charts. The horizontal grid is accessed through the `Graph::hgrid` property and the line (used in the grid) is accessed through the `Graph::hgrid::line` sub-property.

In order to specify alternating line colors the following method is used

- `HorizontalGridLine::SetRowFillColor($aColor1,$aColor2='')`  
  \$aColor1, \$aColor2, alternating color for each activity line

For example, to use an alternating blue background with blue grid line the following lines would have to be added to the graph script

```
<?php
// Setup a horizontal grid
$graph->hgrid->Show();
$graph->hgrid->line->SetColor('lightblue');
$graph->hgrid->SetRowFillColor('darkblue@0.9');
?>
```

An example of this is shown in Figure 16.85, “(gantthgridex1.php)”

## Adding icons to Gantt graphs

In the same way as for ordinary x-y graphs it is possible to add small images (or icons) to a Gantt graph by creating an instance of class `IconPlot` and then adding that instance to the graph.

For example the following line will add an icon (similar to what was shown in ??) to a gantt chart with the result shown in Figure 16.85, “(gantthgridex1.php)”

```
<?php
$icon = new IconPlot('penguin.png' , 0.01 , 0.95 , 1 , 15);
$icon->SetAnchor('left', 'bottom');
$graph->Add($icon);
?>
```

**Figure 16.85. (gantthgridex1.php) [example\_src/gantthgridex1.html]**

## Adjusting the vertical spacing between activity bars

By default the library adds 20% margin above and below each activity bar. In order to set the activities closer or further away from each other the method

- `GanttGraph::SetVMarginFactor($aFractionMargin)`

can be used. For example

```
$graph->SetVMarginFactor(0.0);
```

will cause the gantt bars to touch each other since there will be no margins. If we instead use

```
$graph->SetVMarginFactor(1.0);
```

we will in effect get "double-line-spacing" since we add the width of one activity height as margin. The default 40% corresponds to

```
$graph->SetVMarginFactor(0.4);
```

## Adjusting the margins with auto-sizing

It is possible to use `GanttGraph::SetMargin()` to specify the margin for a Gantt graph even when the vertical height is determined automatically. For example to generate a graph with no left, right or bottom margin the following lines would be needed

```
<?php
$graph = new GanttGraph (500);
$graph->SetMargin (0, 0, 30, 0);
?>
```

## Limiting the date range for the Gantt chart

By default the scale will be wide enough to make room for all specified activities. It is however possible to manually set the scale range to limit the size of the Gantt chart.

This is done by calling the method

- `GanttGraph::SetDateRange($aStartDate, $aEndDate)`

For example the following code snippet would set the specified start and end date

```
$graph->SetDateRange('2001-12-20' , '2002-01-20');
```

## Specifying the first day in the week

By default Monday is the first day of the week. It is possible to manually set an arbitrary start day by calling the method

- `GanttScale::SetWeekStart($aStartDay)`

`$aStartDay`, Specified as an integer in the range 0-6 where 0=Sunday, 1=Monday, ..., 6=Saturday

## Adding plot icons to the graph

In the same way as was described in Section 14.14, “Adding icons (and small images) to the graph” for x-y graphs it is possible to add icons and country flags to a Gantt chart. The following code snippet adds a Norwegian flag to the top left corner of a graph

```
<?php
// Setup the Gantt graph
$graph = new GanttGraph();

// Create some activities
// ...

$icon = new IconPlot();
$icon->SetAnchor('left','top');
$icon->SetCountryFlag('norway');
$icon->SetPos(5,5);
$icon->SetMix(50);
```

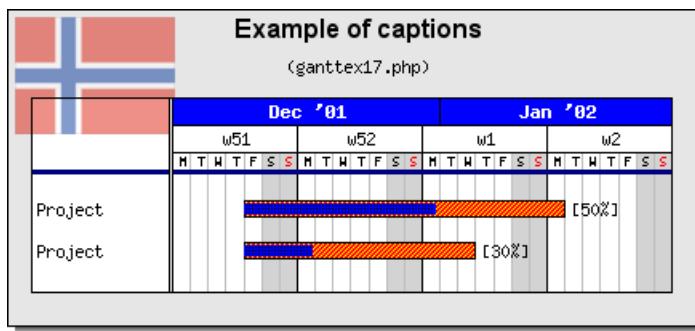
```
$icon->SetScale(1.0);

// The above four method calls could also have been done as
// $icon->SetCountryFlag('norway', 5,5, 1.0, 50);

$graph->Add($icon);

// Any other gantt formatting
// ...
```

The result of adding these lines to a typical Gantt chart is shown in Figure 16.86, “Adding a country flag to the top left corner of the gantt graph (`ganttex17-flag.php`) [[example\\_src/ganttex17-flag.html](#)]



For more details on adding icons to a graph see Section 14.14, “Adding icons (and small images) to the graph”.

## Adding text strings to the graph

In exactly the same way as was described in Section 14.17, “Adding arbitrary texts to the graph” it is also possible to add arbitrary formatted text paragraphs to the gantt chart.

The position of the text strings is specified as either an absolute position in pixels (as usual the top left corner is (0,0)) or the position can be specified as a scale position with date and row index. The following two methods of the text class is used for this

- `Text::SetPos($aPosX, $aPosY)`  
Set the absolute position for the text anchor point
- `Text::SetScalePos($aDate, $aRow)`  
Set the position for the text anchor point using the actual date scale (as the horizontal position) and the row number as the vertical position.

For example the lines below show how to add one text with absolute scale position in the top left corner and one text string at the second row at the date "2002-01-01"

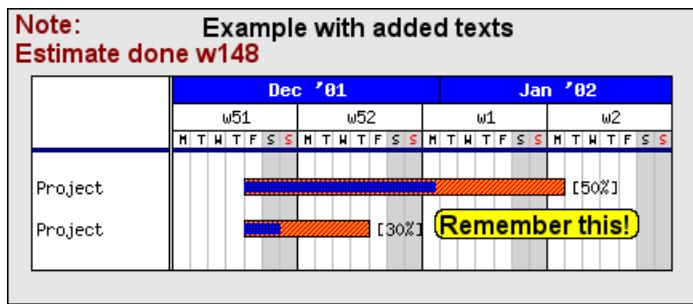
```
<?php
// Add text to top left corner of graph
$txt1 = new Text();
$txt1->SetPos(5,2);
$txt1->Set("Note:\nEstimate done w148");
```

```
$txt1->SetFont(FF_ARIAL,FS_BOLD,12);
$txt1->SetColor('darkred');
$graph->Add($txt1);

// Add text to the top bar
$txt2 = new Text();
$txt2->SetScalePos('2002-01-01',1);
$txt2->SetFont(FF_ARIAL,FS_BOLD,12);
$txt2->SetAlign('left','center');
$txt2->Set("Remember this!");
$txt2->SetBox('yellow');
$graph->Add($txt2);
?>
```

The result of adding these lines to a typical gantt graph is shown in Figure 16.87, “Adding two text objects to a Gantt graph (`gantt_textex1.php`) [`example_src/gantt_textex1.html`]”

**Figure 16.87. Adding two text objects to a Gantt graph (`gantt_textex1.php`) [`example_src/gantt_textex1.html`]**



Since there is no other changes in functionality we refer to Section 14.17, “Adding arbitrary texts to the graph” for a full discussion of text paragraph features.

### Note

This feature was added in v2.5

## 16.4.12. Localizing the Gantt chart scale

Since the name of the week and months are constructed by the library it must be possible to adjust which locale should be used to construct the names.

Depending on the server setup of PHP there might be support for several locales. By default the locale is set to use the default locale on the server.

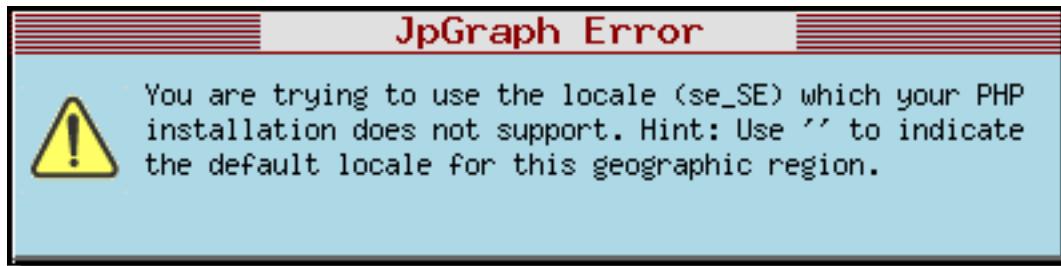
A specific locale is specified with the locale string, for example American English is specified with the string 'EN\_US', British English with 'EN\_UK' 'nl\_NL' for Dutch and so on. If the server installation does not support the specified locale an error message like the one shown in Figure 16.88, “Error message when using an unsupported Locale in Gantt chart” will be shown.

### Tip

If the server is running on a Unix derivate the supported locales can be found by the following command

```
$> locale -a
```

**Figure 16.88. Error message when using an unsupported Locale in Gantt chart**



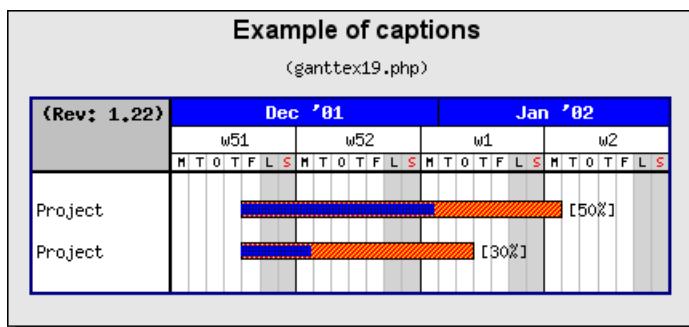
To set the locale the following method is used

- `GanttScale::SetDateLocale($aLocaleString)`

For example to set the locale to swedish the following line would be needed

```
$graph->scale->SetDateLocale('sv_SE');
```

An example of using Swedish locale is shown in Figure 16.89, “Using Swedish locale. Notice the L for Lordag instead of S for Saturday (`ganttex19.php`) [example\_src/ganttex19.html]



### Note

Internally the library uses the PHP function `setlocale()` and only affects the category `LC_TIME`

## 16.4.13. CSIM Support in Gantt charts

The generic description on how to use CSIM (Client side image maps) together with the library is fully described in Chapter 10, *Using CSIM (Client side image maps)*.

### Adding CSIM (Client side Image Maps) to Gantt charts

Gantt charts can have independent (different targets) hot spots in both the activities and in the associated title (or titles) for each activity. For activities both activity bars and milestones support CSIM functionality.

The targets and the associated "Alt" text for an activity bar is set by using one or both of the methods

- GanttPlotObject::SetCSIMTarget(\$aTarget, \$aAlt='', \$aWinTarget='')
- GanttPlotObject::SetCSIMAlt(\$aAlt)

The following code snippet sets CSIM targets for both the entire activity bar as well as the title

```
$bar->SetCSIMTarget("http://localhost/abc/", "Alt Text for the bar");
$bar->title->SetCSIMTarget("http://localhost/abc", "Alt Text for the title");
```

The following properties of a Gantt object can have CSIM targets

- GanttPlotObject::title,
- GanttPlotObject::leftmark, BarPlot::rightmark
- GanttPlotObject::caption

### Note

In the example directory there are several complete examples of how to use CSIM together with Gantt charts.

## Specifying CSIM entries for column titles

In exactly the same way as for a single title it is possible to specify individual CSIM targets for each of the title columns. This is accomplished by specifying an array for both the target and the alt text instead of a single string as arguments for SetCSIMTarget(). The following code snippet shows how to specify that.

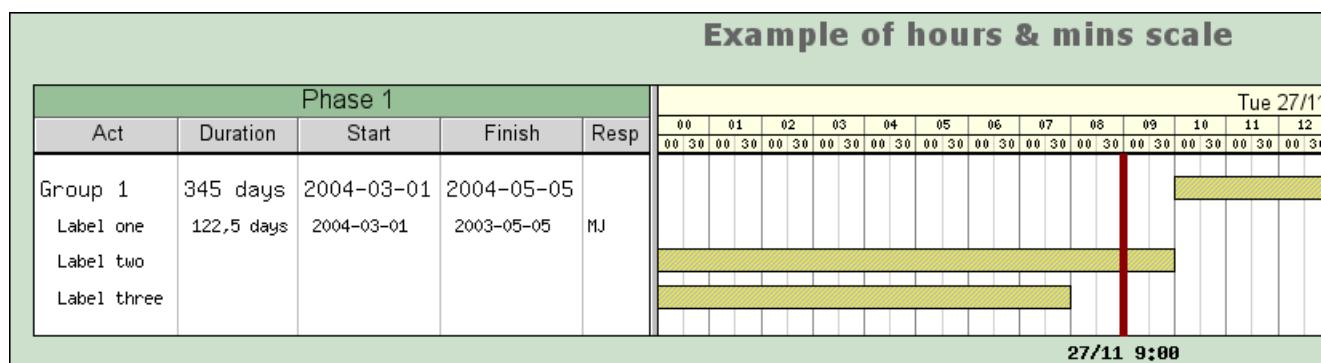
```
$bar->title->SetCSIMTarget(
 array('sometarget1.html', 'sometarget1.html'),
 array('alttext1', 'alttext2'));
```

## 16.4.14. Some final Gantt graph examples

The example given below illustrates one or more of the features available for the gantt graphs and shows how they may be combined to achieve the wanted effect.

### Examples of using hours and minute scales

**Figure 16.90. Using multiple title columns with a scale of one day (gantthourminex1.php) [example\_src/gantthourminex1.html]**

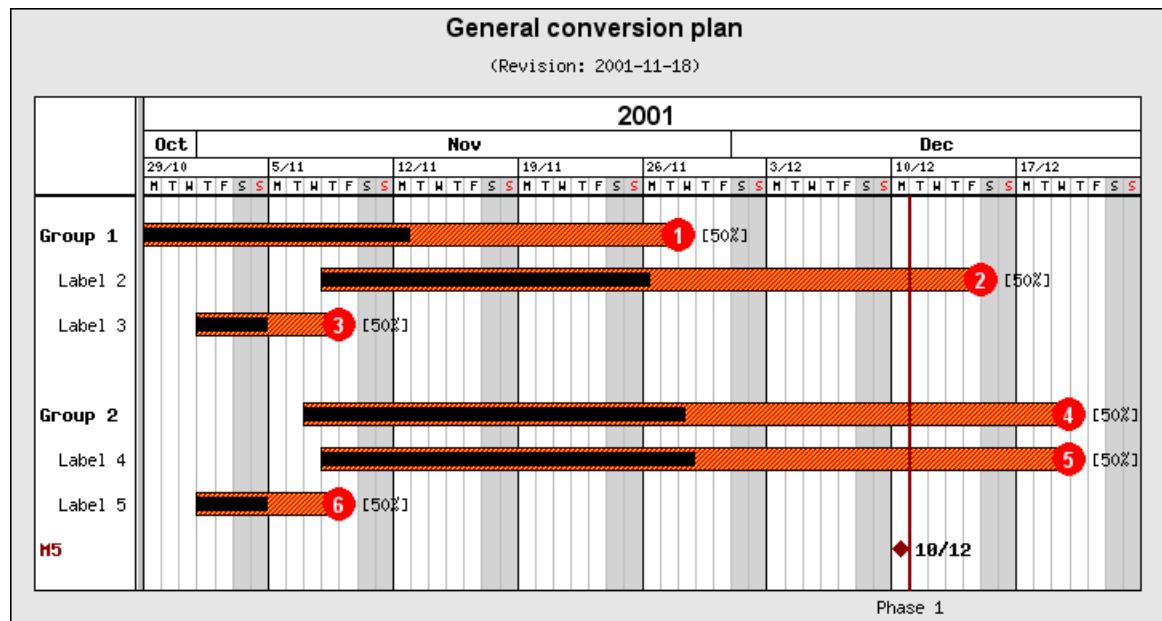


## Note

It is not possible to show less than one day in the scale. This is a limitation of the library.

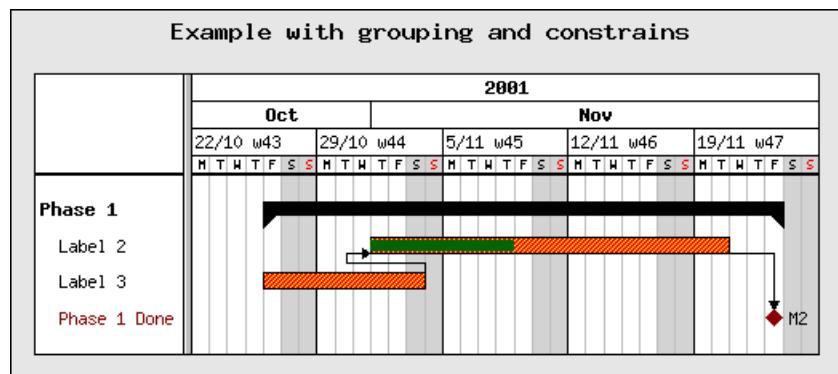
## Adding multiple activities with markers and indenting titles

**Figure 16.91.** Using multiple markers and indenting titles in the Gantt chart (`ganttex30.php`) [[example\\_src/ganttex30.html](#)]



## Using grouping of bars together with constraints

**Figure 16.92.** Using a grouping bar together with constraints (`ganttconstrainex2.php`) [[example\\_src/ganttconstrainex2.html](#)]



---

# Chapter 17. Additional graph types

These chapter describes the types of graphs supported by the library that is not a standard graph type used to display data series.

## 17.1. LED bill boards

This feature allows the creation of displayed letters (some) and digits which have the look of a 4x7 LED display.

In order to use this feature the module "jpgraph\_led.php" must be included.

### Caution

In order to use the LED module the PHP installation must have multi-byte strings enabled so that the function `mb_strlen()` is available. This is normally enabled at compile time for PHP by specifying the options `--enable-mbstring` `--enable-mbregex` when configuring the compile options.

The figures below shows some samples of the characters supported and the available colors

Figure 17.1. LEDC\_GREEN (`ledex1.php`) [example\_src/  
`ledex1.html`]

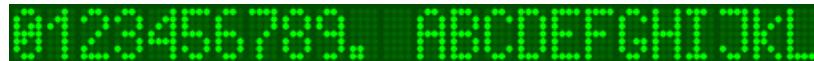


Figure 17.2. LEDC\_RED (`ledex2.php`) [example\_src/`ledex2.html`]



Figure 17.3. LEDC\_YELLOW (`ledex3.php`) [example\_src/  
`ledex3.html`]



Figure 17.4. LEDC\_BLUE (`ledex5.php`) [example\_src/`ledex5.html`]



Figure 17.5. LEDC\_GRAY (`ledex6.php`) [example\_src/`ledex6.html`]



Figure 17.6. LEDC\_INVERTGRAY (`ledex17.php`) [example\_src/  
`ledex17.html`]



Figure 17.7. LEDC\_CHOCOLATE (ledex7.php) [example\_src/ledex7.html]



Figure 17.8. LEDC\_PERU (ledex8.php) [example\_src/ledex8.html]



Figure 17.9. LEDC\_GOLDENROD (ledex9.php) [example\_src/ledex9.html]



Figure 17.10. LEDC\_KHAKI (ledex10.php) [example\_src/ledex10.html]



Figure 17.11. LEDC\_Olive (ledex11.php) [example\_src/ledex11.html]



Figure 17.12. LEDC\_LIMEGREEN (ledex12.php) [example\_src/ledex12.html]



Figure 17.13. LEDC\_FORESTGREEN (ledex13.php) [example\_src/ledex13.html]



Figure 17.14. LEDC\_TEAL (ledex14.php) [example\_src/ledex14.html]



Figure 17.15. LEDC\_STEELBLUE (ledex15.php) [example\_src/ledex15.html]



**Figure 17.16. LEDC\_NAVY (ledex16.php) [example\_src/ledex16.html]**



A LED display is constructed by creating an instance of class DigitalLED47

- `DigitalLED47::__construct($aRadius = 2, $aMargin= 0.6)`

The first argument specifies the radius of each "led-light" and the second argument the margin around each character. Adjusting the radius is the only way to adjust the size of the LED characters.

In order to generate the billboard the method

- `DigitalLED47::Stroke($aValStr, $aColor , $aFileName = '')`

`$aValStr`, The string to be displayed

`$aColor`, one of the available colors as shown in the figures above

`$aFileName`, An optional file name which specifies a file that the image will be written to. No image will be streamed back to the browser in this case.

is called with the string to be displayed as the first argument. The following code snippet shows how to create a green LED bill board

```
$led = new DigitalLED74();
$led->Stroke('0123456789. ABCDEFGHIJKL' ,LEDC_GREEN);
```

In order to improve visual quality of the LED display it uses super sampling internally to achieve the anti-alias smoothing effect. The number of oversampling is specified with

- `DigitalLED47::SetSupersampling($aSuperSampling)`

By default a 2-times super-sampling is used. Using any value higher than 4 does not give any visual improvements

In the three figures below different levels of super sampling is used, in the left most image no super sampling is used (super sampling=1), in the middle figure super sampling=2 (the default) and in the rightmost figure super sampling is set to 4

**Figure 17.18 supersampling=1 (ledex4.php) [example\_src/ledex4.html]**



**Figure 17.18 supersampling=2 (default) (ledex4.1.php) [example\_src/ledex4.1.html]**



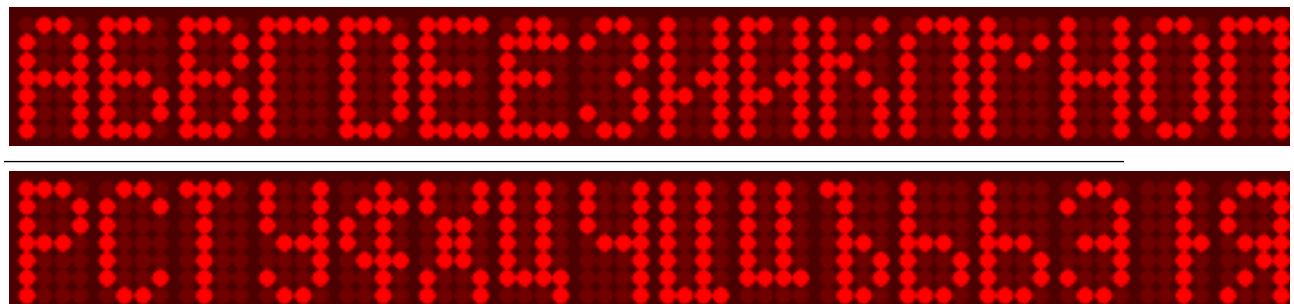
**Figure 17.19 supersampling=4 (ledex4.2.php) [example\_src/ledex4.2.html]**



### 17.1.1. Cyrillic character support

In addition to most of the basic latin characters the library also have some support for Cyrillic characters. Due to the limed resolution (4x7) some more complex cyrillic characters are not rendered very faithfully. The available rendering for the cyrillic alphabet is shown in Figure 17.20, “LED 4x7 Cyrillic alphabet support”

**Figure 17.20. LED 4x7 Cyrillic alphabet support**



### **Caution**

Since the library internally uses utf8 character encoding it means that the string passed on to the library must also be encoded in utf8. In case the characters have another encoding, say cp1251, they must first be converted to utf8. This could for example be done by the PHP function `iconv()`.

(This is the only place in the library that is truly multi-byte text safe.)

## 17.2. Captcha generation

Captcha is an acronym for "*Completely Automated Public Turing test to tell Computers and Humans Apart.*" and is often used to prevent automatic sign-ups in various WEB-based applications.

The library support generation of a simple captchas based on hand drawn "ugly" letters and digits. They are combined into a word which can be presented as an image.

## Warning

It should be noted that with ample processing power and modern image analysis it is probably not that difficult to actually break these captchas so the usage of these captchas in mission critical environments is entirely the responsibility of the user.

The module in the library that is needed is "jpgraph\_antispam.php" and behaves as a simplified plot module.

?? shows an example on how this can look

Figure 17.21. Sample illustration of captcha challenge (`antispamex01.php`)  
[`example_src/antispamex01.html`]



Captcha images have less functionality than the usual graphs generated with the library in order to keep this utility reasonable small. The primary limitation is that there are no additional formatting options for the images and the image generated will always use the JPEG image format. Hence it is not possible to change this to use, for example, PNG format.

## 17.2.1. Generating Captchas

There are two basic alternatives on how to generate the content of the captcha

1. Submit a string that should be used
2. Automatically generate a random string. If this alternative is chosen then the user of the library should save the created string and compare it to what the user enters.

In order to write a script to generate a new challenge there are four steps to be completed.

1. include the library file "jpgraph\_antispam.php". Note that there is no need to include the "jpgraph.php" library since all functionality is included in this library file.

```
require_once "jpgraph_antispam.php";
```

2. a new instance of the class AntiSpam must be created

```
$spam = new AntiSpam();
```

3. the string to be used in the challenge must be specified. To automatically generate a suitable string use

```
// The argument determines the length of the generated string
$chars = $spam->Rand(5);
```

If instead the string to be used should be specified this string should be specified in the initial creation of the AntiSpam() or by calling the Set() method as in

```
$spam->Set('au18k');
```

Please note that in order to minimize the risk for confusion the letters 'O' and the number '0' (zero) is not allowed since they are too alike and can be mistaken for each other.

4. output the image with a call the method Stroke() on the created instance of the AntiSpam class.

```
if($spam->Stroke() === false) {
 die("Illegal or no data to plot");
}
```

Note that we have put a guard around the output since in the case of an error this method will result a boolean false value. As with the other graph types it is possible to write the generated image to a file by submitting a file name as an argument to Stroke().

In order to practically use this module the challenge string is most likely passed to the image script via a URL argument, saved to a file and the read back in the HTML page that is providing the captcha challenge.

### Warning

It should be pointed out on more time that modern image analysis technology is fairly good at automatically read these types of images and translate it back to the letters they represent so this

type of captchas does not provide any guarantee for automatic sign-ups. There are active academic research on how to apply various types of artificial intelligence to read many types of captchas.

## 17.3. Canvas graphs

This is basically a free form blank canvas where any graphics can be drawn. Canvas graphs area also used as the basic graph for graphic tables as described in Chapter 19, *Graphical tables*

The library provides two module files to aid with this "jpgraph\_canvas.php" and "jpgraph\_canvtools.php". The second module is strictly speaking not necessary but provides a number of utility classes that makes it much easier to work with canvas.

Canvas graphs is provided as a layer to make it easier to create arbitrary images and still make use of some of the convenience methods available in the library (like font and color support).

Canvas graphs makes it easy to use the low level graphic drawing primitives in the library provided by the two classes

### 1. class Image

This is the basic low level class that encapsulates the GD library by providing a complete framework for using graphic primitives with a coherent API and many convenience methods.

### 2. class RotImage

This is a descendant of class Image that in addition to the basic primitives also provides support for rotation of graphs

It is possible to work at different levels of abstraction when creating a canvas.

1. By only using the basic Image and RotImage classes. This can be considered one abstraction level above the most basic GD libraries
2. By making use of the functionality provided by class CanvasGraph and the supporting classes (e.g. class CanvasScale). By using the canvas scale class it is possible to defining a coordinate grid on the canvas. These coordinates can then be used instead of absolute pixels which is necessary to us ethe basic Image and RotImage classes.

A concrete example of how canvas graphs can be used can be seen in Figure 8.1, “List of all latin TTF fonts. (listfontsex1.php)” where all the available fonts have been drawn on a canvas.

### 17.3.1. Creating a simple canvas

Creating a canvas gives you the opportunity draw arbitrary shapes on a "white" piece of paper. Let's first show a simple example were we just draw a text box.

**Example 17.1. A simple canvas graph to draw a text box (canvasesx01.php)**

```
<?php // content="text/plain; charset=utf-8"
// $Id: canvasesx01.php,v 1.3 2002/10/23 08:17:23 aditus Exp $
include "jpgraph/jpgraph.php";
include "jpgraph/jpgraph_canvas.php";

// Setup a basic canvas we can work
$g = new CanvasGraph(400,300,'auto');
$g->SetMargin(5,11,6,11);
$g->SetShadow();
$g->SetMarginColor("teal");

// We need to stroke the plotarea and margin before we add the
// text since we otherwise would overwrite the text.
$g->InitFrame();

// Draw a text box in the middle
$txt="This\nis\na TEXT!!!";
$t = new Text($txt,200,10);
$t->SetFont(FF_ARIAL,FS_BOLD,40);

// How should the text box interpret the coordinates?
$t->Align('center','top');

// How should the paragraph be aligned?
$t->ParagraphAlign('center');

// Add a box around the text, white fill, black border and gray shadow
$t->SetBox("white","black","gray");

// Stroke the text
$t->Stroke($g->img);

// Stroke the graph
$g->Stroke();

?>
```

**Figure 17.22. A simple canvas graph to draw a text box (canvases01.php) [example\_src/canvases01.html]**



The example in Figure 17.22, “A simple canvas graph to draw a text box (canvases01.php) ” starts by creating a 400x200 sized image.

The margins are then set to to get a frame around the image.

For canvases the margins has no effect in the way coordinates are entered. Top left is (0,0) and bottom right (including any potential margin and shadow) is the maximum. In this case the coordinate range are X:0-399, and Y:0-199

The `InitFrame()` method is then called which actually strokes the margin and plotarea to the graph. Since everything is stroked in the order the commands are given it is necessary to make sure that the graphical objects that should be on top are stroked last. This is different from the other graph types since they will hide these details and stroke the details in the graph in the correct order.

We then create a `Text` object, setup it's properties, including the absolute screen position where we want the text, and then stroke it.

The `Text::Align()` method specifies the anchor point of the text as explained in Chapter 8, *Text and font handling*

We also specify that the lines within the paragraph should be centered with a call to `Text::ParagraphAlign()` Since we also choose to have a box around the text we have to make use of the method `Text::SetBox()` which is used to specify the fill color, the border color and the shadow color (if the shadow color is set to ", no shadow will be used).

Now we are ready to stroke the text onto the canvas. In order to do so we must specify the graphical context when stroking the text. The current graphic context is always available in the "\$img" property of the `CanvasGraph` class.

This specification of the graphical context is needed to all `Stroke` method that is external to the main graph class.

### 17.3.2. Adding lines and rectangles to a canvas

A canvas also makes a good background for using standard graphic primitives, for example circles and lines. The first thing to remember is that this requires working with absolute screen coordinates and secondly that all drawing primitives are found in the `Image Class` accessible as a property of the `Graph class`.

This means that to draw a line between absolute coordinates (0,0) and (100,100) the following code must be created

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

The following example shows how to create some basic graphic shapes on a canvas

**Example 17.2. Drawing some basic geometric shapes on a canvas  
(canvasesex02.php)**

```
<?php // content="text/plain; charset=utf-8"
// $Id: canvasesex02.php,v 1.1 2002/08/27 20:08:57 aditus Exp $
include "jpgraph/jpgraph.php";
include "jpgraph/jpgraph_canvas.php";

// Setup a basic canvas we can work
$g = new CanvasGraph(400,200,'auto');
$g->SetMargin(5,11,6,11);
$g->SetShadow();
$g->SetMarginColor("teal");

// We need to stroke the plotarea and margin before we add the
// text since we otherwise would overwrite the text.
$g->InitFrame();

// Add a black line
$g->img->SetColor('black');
$g->img->Line(0,0,100,100);

// .. and a circle (x,y,diameter)
$g->img->Circle(100,100,50);

// .. and a filled circle (x,y,diameter)
$g->img->SetColor('red');
$g->img->FilledCircle(200,100,50);

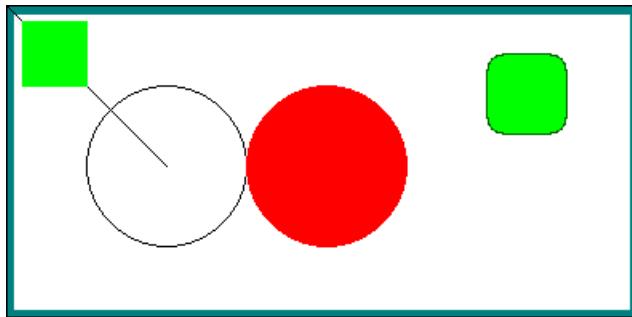
// .. add a rectangle
$g->img->SetColor('green');
$g->img->FilledRectangle(10,10,50,50);

// .. add a filled rounded rectangle
$g->img->SetColor('green');
$g->img->FilledRoundedRectangle(300,30,350,80,10);
// .. with a darker border
$g->img->SetColor('darkgreen');
$g->img->RoundedRectangle(300,30,350,80,10);

// Stroke the graph
$g->Stroke();

?>
```

**Figure 17.23. Drawing some basic geometric shapes on a canvas (`canvasesx02.php`) [[example\\_src/canvasesx02.html](#)]**



### 17.3.3. Using a canvas scale together with the shape class

The previous method using absolute coordinates works. But it gives very little flexibility in, for example, scaling the image. Working with absolute coordinates is therefore not very practical and gets tedious for complex figures.

To mitigate these issues the library offers a convenience class, `class CanvasScale`, this will allow the creation of a suitable scale that will make it easier to both create and scale the graph.

In order to use a canvas scale the module "`jpgraph_canvtools.php`" must first be included. In addition to the canvas scale class this module also provides the companion `Shape` class. This companion shape class is used to draw shapes using the specified scale onto the canvas.

Using the scale is quite simple and requires only two steps

1. An instance of the scale class is first created with the `CanvasGraph` instance as the first parameter
2. The wanted scale is then specified with the wanted x and y scales

The following code snippet shows how this is done

```
<?php
$graph = CanvasGraph(400,300);

// ... other canvas formatting

// Create a new scale
$scale = new CanvasScale($graph);
$scale->Set($xmin, $xmax, $ymin, $ymax);

// ...
?>
```

The next step needed to be able to create geometric shapes is to create an instance of class `Shape` and give the canvas graph and the scale as the two parameters as the following line shows.

```
<?php
```

```
$shape = new Shape($graph, $scale);
?>
```

It is now possible to create arbitrary geometric shapes by using the provided methods in the shape class. The shape class provides the following methods

- `Shape::SetColor($aColor)`
- `Shape::Line($x1,$y1,$x2,$y2)`
- `Shape::Polygon($p,$aClosed=false)`
- `Shape::FilledPolygon($p)`
- `Shape::Bezier($p,$aSteps=40)`
- `Shape::Rectangle($x1,$y1,$x2,$y2)`
- `Shape::FilledRectangle($x1,$y1,$x2,$y2)`
- `Shape::Circle($x1,$y1,$r)`
- `Shape::FilledCircle($x1,$y1,$r)`
- `Shape::RoundedRectangle($x1,$y1,$x2,$y2,$r=null)`
- `Shape::FilledRoundedRectangle($x1,$y1,$x2,$y2,$r=null)`
- `Shape::ShadowRectangle($x1,$y1,$x2,$y2,$fc=false,  
$shadow_width=null,$shadow_color=array(102,102,102))`
- `Shape::SetTextAlign($halign,$valign="bottom")`
- `Shape::StrokeText($x1,$y1,$txt,$dir=0,$paragraph_align="left")`
- `Shape::IndentedRectangle($xt,$yt,$w,$h,$iw=0,$ih=0,$aCorner=3,  
$aFillColor="",$r=4)`

We can now show a complete example of using the shape class

```

<?php // content="text/plain; charset=utf-8"
// $Id: canvase03.php,v 1.1 2002/08/27 20:08:57 aditus Exp $
include "jpgraph/jpgraph.php";
include "jpgraph/jpgraph_canvas.php";
include "jpgraph/jpgraph_canvtools.php";

// Define work space
$xmax=20;
$ymax=20;

// Setup a basic canvas we can work
$g = new CanvasGraph(400,200,'auto');
$g->SetMargin(5,11,6,11);
$g->SetShadow();
$g->SetMarginColor("teal");

// We need to stroke the plotarea and margin before we add the
// text since we otherwise would overwrite the text.
$g->InitFrame();

// Create a new scale
$scale = new CanvasScale($g);
$scale->Set(0,$xmax,0,$ymax);

// The shape class is wrapper around the Imgae class which translates
// the coordinates for us
$shape = new Shape($g,$scale);
$shape->SetColor('black');

// Add a black line
$shape->SetColor('black');
$shape->Line(0,0,20,20);

// .. and a circle (x,y,diameter)
$shape->Circle(5,14,2);

// .. and a filled circle (x,y,diameter)
$shape->SetColor('red');
$shape->FilledCircle(11,8,3);

// .. add a rectangle
$shape->SetColor('green');
$shape->FilledRectangle(15,8,19,14);

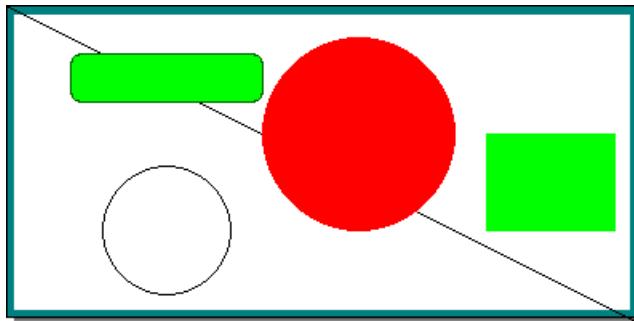
// .. add a filled rounded rectangle
$shape->SetColor('green');
$shape->FilledRoundedRectangle(2,3,8,6);
// .. with a darker border
$shape->SetColor('darkgreen');
$shape->RoundedRectangle(2,3,8,6);

// Stroke the graph
$g->Stroke();

?>

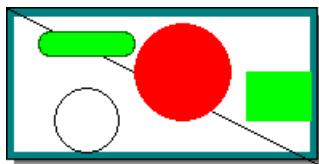
```

**Figure 17.24. Creating a canvas graph with a scale and using the shape class (canvasesx03.php) [example\_src/canvasesx03.html]**



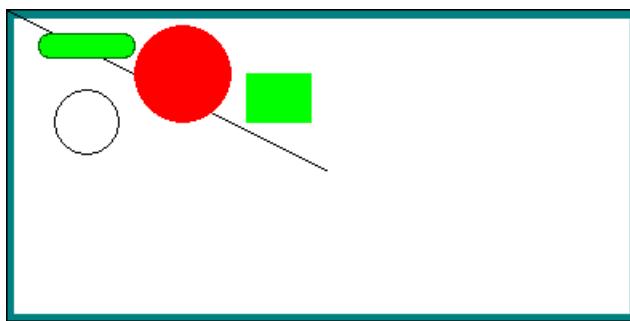
To show the usefulness of using the scale class Figure 17.25, “Changing the image size to create a smaller version of the previous example (canvasesx04.php) ” shows the result after just changing the image size but keeping the code the same. The relation between the shapes will remain the same but in a smaller format to fit the new image size.

**Figure 17.25. Changing the image size to create a smaller version of the previous example (canvasesx04.php) [example\_src/canvasesx04.html]**



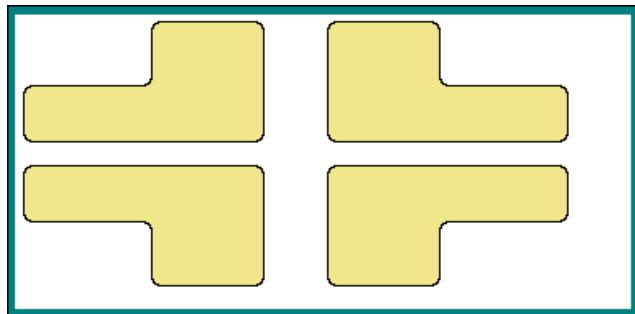
If we instead wanted to keep the image size but shrink the shapes we could just make the scale twice as large which would result in Figure 17.26, “Keeping the image size but changing the scale (canvasesx05.php) ”

**Figure 17.26. Keeping the image size but changing the scale (canvasesx05.php) [example\_src/canvasesx05.html]**



We previously mentioned that the Shape class was a wrapper around the image class with one exception. The exception is that it contains one additional method which draws an "indented rectangle". An indented rectangle is a rectangle where one of its four corners have been moved into the rectangle Figure 17.27, “Example of using an indented rectangle (canvasesx06.php) ” shows an example of using this shape.

**Figure 17.27. Example of using an indented rectangle (`canvasesx06.php`)  
[`example_src/canvasesx06.html`]**



As a final note we mention the utility class `class CanvasRectangleText`. This can be used to add a text with a rounded rectangle (possibly filled) onto the canvas. Figure 8.1, “List of all latin TTF fonts. (`listfontsex1.php`)” where all the available fonts were shown were using this class.

---

# Chapter 18. Miscellaneous formatting and tools

This chapter tries to collect all other formatting options (and tricks) available in the library that doesn't really fit anywhere else

## 18.1. Linear regression analysis

The library offers support to do basic linear regression analysis with the help of the utility : class `LinearRegression` defined in the utility module "`jpgraph_utils.inc.php`". With this class it is possible to make a linear estimation of data and calculate some of the basic statistics of the data, i.e. correlation coefficient and standard error.

The `LinearRegression` class is instantiated with the data to be analyzed and it is then possible to both get hold of the statistics that corresponds to the data and also to automatically calculate a range of estimated y-values for a given set of x-coordinates.

The following code snippet shows how to instantiate a regression analysis with some data

```
<?php
require_once('jpgraph_utils.inc.php')

// Some data to be analyzed (both x, and y-data must be specified)
$datay = array(...) ;
$datax = array(...) ;

// Instantiate the linear regression class
$linreg = new LinearRegression($datax, $datay);

// Get the basic statistics
list($stderr, $corr) = $linreg->GetStat();

// Get a set of estimated y-value for x-values in range [0,20]
list($x, $esty) = $linreg->GetY(0,20)

...
?>
```

The methods available in the `LinearRegression` class that can be used are

- `LinearRegression::GetStat()`

Returns an array with (standard error, correlation coefficient, determination coefficient)

The closer the correlation coefficient is to 1 the more of the data variation can be explained by a linear estimate.

- `LinearRegression::GetAB()`

Return an array of the linear coefficients (a,b) where the linear estimation is  $y = a + b*x$

- `LinearRegression::GetY($aMinX, $aMaxX, $aStep=1)`

Return an array with (xdata, ydata) corresponding to an x-range between x values in range [`$aMinX`, `$aMaxX`] with steps of `$aStep`

The following example shows how to use this utility class to plot both the original data as well as the estimated linear line.

```

<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_scatter.php");
require_once ("jpgraph/jpgraph_line.php");
require_once ("jpgraph/jpgraph_utils.inc.php");

// Create some "fake" regression data
$datay = array();
$datax = array();
$a= 3.2;
$b= 2.5;
for($x=0; $x < 20; ++$x) {
 $datax[$x] = $x;
 $datay[$x] = $a + $b*$x + rand(-20,20);
}

$lr = new LinearRegression($datax, $datay);
list($stderr, $corr) = $lr->GetStat();
list($xd, $yd) = $lr->GetY(0,19);

// Create the graph
$graph = new Graph(300,250);
$graph->SetScale('linlin');

// Setup title
$graph->title->Set("Linear regression");
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);

$graph->subtitle->Set('stderr='.sprintf('.%.2f',$stderr).', corr='.sprintf('%.2f',
$graph->subtitle->SetFont(FF_ARIAL,FS_NORMAL,12);

// make sure that the X-axis is always at the
// bottom at the plot and not just at Y=0 which is
// the default position
$graph->xaxis->SetPos('min');

// Create the scatter plot with some nice colors
$sp1 = new ScatterPlot($datay,$datax);
$sp1->mark->SetType(MARK_FILLED CIRCLE);
$sp1->mark->SetFillColor("red");
$sp1->SetColor("blue");
$sp1->SetWeight(3);
$sp1->mark->SetWidth(4);

// Create the regression line
$lplot = new LinePlot($yd);
$lplot->SetWeight(2);
$lplot->SetColor('navy');

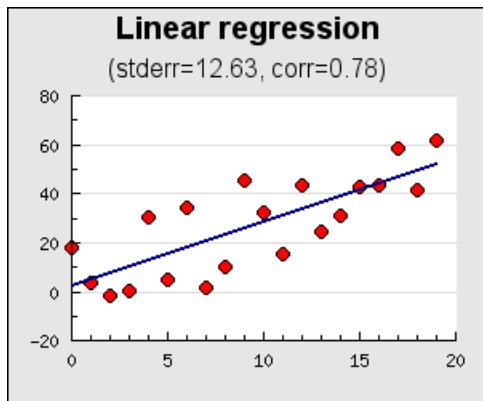
// Add the pltos to the line
$graph->Add($sp1);
$graph->Add($lplot);

// ... and stroke
$graph->Stroke();

?>

```

**Figure 18.1. Linear regression using utility class (`example16.6.php`)  
[`example_src/example16.6.html`]**



---

## **Part V. Additional graph types available in the professional version**

---

## Table of Contents

19.	Graphical tables .....	368
19.1.	Introduction .....	368
19.2.	Constructing tables .....	369
19.3.	CSIM Table support .....	379
19.4.	Table API Overview .....	380
19.5.	Examples .....	381
19.6.	Case study: Adding a table to a bar graph .....	384
20.	Odometer .....	387
20.1.	Introduction .....	387
20.2.	Creating and formatting basic odometer graphs .....	388
20.3.	Working with the odometer scale .....	399
20.4.	Adding and positioning multiple odometers to a graph .....	404
20.5.	Adding icon and text objects to the graph .....	408
21.	Windrose .....	411
21.1.	Introduction .....	411
21.2.	Creating and formatting basic Windrose graphs .....	416
21.3.	Formatting the plot .....	423
21.4.	Some more advanced formatting .....	429
21.5.	Adding icon and text objects to the graph .....	435
21.6.	Using layout classes to position Windrose plots .....	438
21.7.	Example section .....	442
22.	Matrix graphs .....	463
22.1.	Introduction .....	463
22.2.	Creating and formatting a basic matrix graph .....	465
22.3.	Mesh interpolating of input data .....	467
22.4.	Formatting the matrix plot .....	468
22.5.	Adding icon and text objects to the graph .....	476
22.6.	Adding marker lines to the matrix plot .....	481
22.7.	Using layout classes to position matrix plots .....	482
22.8.	Built in color maps .....	485
22.9.	Using CSIM with matrix plots .....	487
22.10.	Matrix graph examples .....	490
23.	Filled contour graphs .....	491
23.1.	Filled Contour graphs .....	491

# Chapter 19. Graphical tables

## 19.1. Introduction

### Note

This module is only available in the pro-version of the library.

When visualizing data it is often useful to have both a quick graphical view which can show high level trends and a detailed table view with the exact figures. The table module supports the creation of an almost endless varieties of tables.

The table library module will allow any kind of rectangular table with an arbitrary number of rows and columns. user selectable size, cells may be merged with other cells, have individual background colors, different fonts and various types of borders.

Since graphic tables can be created as objects this also means that they can be added to ordinary graph types in much the same way as for example icons and text objects.

A good way to get an idea of the capabilities of the library is to review Figure 19.1, “Standalone table examples” where a number of different tables that all have been created with the library are shown. In the remainder of this chapter the APIs available will be explained by means of a number of worked through examples.

Since table has full support for Client Side Image Maps (CSIM) they can also be used as a launch pad to take the user to further detailed information.

In addition to adding data tables to graphs it is also possible to create graphic tables on the fly all by its own. This has the advantage compared with HTML tables to allow users to make copies of the table while maintaining the exact formatting of the table.

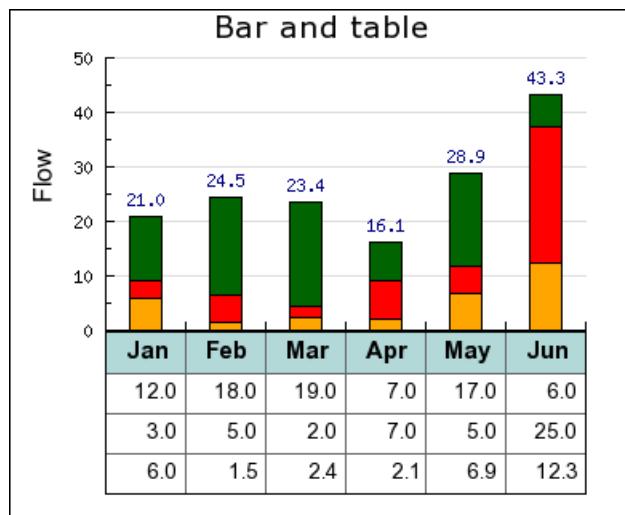
**Figure 19.1. Standalone table examples**

Figure 19.1 displays six examples of graphical tables:

- Table 1:** A standard 6x6 grid table with columns labeled w631 through w636. Rows include Critical (sum), High (sum), Low (sum), and Sum. Cells contain numerical values ranging from 3 to 47.
- Table 2:** A 6x6 grid table with columns labeled w631 through w636. Rows include Critical (sum), High (sum), Low (sum), and Sum. Cells contain numerical values ranging from 3 to 47. The first column is shaded red.
- Table 3:** A 6x6 grid table with columns labeled 2007, Q1, Q2, Jan, Feb, Mar, Apr, May, Jun. Rows include Min and Max. Cells contain numerical values ranging from 15.2 to 21.5 for Jan, and 23.9 to 42.6 for Max.
- Table 4:** A 6x6 grid table with columns labeled 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12. Rows include 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, and Sum. Cells contain numerical values ranging from 1.0 to 12.0.
- Table 5:** A 6x6 grid table with columns labeled Jan, Feb, Mar, Apr, May, Jun. Rows include Team 1 and Team 2. Cells contain numerical values ranging from 15.2 to 42.6.
- Table 6:** A 6x6 grid table with columns labeled Feb, Mar, Apr, May, Jun. Rows include Team 1, Team 2, and Sum. Cells contain numerical values ranging from 12.5 to 64.1. The first row is shaded yellow, the second row is shaded red, and the third row is shaded green.

In addition to the standalone tables shown above another common usage is to combine a graph and a table. An example of how this can be done is shown in Figure 19.2, “Combining a graphic table and a bar graph” below. The exact steps how to create this graph is described in Section 19.6, “Case study: Adding a table to a bar graph”.

**Figure 19.2. Combining a graphic table and a bar graph**



## 19.2. Constructing tables

In this section we will introduce the basic concepts of tables and explain the basic formatting possibilities that are available.

### 19.2.1. Basic tables

A table is created as an instance of the class `GTextTable` and can be managed in much the same way as for example Icons, Texts or other plot objects. It can be added to the graph (at specified X and Y coordinates) using the standard `Graph::Add()` method.

The creation of a table starts with including the necessary file "jpgraph\_table.php" which contains the class definition for the `GTextTable` class. In order to create a table we also need a graph context to which the table can be added. This can be either one of the ordinary graphs which used an instance of class `Graph` or to create a stand alone table by using an instance of class `CanvasGraph`.

Creating a new table is then just matter of creating a new instance of the class `GTextTable` and calling the initialization method with the desired size (in rows and columns) of the table. To display the table stand alone we also need to create a canvas graph and then add the table to the graph.

```
<?php
require_once "jprgraph/jpgraph.php";
require_once 'jpgraph/jpgraph_canvas.php';
require_once "jprgraph/jpgraph_table.php";

// Setup a basic canvas graph context
$graph = new CanvasGraph(630,600);

// Create a basic graph
```

```
$table = new GTextTable();
$table->Init(5,7); // Create a 5 rows x 7 columns table

...
// Add the table to the graph
$graph->Add($table);

// and send back to the client
$graph->Stroke();
?>
```

As can be seen above it is not necessary (but still possible) to specify an explicit physical size for the table. The physical size of the table will be automatically determined depending on the content of the table.

The cells in the table are numbered sequentially where (0,0) is the top left and (n-1, m-1) is the bottom right cell in a table with  $n$  rows and  $m$  columns.

The next step is to populate the table with some data. The content in one cell is specified with the using one of the following methods

- `GTextTable::Set($aRow, $aCol, $aText)`  
Set the specified text in cell (\$aRow, \$aCol)  
`GTextTable::Set($aData)`  
Copy the data in the two dimensional array \$aData to the table
- `GTextTable::SetImage($aRow, $aCol, $aText, $aImage)`  
Set the specified image in cell (\$aRow, \$aCol)
- `GTextTable::SetCellCountryFlag($aRow,$aCol,$aFlag, $aScale=1.0,$aMix=100,$aStdSize=3)`  
Set the specified country flag in cell (\$aRow, \$aCol)

The `Set()` method is polymorphic and can be called in two different ways.

It can either be used to specify the value of each individual cell by identifying the cell by its row and column index (starting at 0) ways. For example to continue the example above we set the first two cells on the first row to 1 and 2 vi the two calls

```
...
$table->Set(0,0,1);
$table->Set(0,1,1);
...
```

The second, and probably less tedious way is to setup a 2-dimensional array (matrix) with suitable values and then pass that 2-dimensional array as the first and only argument to the `Set()` method as the following example shows

```
$data = array(array(12, 7),
 array(10, 5));
$table->Set($data)
```

If the table is specified by a matrix directly it is not necessary to make the initial `Init()` call since it will be automatically determined by the size of the matrix.

The following basic full example creates a 2 rows by 4 columns table with consecutive values. Note here that we make use of `CanvasGraph` to provide the graph on which to add the table. The result of this script is shown in Figure 19.3, “The most basic 2x4 table (`table_howto1.php`)”

### Example 19.1. The most basic 2x4 table (`table_howto1.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once 'jpgraph/jpgraph.php';
require_once 'jpgraph/jpgraph_canvas.php';
require_once 'jpgraph/jpgraph_table.php';

// Create a canvas graph where the table can be added
$graph = new CanvasGraph(70,50);

// Setup the basic table
$data = array(array(1,2,3,4),array(5,6,7,8));
$table = new GTextTable();
$table->Set($data);

// Add the table to the graph
$graph->Add($table);

// ... and send back the table to the client
$graph->Stroke();

?>
```

**Figure 19.3. The most basic 2x4 table (`table_howto1.php`)** [[example\\_src/table\\_howto1.html](#)]

1	2	3	4
5	6	7	8

In the example above we also made use of a minor shortcut. If the data to the table is specified with a 2-dimensional array there is no need to call the `Init()` method since the size will be determined by the supplied array.

We will not yet begin discussion how to change fonts, colors and grids. Instead we will focus on the structure of the table and introduce the merge operation.

## 19.2.2. Merging cells

The only way to change the structure (as opposed to the look & feel) of the table is by merging two or more cells. The merge operation is done via one of three method calls:

- `MergeRow()`

Merge all cells in a specified row

- `MergeCol()`

Merge all cells in a specified column

- `MergeCells()`

Merge a rectangular range of cells

The first two methods are really just simplifications for the third most generic method. The first two methods merge all the cells in a specified row or column and the third method merges any arbitrary rectangular range of cells specified by the upper left and lower right corner.

In addition, by default, these calls will also center align the text in the merged cells (both horizontally and vertically). This can however be adjusted by giving the wanted alignment as additional argument to the methods or by calling the specific APIs that controls the alignment of the data within each cell.

The third (and most generic) method merge a range of cells specified by the top left and bottom right corner. So, for example, the following line will merge the rightmost four cells in the previous example

```
$table->MergeCells(0,2,1,3);
```

**Figure 19.4. Merging the rightmost 4 cells in the table (`table_howto2.php`) [`example_src/table_howto2.html`]**

1	2		3
5	6		

The merged cell can later on be referenced by the top left of the original merged cell range. This means that the merged cell in the above example is referred to as the (0,2) cell.

If we instead wanted to merge the top row of the table we could have simplified the call by using one of the alternative merge methods. In this case the `MergeRow()` method would be applicable as in

```
$table->MergeRow(0);
```

which would then give the table as shown in Figure 19.5, “Merging the top row (`table_howto3.php`)”

**Figure 19.5. Merging the top row (`table_howto3.php`) [`example_src/table_howto3.html`]**

1
5
6

In summary, the following three rules apply to merging and/or merged cells

- It is an error to try to merge already merged cells.
- A merged cell is referenced by the top left cell in the merged range.
- All formatting that can be applied to a single cell can also be applied to a merged cell.

### 19.2.3. Font adjustment

The most basic way to change the look and feel of the table is to adjust the font that is used in the different cells in the table. To adjust the font within the table the following methods are available

- `SetFont()`, Specify the default font for the entire table or for a specified range of cells
- `SetRowFont()`, Specify the font for an entire row of cells
- `SetColFont()`, Specify the font for an entire column of cells
- `SetCellFont()`, Specify the font for a single cell

By default the font in the table is set to `FF_FONT1`, i.e. the built-in bit mapped font of medium size. The default color of the font is 'black'.

### Note

If several font settings are applied to the same cell it will be the last method call before the table `Stroke()` method that will take precedence.

As a simple example the following script shows how to adjust the font used in the first merged row in the previous example.

**Figure 19.6. Adjusting the font in the top row (`table_howto5.php`) [`example_src/table_howto5.html`]**

1			
5	6	7	8

The font in Figure 19.6, "Adjusting the font in the top row (`table_howto5.php`)" was adjusted with a call to

```
$table->SetFont(0,0,FF_ARIAL,FS_BOLD,14);
```

#### Polymorphic `SetFont()`

Before finishing this section it is worth noting that the `SetFont()` method is polymorphic and can be called in two different ways.

1. In the first way it can be called with a single font (triple) argument. That font specification will then become the default for all cells in the table, for example

```
$table->SetFont(FF_TIMES,FS_NORMAL,12);
```

will set the entire table to use Times TTF font.

2. In the second way the first four arguments specifies a cell range (top left, bottom right) to apply the font setting to, for example

```
$table->SetFont(0,1,1,2,FF_TIMES,FS_NORMAL,12);
```

## 19.2.4. Color adjustment

In order to adjust the look and feel of the table it is possible to set the following colors of the table:

- The color of the text/number in each cell
- The background color of each cell

- The color of the grid (inside the table) and border (around the table)

The font color for a cell (or range of cells) is specified with one of

- `SetColor()`, Set the color for a range of cells or the entire table
- `SetRowColor()`, Set the color for a specified row of cells
- `SetColColor()`, Set the color for a specified column of cells
- `SetCellColor()`, Set the color for a specified cells

In a similar way the background color (or the fill color) of a cell can be specified with

- `SetFillColor()`, Set the fill color for a range of cells or the entire table
- `SetRowFillColor()`, Set the fill color for a specified row of cells
- `SetColFillColor()`, Set the fill color for a specified column of cells
- `SetCellFillColor()`, Set the fill color for a specified cells

The way to adjust the color of the grid and border will be discussed in the section regarding borders and grids below.

#### **Polymorphic `SetColor()` and `SetFillColor()`**

In the same manner as `SetFont()` can act on both all cells and a specified range of cells the two color setting methods work in the same way. Both `SetColor()` and `SetFillColor()` can both be called either with a single argument (in which all cells will be addressed) or by giving a range (top left and bottom right cell) to be acted upon.

The following method calls will set the background color of the first three cells in row 2 and 3 and the entire table

```
...
// First set the default fill color to lightgray
$table->SetFillColor('lightgray');

// Then Set a range to yellow
$table->SetFillColor(2,0,3,2,'yellow');
...
```

Finally we give some concluding examples to make the usage of these methods more clear.

```
...
// Set the default font color for all table cells
$table->SetColor('darkgray');

// Set the font color for all cells in row one
$table->SetRowColor(1,'darkgray');

// Set the font color for the first two cells in the second row
// (row with index=1)
$table->SetColor(1,0,1,1,'darkgray');
```

```
...
```

## Note

If there are multiple color settings for the same cell then the last method called before the final `Stroke()` call will be the color that shows.

For example the following script sets the color of cell (0,0) to both 'black' and 'yellow' but since the yellow is the last method to be called this is the color that will be used for cell (0,0)

```
...
$table->SetRowColor(0, 'black');
$table->SetColColor(0, 'yellow');
...
$graph->Stroke();
```

Finally, we illustrate the usage of these method by continuing with the previous example and coloring row 0 as shown in Figure 19.7, “Merging and setting the colors the top row (`table_howto4.php`) ”

**Figure 19.7. Merging and setting the colors the top row (`table_howto4.php`)  
[`example_src/table_howto4.html`]**

	1	
5	6	7 8

The coloring in Figure 19.7, “Merging and setting the colors the top row (`table_howto4.php`) ” was accomplished by adding the following two lines to the previous script.

```
$table->SetCellFillColor(0,0,'orange@0.7');
$table->SetCellColor(0,0,'darkred');
```

The same effect could also have been accomplished by using the row version of the color methods as in

```
$table->SetRowFillColor(0,'orange@0.7');
$table->SetRowColor(0,'darkred');
```

Which sets of methods to use in this case is a matter of personal preference since both achieve the same effect.

## 19.2.5. Adjusting table and cell sizes

The exact size of the table can only be indirectly controlled by specifying the width of the columns and the height of the rows. The width/height can either be controlled individually on a row by row (or column by column) basis or by giving all rows/columns the same width/height. This is accomplished by the methods

- `SetMinRowHeight()`, Sets the minimum height of rows
- `SetMinColWidth()`, Sets the minimum width of columns

As can be noted from the name of the methods the width and height specified is not necessarily the exact size of the row/column. It's just the minimum size. The actual size is depending on the text in the cells. The row height (and column width) will always grow in order to fit the text within the largest cell.

### Polymorphic functions

The number of arguments to the two methods can in fact be either one or two. If only one argument is supplied then it is interpreted as the minimum width/height for all columns/rows in the table. If two arguments are used then the first argument will specify what row/columns the height/width is specified for.

Continuing with the table in Figure 19.7, “Merging and setting the colors the top row (table\_howto4.php)”. If we want to make sure that all the cells in row 1 is at least 35 pixels wide we could add to add the following in the previous script

```
...
$table->SetMinColWidth(35);
...
```

Which would then give the table shown in Figure 19.8, “Setting the minimum column width to 35 pixels. (table\_howto6.php) [example\_src/table\_howto6.html]

1			
5	6	7	8

## 19.2.6. Fine tuning cell alignment and cell padding

The final bit of important formatting is the ability to specify the alignment of the data in each of the cells within the table. It is possible to specify both the horizontal and vertical alignment. The alignment can be specified as one of

- Horizontal alignment: "left", "right", "center"
- Vertical alignment: "top", "bottom", "center"

As usual there are four variants of the alignment methods

- SetAlign(), Set the align for a range of cells or the entire table
- SetRowAlign(), Set the alignment for an entire row
- SetColAlign(), Set the alignment for an entire column
- SetCellAlign(), Set the alignment for a specific cell

Each of the methods accepts two alignment arguments, the horizontal and the vertical alignment. We don't give any examples here since the usage of these methods should be obvious.

### Note

It should be mentioned that in addition to using these methods the alignment can also be set when using the merge methods (as discussed above). This is purely as a convenience since it is such a common use case to adjust the alignment when merging cells.

## 19.2.7. Adjusting border and grid lines

The final formatting option available is the shape, size and color of the border and grid lines in the table. As of this writing the library supports the following styles of grid lines.

- TGRID\_SINGLE, a basic solid line of specified width and color (default)
- TGRID\_DOUBLE, two lines of equal width separated by the same width as the line width.
- TGRID\_DOUBLE2, two lines where the left/top line is twice as thick as the bottom/right line.

**Figure 19.9. Double lines 1**  
**(table\_howto7.1.php)**  
**[example\_src/**  
**table\_howto7.1.html]**

1	2	3	4
5	6	7	8
6	8	10	12

**Figure 19.10. Double lines 2**  
**(table\_howto7.2.php)**  
**[example\_src/**  
**table\_howto7.2.html]**

1	2	3	4
5	6	7	8
6	8	10	12

The methods available to adjust the grid lines are

- SetGrid(), Set the grid line style for all grid lines
- SetColGrid(), Set the specified vertical grid line
- SetRowGrid(), Set the specified horizontal grid line

The borders in Figure 19.9, “Double lines 1 (table\_howto7.1.php) ” was modified by adding the following call to the basic table script.

```
$table->SetRowGrid(2,1,'black',TGRID_DOUBLE);
```

and in Figure 19.10, “Double lines 2 (table\_howto7.2.php) ”

```
$table->SetRowGrid(2,1,'black',TGRID_DOUBLE2);
```

The order of the arguments are 1) Row, 2) Width (weight) and 3) Line style.

Finally, in order to adjust the outer border of the table there is one last method available

- SetBorder(), Set the width and color of the outer border

### Tip

In order to remove a specific grid line or the border the width is specified as 0

As a final example the following script snippet removes many of the grid lines and borders by adding the following lines to the previous basic table script.

```
...
$table->SetBorder(0);
$table->SetGrid(0);
$table->SetRowGrid(2,1,'black',TGRID_DOUBLE2);
...
```

The result of this modification is shown in Figure 19.11, “Removing some grid lines and border. In addition we have right aligned all the cells as is common practice for numeric data. (table\_howto8.php) ” below.

**Figure 19.11.** Removing some grid lines and border. In addition we have right aligned all the cells as is common practice for numeric data.  
(table\_howto8.php) [[example\\_src/table\\_howto8.html](#)]

1	2	3	4
5	6	7	8
<b>6</b>	<b>8</b>	<b>10</b>	<b>12</b>

## 19.2.8. Specific number formatting

The final method of formatting we will briefly touch upon is the number formatting. By setting a specific number formatting it is possible to have numeric values formatted in a uniform way regardless of the original format. The number format is the same format string that would be used with the `printf()` family.

The available methods for this formatting follows the usual structure

- `SetNumberFormat()`, Set the number format for a range or the whole table
- `SetRowNumberFormat()`, Set the number format for a specific row
- `SetColNumberFormat()`, Set the number format for a specific column
- `SetCellNumberFormat()`, Set the number format for a specific cell

To illustrate the use of this formatting we start with the table in Figure 19.11, “Removing some grid lines and border. In addition we have right aligned all the cells as is common practice for numeric data. (table\_howto8.php) ”. and apply the format string “`%0.1f`” which will format the values as floating point values with one decimal. We do this by adding the method following method call

```
$table->SetNumberFormat('%.1f');
```

**Figure 19.12.** Applying a number format to the data in the cells  
(table\_howto9.php) [[example\\_src/table\\_howto9.html](#)]

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
<b>6.0</b>	<b>8.0</b>	<b>10.0</b>	<b>12.0</b>

### Note

The number format will only set the number format it will not do any rounding of the actual values. This is the responsibility of the client.

## 19.2.9. Using images and country flags in the table

In addition to ordinary text it is also possible to have arbitrary images (read from a file) or use one of the built-in country flags available in the library as background images in the table cells.

## Note

Remember to include "jpgraph\_iconplot.php" if images should be included and "jpgraph\_flags.php" if country flags are included.

The methods available for this purpose are

- `SetImage()`, Set the same image for all cells or a range of cells
- `SetCellImage()`, Set the image for a specific cell
- `SetRowImage()`, Set the image for a row of cells
- `SetColImage()`, Set the image for a column of cells
- `SetCellImageConstrain()`, Set the height or width of an image in a specified cell
- `SetCellCountryFlag()`, Use a specified country flag as image in a specified cell

For example, to add the United Kingdoms flag in cell (2,1) in the table one would have to add the call

```
$table->SetCellCountryFlag(2,1,'united kingdom');
```

as usual with the background images it is possible to scale it and also adjust the mix-in factor by using additional arguments to the above methods.

However there is one extra feature available with images and tables. If we wanted to make sure that the flag is of a specific height or width it is possible to specify this and have the library auto-scale the image to fit the specified parameter.

For example to restrict the height of the UK flag set in cell (2,1) to 20 pixels the call would be

```
$table->SetCellImageConstrain(2,1,TIMG_HEIGHT,20);
```

We conclude this section with a small example on the use of country flags in a table in Figure 19.13, “Using country flags in the table cells (`table_flagex1.php`) [[example\\_src/table\\_flagex1.html](#)]

Areas						
	USA	UK	France	Denmark	Iceland	Canada
Feb	13	17	15	8	3	9
Mar	34	35	26	20	22	16
Apr	41	43	49	45	51	47
Sum:	<b>88</b>	<b>95</b>	<b>90</b>	<b>73</b>	<b>76</b>	<b>72</b>

## 19.3. CSIM Table support

Each cell in a table can be a hotspot target. The following line will make cell (1,1) a hotspot

```
$table->SetCellCSIMTarget(1,1,'tableex02.php','View details');
```

As usual it is necessary to call `Graph::StrokeCSIM()` instead of the ordinary `Graph::Stroke()` method to generate the CSIM result.

## 19.4. Table API Overview

In this overview we do not give details about the arguments required by each method. Instead this is meant as an overview to get a feel for what is available in the library. All details are provided in the class reference.

**Table 19.1. Table API Overview**

Method name	Description
<b>Constructing</b>	
<code>GTextTable()</code>	Create a new table
<code>Init()</code>	Specify table size
<code>Set()</code>	Specify data in table
<code>SetPos()</code>	Specify position of table in graph
<code>SetScalePos()</code>	Specify scale position of table in graph
<code>SetAnchorPos()</code>	Specify anchor position for table. By default the top left
<b>Merging cells</b>	
<code>MergeCol()</code>	Merge all cells in the same column
<code>MergeRow()</code>	Merge all cells in the same row
<code>MergeCells()</code>	Merge all cells in a range
<b>Font specification</b>	
<code>SetFont()</code>	Set default font for range or entire table
<code>SetRowFont()</code>	Set font for a specific row
<code>SetColFont()</code>	Set font for a specific column
<code>SetCellFont()</code>	Set font for a specific cell
<b>Font color</b>	
<code>SetColor()</code>	Set default font color for range or entire table
<code>SetRowColor()</code>	Set font color for a specific row
<code>SetColColor()</code>	Set font color for a specific column
<code>SetCellColor()</code>	Set font color for a specific cell
<b>Fill color</b>	
<code>SetFillColor()</code>	Set fill color for range or entire table
<code>SetRowFillColor()</code>	Set fill color for an entire row
<code>SetColFillColor()</code>	Set fill color for an entire column
<code>SetCellFillColor()</code>	Set fill color fro a specific cell
<b>Alignment in cells</b>	
<code>SetAlign()</code>	Set default align for range or entire table
<code>SetRowAlign()</code>	Set align in a specific row
<code>SetColAlign()</code>	Set align in a specific column

<b>Method name</b>	<b>Description</b>
<code>SetCellAlign()</code>	Set align in a specific cell
<b>Borders and grid</b>	
<code>SetBorder()</code>	Set weight of border around the table
<code>SetGrid()</code>	Set style and weight for interior grid
<code>SetColGrid()</code>	Specify grid for a specific column
<code>SetRowGrid()</code>	Specify grid for a specific row
<b>Background images &amp; country flags</b>	
<code>SetImage()</code>	Use the specified image as background
<code>SetCellImage()</code>	Use the specified image as background
<code>SetRowImage()</code>	Use the specified image as background
<code>SetColImage()</code>	Use the specified image as background
<code>SetCountryFlag()</code>	Use a country flag as background
<code>SetImageConstrain()</code>	Specify width or height for cell image (or flag)
<b>Width and height of cells</b>	
<code>SetMinColWidth()</code>	Set minimum column width
<code>SetMinRowHeight()</code>	Set minimum row height
<b>Number formatting</b>	
<code>SetNumberFormat()</code>	Set number format for range or entire table
<code>SetRowNumberFormat()</code>	Number format for a specific row
<code>SetColNumberFormat()</code>	Number format for a specific column
<code>SetCellNumberFormat()</code>	Number format for a specific cell
<b>Cell padding and margins</b>	
<code>SetPadding()</code>	Set default padding for range or all cells
<code>SetRowPadding()</code>	Set padding in a specific row
<code>SetColPadding()</code>	Set padding in a specific column
<code>SetCellPadding()</code>	Set padding in a specific cell
<b>Text orientation</b>	
<code>SetTextOrientation()</code>	Set text orientation for entire table
<code>SetCellTextOrientation()</code>	Set text orientation for a specific cell
<b>CSIM Handling</b>	
<code>SetCellCSIMTarget()</code>	Set URL target for specified cell
<code>SetCSIMTarget()</code>	Set URL target for all cells. By using this method it is target URL as GET parameters

## 19.5. Examples

### 19.5.1. Example 1

Table formatting, variant 1, right adjusted Times font

**Figure 19.14. xx (table\_mex00.php)**

[example\_src/

table\_mex00.html]

	w631	w632	w633	w634	w635	w636
Critical (sum)	13	17	15	8	3	9
High (sum)	34	35	26	20	22	16
Low (sum)	41	43	49	45	51	47
Sum:	88	95	90	73	76	72

## 19.5.2. Example 2

Table formatting, variant 1, using Times font for data and a sans serif font for headings together with removed grid lines and filled headers.

**Figure 19.15. xx (table\_mex0.php) [example\_src/table\_mex0.html]**

	w631	w632	w633	w634	w635	w636
Critical (sum)	13	17	15	8	3	9
High (sum)	34	35	26	20	22	16
Low (sum)	41	43	49	45	51	47
<b>Sum:</b>	<b>88</b>	<b>95</b>	<b>90</b>	<b>73</b>	<b>76</b>	<b>72</b>

## 19.5.3. Example 3

Table formatting, variant 2, using the same sans serif font for all information but with some grid lines.

**Figure 19.16. xx (table\_mex1.php) [example\_src/table\_mex1.html]**

	w631	w632	w633	w634	w635	w636
Critical (sum)	13	17	15	8	3	9
High (sum)	34	35	26	20	22	16
Low (sum)	41	43	49	45	51	47
<b>Sum:</b>	<b>88</b>	<b>95</b>	<b>90</b>	<b>73</b>	<b>76</b>	<b>72</b>

## 19.5.4. Example 4

Table formatting, variant 3, where some data is highlighted and we use a thicker border.

**Figure 19.17.** xx (`table_mex3.php`) [example\_src/table\_mex3.html]

	w631	w632	w633	w634	w635	w636
Critical (sum)	13	17	15	8	3	9
High (sum)	34	35	26	20	22	16
Low (sum)	41	43	49	45	51	47
Sum:	88	95	90	73	76	72

## 19.5.5. Example 5

Show how the table can be turned 90 degrees

**Figure 19.18.** Vertical text (`table_vtext.php`) [example\_src/table\_vtext.html]

GROUP 10						
Sum:	Low (sum)	High (sum)	Critical (sum)	w631	w632	w633
88	41	34	13			
95	43	35	17			
90	49	26	15			
73	45	20	8			
76	51	22	3			
72	47	16	9			

## 19.5.6. Example 6

The following example shows how to add images (icons) to a number of cells.

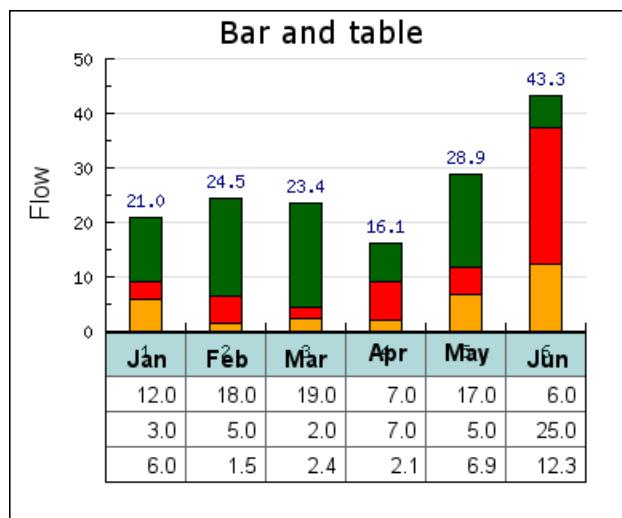
**Figure 19.19. Adding an icon (image) in a table cell (`table_iconex1.php`) [`example_src/table_iconex1.html`]**

	April	May	June	July	August
2005	7	13	17	15	8
2006	7	34	35	26	20
2007	7	41	43	49	45
<b>Sum:</b>	<b>21</b>	<b>88</b>	<b>95</b>	<b>90</b>	<b>73</b>

## 19.6. Case study: Adding a table to a bar graph

The goal of this example is to explain in detail how to achieve the graphs shown in Figure 19.20, “Combining a bar graph and a table (`tablebarex1.php`) [`example_src/tablebarex1.html`]”

**Figure 19.20. Combining a bar graph and a table (`tablebarex1.php`) [`example_src/tablebarex1.html`]**



The graph in Figure 19.20, “Combining a bar graph and a table (`tablebarex1.php`)” consists of a standard accumulated bar graph combined with a table just below the X-axis. The way to create such a graph is very basic. First create a standard bar graph with some extra bottom margin (to make room for the table) and then create and position the table below the graph. The only challenge here is to make sure that the width of the table and graph matches to line up each column under each bar.

Lets walk through the steps to do this.

The data to be plotted in the bar and in the table is created in a table as follows

```
<?php
$datay = array(
 array('Jan','Feb','Mar','Apr','May','Jun'),
 array(12,18,19,7,17,6),
 array(3,5,2,7,5,25),
 array(6,1.5,2.4,2.1,6.9,12.3));
?>
```

each line in this matrix corresponds to one row in the table and one set of bars that will make up the accumulated bar plot.

First let's define some constants that will control the size and position of the graph and table.

```
<?php
$nbrbar = 6; // number of bars and hence number of columns in table
$cellwidth = 50; // width of each column in the table
$tableypos = 200; // top left Y-coordinate for table
$tablexpos = 60; // top left X-coordinate for table
$tablewidth = $nbrbar*$cellwidth; // overall table width
$rightmargin = 30; // right margin in the graph
$topmargin = 30 // top margin of graph
?>
```

The next step is to calculate a suitable size for the entire graph, i.e. the entire image.

```
<?php
$height = 320; // a suitable height for the image
$width = $tablexpos+$tablewidth+$rightmargin; // the width of the image
?>
```

Finally its time to make use of these constants to recreate the basic graph. Note that we calculate the bottom margin to just fit the tables selected Y-coordinate position.

```
<?php
$graph = new Graph($width,$height);
$graph->img->SetMargin($tablexpos,$rightmargin,$topmargin,$height-$tableypos);
$graph->SetScale('textlin');
$graph->SetMarginColor('white');
?>
```

The rest of the code to create the accumulated bar plot is pretty mundane and we just show the code here without further comment.

```
<?php
// Create the bars and the accbar plot
$bplot = new BarPlot($datay[3]);
$bplot->SetFillColor('orange');
$bplot2 = new BarPlot($datay[2]);
$bplot2->SetFillColor('red');
$bplot3 = new BarPlot($datay[1]);
$bplot3->SetFillColor('darkgreen');
$accbplot = new AccBarPlot(array($bplot,$bplot2,$bplot3));
$accbplot->value->Show();
$graph->Add($accbplot);
?>
```

It's now time to setup the table. The position of the table is determined by the previous set constants and the original data as specified above.

```
<?php
$stable = new GTextTable();
$stable->Set($datay);
$stable->SetPos($tablexpos,$tableypos+1);
?>
```

The rest of the table formatting is fairly basic and again we just show the necessary script without further comment.

```
<?php
$stable->SetFont(FF_ARIAL,FS_NORMAL,10);
$stable->SetAlign('right');
$stable->SetMinColWidth($cellwidth);
$stable->SetNumberFormat('%0.1f');

// Format table header row
$stable->SetRowFillColor(0,'teal@0.7');
$stable->SetRowFont(0,FF_ARIAL,FS_BOLD,11);
$stable->SetRowAlign(0,'center');

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

# Chapter 20. Odometer

## 20.1. Introduction

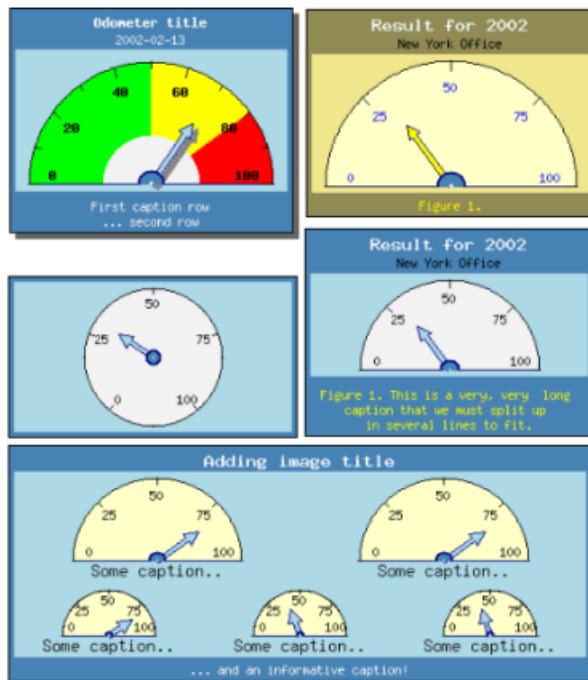
### Note

This module is only available in the pro-version of the library.

Odometer graphs uses the metaphor of an old fashioned speedometer for cars. It is normally used to give a quick overview of one specific parameter. Experience have shown that this type of metaphor is easily and quickly understood by most people. By adding color indications to the odometer it is very quick to see whether or not a specific value is within a allowed range or outside.

Figure 20.1, “Odometers” shows an overview of some typical odometers created with the library.

**Figure 20.1. Odometers**



### 20.1.1. Features

Odometers produced with this package have a wide range of functionality and as you would expect almost every bit of visual layout is in one way or the other customisable. For simple use there are suitable default values for most parameters which will make creation of simple Odometer just a matter of a few lines of code.

Among other things Odometers support the following features:

- Both half (semi-circle) and full (circle) odometers are supported
- Fully customizable scales and odometer sizes

- Different size and shapes of the indicator needles
- Scale color indicators
- Multiple dials in the same graph
- Full support for library standard fonts and colors
- Automatic layout engine for multiple plots in the same graph which aids in positioning of the graphs without having to specify pixel coordinates

## 20.2. Creating and formatting basic odometer graphs

In order to use odometer graphs the extension module "jpgraph\_odo.php" must be included in the script (in addition to the ordinary "jpgraph.php" code module).

The creation of Odometer graphs otherwise follows the traditional steps in the library of creating a graph and then adding one or several plots to the canvas.

1. Create a basic canvas graph as an instance of `class OdoGraph`
2. Create an instance of one or several odometer plots (the dials) as instances of `class OdoPlot`, set up the scale and appearance and then add them to the graph canvas.
3. Send back the graph to the client with a call to `OdoGraph::Stroke()`

The following script shows the principles

```
<?php
// Create a new canvas 300x200 pixels
$graph = new OdoGraph(300,200);

// Adjust parameters as needed
$graph->title->Set(.....)
$graph->caption->Set(...)

// Create one of more Odometer
$odo1 = new Odometer();
$odo2 = new Odometer();

...
// Adjust odometer parameters, colors etc
$odo1->needle->Set(21);
$odo2->needle->Set(47);

...
// Position the odometer plots vertically
$l = new LayoutVert(array($odo1, $odo2, ...));

// Add the odometers with the proper layout to the canvas
$graph->Add($l);
```

```
// If you only have a single odometer you may also write
// $graph->Add($odo1);

// Send back the image to the client
$graph->Stroke();
?>
```

## 20.2.1. A basic odometer

The following script shows the simplest possible odometer using just default values

### Example 20.1. A basic odometer (`odotutex00.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_odo.php");

// Create a new odometer graph (width=250, height=200 pixels)
$graph = new OdoGraph(250,140);

// Now we need to create an odometer to add to the graph.
// By default the scale will be 0 to 100
$odo = new Odometer();

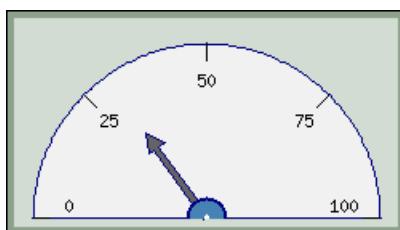
// Set display value for the odometer
$odo->needle->Set(30);

// Add the odometer to the graph
$graph->Add($odo);

// ... and finally stroke and stream the image back to the client
$graph->Stroke();

?>
```

**Figure 20.2. A basic odometer (`odotutex00.php`)** [example\_src/  
`odotutex00.html`]



The size of the Odometer is determined automatically to fit the given image size as good as possible. Since the size of the odometer automatically adjust itself to allow for any odometer caption, graph titles or graph captions there is rarely need to manually override this automation.

However it is still possible to manually specify the absolute size (of the radius) in pixels or as a fraction of the image size.

To set a specific value top the Odometer radius the following method is used

- `OdoPlot::SetSize($aRadius)`

Since clas OdoGraph inherits all the basic graph features we can easily augment the graph in Figure 20.2, “A basic odometer (`odotutex00.php`)” to, for example, have titles by adding the line

```
$graph->title->Set('A suitable graph title');
```

In the following chapters we will go through all the options that exists to modify the size, shape and appearance of the graph.

### Tip

Remember that since Odometer inherits all the basic graph feature it will support functionality like image cache (see Section 5.6, “Efficient graph generation using the built-in cache subsystem”.)

## 20.2.2. Half and Full Odometers

Odometers comes in two fundamental shapes, full and half circle. The default if nothing else is specified is to create a half circle. The type of odometer is specified in the creation of the odometer.

The type is controlled by the two constants

- `ODO_FULL` Create a full circle odometer
- `ODO_HALF` Create a half circle odometer. This is also the default value.

As an example the following odometer plot instantiation will create a full odometer plot

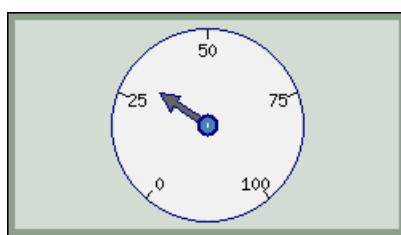
```
$odo = new Odometer(ODO_FULL);
```

and the following a half odometer plot

```
$odo = new Odometer(ODO_HALF);
```

In ?? an example of a very basic full circle odometer is shown

**Figure 20.3. A full circle odometer (`odotutex01.php`) [example\_src/  
`odotutex01.html`]**



## 20.2.3. Adding titles and captions

Remember that each graph can have a title, a subtitle and subsubtitle. In addition the Odmetter graphs offers the possibility to have a caption string. This is a text (much like the title) but is drawn at the bottom of the graph instead of at the top.

The following script shows how to add a title, subtitle and caption to our previous example

### Example 20.2. Adding titles and captions (`odotutex02.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_odo.php");

// Create a new odometer graph (width=250, height=200 pixels)
$graph = new OdoGraph(250,180);

// Setup titles
$graph->title->Set("Result for 2002");
$graph->title->SetColor("white");
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);
$graph->subtitle->Set("New York Office");
$graph->subtitle->SetColor("white");
$graph->caption->Set("Figure 1. Branch results.");
$graph->caption->SetColor("white");

// Now we need to create an odometer to add to the graph.
// By default the scale will be 0 to 100
$odo = new Odometer();

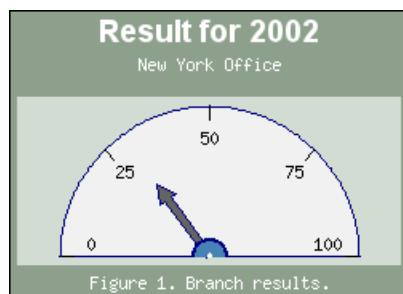
// Set display value for the odometer
$odo->needle->Set(30);

// Add the odometer to the graph
$graph->Add($odo);

// ... and finally stroke and stream the image back to the client
$graph->Stroke();

?>
```

**Figure 20.4. Adding titles and captions (`odotutex02.php`)** [`example_src/odotutex02.html`]



As can be see from the figure the margins will automatically adjust to hold the specified titles and captions.

Remember that text strings can have multiple lines by separating lines with a '\n' character. The following example shows how to add a more descriptive caption to the graph

### Example 20.3. Adding a multi line caption (`odotutex03.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_odo.php");

// Create a new odometer graph (width=250, height=200 pixels)
$graph = new OdoGraph(250,200);

// Setup titles
$graph->title->Set("Result for 2002");
$graph->title->SetColor("white");
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);
$graph->subtitle->Set("New York Office");
$graph->subtitle->SetColor("white");
$graph->caption->Set("Figure 1.This is a very, very\nlong text with multiples lines");
$graph->caption->SetColor("white");

// Now we need to create an odometer to add to the graph.
// By default the scale will be 0 to 100
$odo = new Odometer();

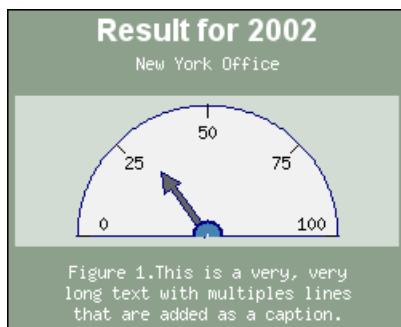
// Set display value for the odometer
$odo->needle->Set(30);

// Add the odometer to the graph
$graph->Add($odo);

// ... and finally stroke and stream the image back to the client
$graph->Stroke();

?>
```

**Figure 20.5. Adding a multi line caption (`odotutex03.php`)** [[example\\_src/odotutex03.html](#)]



## 20.2.4. Adjusting colors

Almost all areas in the graph have a customizable color. This means that for the Odometer graph itself it is possible to adjust

- Margin colors

- The plot area (the rectangular area where the plot(s) are added)
- The border line colors
- Shadow color

For example the following line changes the margin color of the graph

```
<?php
// Make the border 40% darker than normal "khaki"
$graph->SetMarginColor("khaki:0.6");
$graph->SetColor("khaki");
?>
```

For the odometer plot itself it is possible to adjust

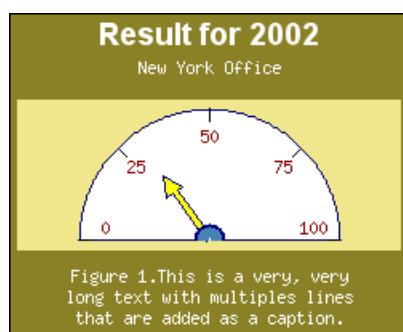
- Fascia color
- Scale label color (and font)
- Frame color (and width)
- Indicator color (the color of the odometer "needle")
- Adding colored segment for scale areas

For example the following lines would change the fascia, indicator and scale label color

```
<?php
$odo->SetColor("white");
$odo->scale->label->SetColor("blue");
$odo->needle->SetFillColor("yellow");
?>
```

Figure 20.6, “Changing colors (odotutex04.php)” shows the effect of adding the two code snippets above to our previous example.

**Figure 20.6. Changing colors (odotutex04.php) [example\_src/odotutex04.html]**



## 20.2.5. Changing indicator needle style

The indicator is accessed through the "\$needle" property of the plot. The indicator needle for the odometer can have six basic shapes, which are set by the method

- Needle::SetStyle(\$aStyle,\$aStyleOption=null)

The style is defined by one of the following symbolic constants.

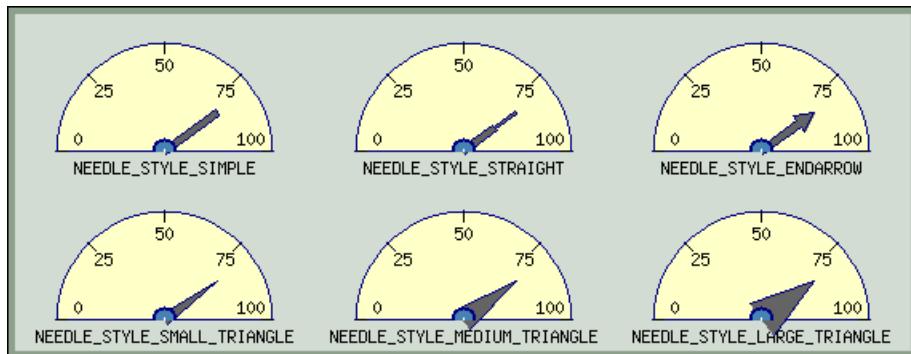
1. NEEDLE\_STYLE\_SIMPLE
2. NEEDLE\_STYLE\_STRAIGHT
3. NEEDLE\_STYLE\_ENDARROW, (default)
4. NEEDLE\_STYLE\_SMALL\_TRIANGLE
5. NEEDLE\_STYLE\_MEDIUM\_TRIANGLE
6. NEEDLE\_STYLE\_LARGE\_TRIANGLE
7. NEEDLE\_STYLE\_HUGE\_TRIANGLE

The following line would therefore create a plot with a "simple" indicator.

```
$odo->needle->SetStyle(NEEDLE_STYLE_SIMPLE);
```

The different styles are illustrated in Figure 20.7, “Possible shapes of the odometer indicator (`odotutex06.php`)”

**Figure 20.7. Possible shapes of the odometer indicator (`odotutex06.php`)**  
[[example\\_src/odotutex06.html](#)]



For the needle style "NEEDLE\_STYLE\_ENDARROW" it is also possible to adjust the ending arrow. This is done by specifying the particular end arrow size as the second parameter to `SetStyle()`, the `$aStyleOption` parameter.

To simplify the arrow style setting a number of predefined sizes are available. When choosing an arrow-head the width and length are specified in three different steps, small , medium and large.

The available end arrow styles are listed in ?? below

**Table 20.1. Available arrow size**

Symbolic name	Description
NEEDLE_ARROW_SS	Small width, small length
NEEDLE_ARROW_SM	Small width, small length
NEEDLE_ARROW_SL	Small width, large length
NEEDLE_ARROW_MS	Medium width, small length
NEEDLE_ARROW_MM	Medium width, small length
NEEDLE_ARROW_ML	Medium width, large length

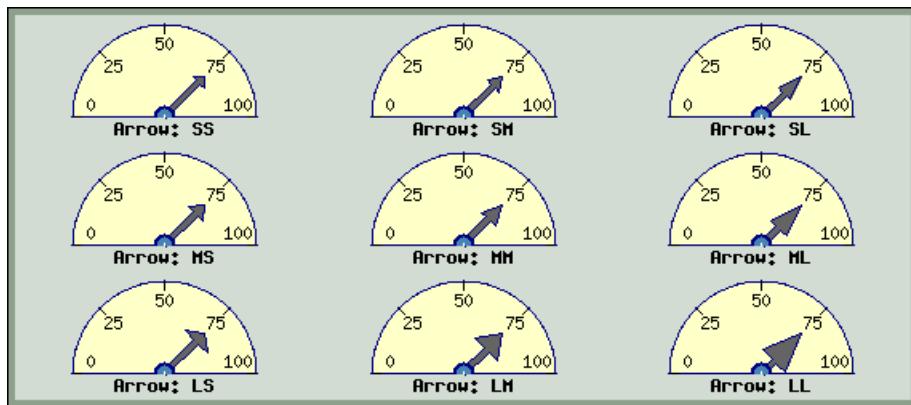
Symbolic name	Description
NEEDLE_ARROW_LS	Large width, small length
NEEDLE_ARROW_LM	Large width, medium length
NEEDLE_ARROW_LL	Large width, large length

The following code specifies a the largest available arrowhead

```
$odo->needle->SetStyle(NEEDLE_STYLE_ARROWHEAD, NEEDLE_ARROW_LL);
```

An illustration of the various arrow heads are given in Figure 20.8, “Illustration of various sizes of arrow heads (`odotutex07.php`)”

**Figure 20.8. Illustration of various sizes of arrow heads (`odotutex07.php`)**  
[`example_src/odotutex07.html`]



The basic length and weight of the needle is specified with the methods

- `Needle::SetLength($aLength)`
- `Needle::SetWeight($aWeight)`

The needle size is specified as fractions of the radius. So for example the following line wold create a needle halfway into the plot area

```
$odo->needle->SetLength(0.5);
```

The default size for a needles is 60% of the radius.

## 20.2.6. Adding drop shadows

It is possible to add drop shadows to both the overall graph s well as the indicator needle. To add a drop shadow to the overall graph the method is as usual

- `OdoGraph::SetShadow($aShadow=true,$aColor="gray@0.5",$aDx=4,$aDy=4)`

and to add a drop shadow to the indicator needle the method

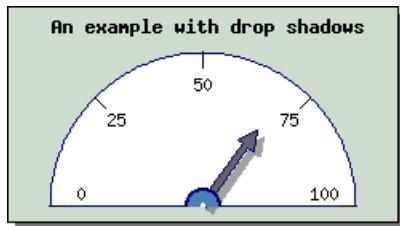
- `OdoNeedle::SetShadow($aShadow=true,$aColor="gray@0.5",$aDx=4,$aDy=4)`

must be used.

The following two lines add both types of drop shadows to an odometer graph with the result shown in ??

```
<?php
$graph->SetShadow();
...
$odo->needle->SetShadow();
?>
```

**Figure 20.9. Adding drop shadows (`odotutex08.php`) [example\_src/`odotutex08.html`]**



## 20.2.7. Changing sizes and margins

The size of the odometer is automatically adjusted to fit the available space in the image after margins for titles and captions have been taken into account. This means that normally it is not necessary to adjust the margins manually.

The odometer plot can be adjusted with the method

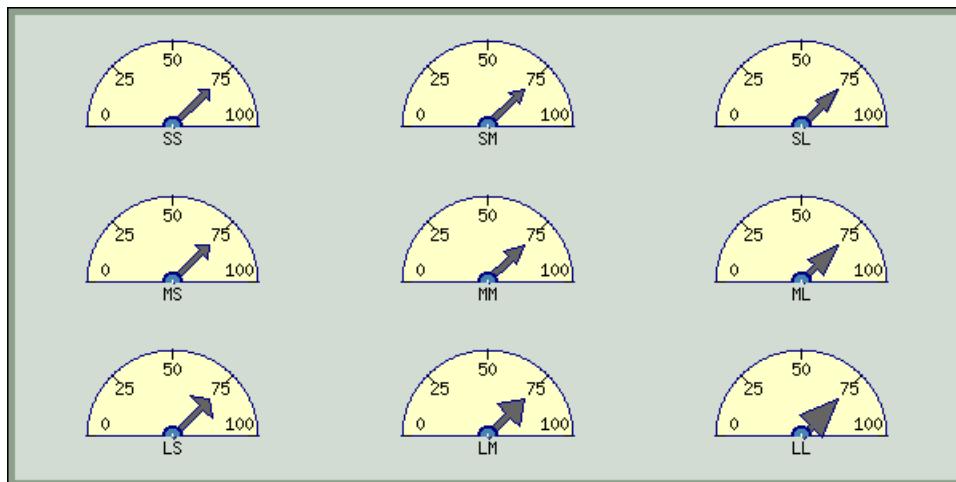
- `OdoPlot::SetSize($aSize)`

Each odometer also have a margin surrounding it. This is the amount of "space" around each individual odometer. The amount of space is adjusted by the method

`OdoPlot::SetMargin($aMargin)`

If we take the example in Figure 20.8, “Illustration of various sizes of arrow heads (`odotutex07.php`)” and increase the individual margin around all odometers they will become smaller (if we keep the original image size) and the result is shown in Figure 20.10, “Increasing the individual margins around the plots (`odotutex09.php`)”

**Figure 20.10. Increasing the individual margins around the plots (`odotutex09.php`) [example\_src/`odotutex09.html`]**



## 20.2.8. Adding color indications in the scale

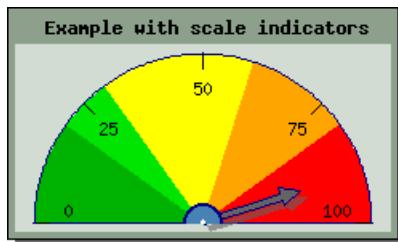
An odometer usually have some colored markings to indicate safe and dangerous values. The library allows the addition of any number of colored scale areas with one or several calls to the method

- `OdoPlot::AddIndication($aMin,$aMax,$aColor)`

This method allows the specification of a min and max value of the scale which will be given the background color specified as the third parameter.

The example in Figure 20.11, “Adding colored scale indicators (odotutex10.php) ” adds three different scale area with 6 different scale areas

**Figure 20.11. Adding colored scale indicators (odotutex10.php)  
[example\_src/odotutex10.html]**



## 20.2.9. Changing size of the non-colored center area

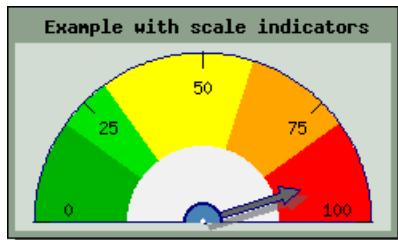
In Figure 20.11, “Adding colored scale indicators (odotutex10.php) ” it can be seen that by default the scale color indication goes all the way down to the base of the needle. It is however possible to adjust this to the colored area stops a bit above the base of the needle. The non colored area is adjusted with a call to the method

- `OdoPlot::SetCenterAreaWidth($aWidth)`

The width is specified as a fraction of the radius of the plot.

The following example in Figure 20.12, “Adjusting the non-colored center area (odotutex11.php) ”set the size of this base area to be 45% of the plot radius.

**Figure 20.12. Adjusting the non-colored center area (odotutex11.php)  
[example\_src/odotutex11.html]**



## 20.2.10. Using multiple indicator needles in one odometer

In all the previous example the odometer plots have all had a single indicator. It is however possible to add up to four indicators to each plot. This could for example be used to show the current value and a sliding 3-day average.

The extra needles are accessed through the properties

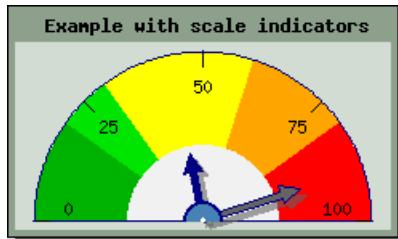
- `OdoPlot::needle2`
- `OdoPlot::needle3`
- `OdoPlot::needle4`

In order to enable one of these extra needles the method `OdoNeedle::Show()` has to be called. The value of the indicator can then be set as the following example shows.

```
$odo->needle2->Show();
$odo->needle2->Set(44);
```

By default all the needles will have the same color so the color should be manually changed to be able to differentiate among multiple needles. The example in Figure 20.13, “Adding a second scale indicator (`odotutex12.php`)” shows an example of this where we also have made the second indicator a bit shorter to further make it easier to read.

**Figure 20.13. Adding a second scale indicator (`odotutex12.php`)**  
[`example_src/odotutex12.html`]



## 20.2.11. Adding an odometer legend (label)

Most types of odometers have a legend in the middle of the meter usually indicating what the scale shows. The odometer have a specific text property to add text in that position.

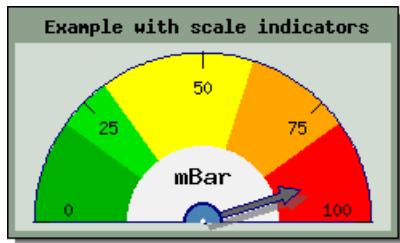
- `OdoPlot::label`

The following code adds the label "mBar" as the scale indicator and also adjusts the default font

```
$odo->label->Set("mBar");
$odo->label->SetFont(FF_FONT2,FS_BOLD);
```

The result of this can be seen in Figure 20.14, “Adding a scale legend by using the label property (`odotutex13.php`)”

**Figure 20.14. Adding a scale legend by using the label property (`odotutex13.php`) [[example\\_src/odotutex13.html](#)]**



## 20.2.12. Adjusting the width and color of the odometer frame (border)

The border of the odometer plot can be adjusted with a call to

- `OdoPlot::SetBorder($aColor, $aWidth=1)`

The following example shows the effect of making the odometer plot border thicker

```
=odotutex08.1|Adding a thicker border around the odometer plot"
```

## 20.3. Working with the odometer scale

In the previous section we ignored the actual scale in order to focus on the formatting options. Now we will do the (almost) opposite, only focus on how to manipulate the scale and the scale labels.

There are a number of properties that can be adjusted in the scale.

- The min and max values
- The fonts and for the scale
- The formatting of the scale values
- The start and end angles for the min and max values
- The tick interval
- The intervall of the scale labels

The scale is an instance of class `OdoScale` and is instantiated as an object that is accessed through the "`$scale`" property of the plot

- `OdoPlot::scale`

### 20.3.1. Adjusting the scale, ticks and labels

By default the scale runs between 0 (inclusively) and 100 (inclusively). trying to set the indicator needle to a value outside this range will result in an error.

The min an max values o the scale are specified with the method

- `OdoScale::Set($aMin, $aMax)`

The min/max values can be specified to any numeric value with the restriction that `$aMin <= $aMax`

To adjust which scale position should have tick marks and which tick marks should have a labels the following method is used.

- `OdoScale::SetTicks($aTickInterval, $aLabelInterval)`

The first argument specifies the tick interval, i.e. the interval tick marks will be drawn in the odometer. The second argument specifies that every n:th tick should have a label.

The following line specifies that every 20 unit of the scale should have a tick mark and that every second tick mark should have a label

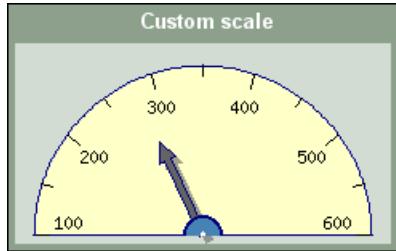
```
$odo->scale->SetTicks(20, 2)
```

### Tip

To make sure that both the min and max value of the scale gets a tick mark the tick multiplier should be an even divider to the scale range and the label multiplier should be a factor in the tick multiplier.

Figure 20.15, “(odotutex14.php)” shows and example of how to change the scale, tick and label positions.

**Figure 20.15. (odotutex14.php) [example\_src/odotutex14.html]**



To adjust the appearance of the tick marks in respect to line weight, length and color the following three methods can be used:

- `OdoScale::SetTickLength($aLength)`
- `OdoScale::SetTickWeight($aWeight)`
- `OdoScale::SetTickColor('red')`

### Note

The tick length of a major tick mark (which have labels) are 1.5 times that of a minor tick mark. This factor can not be adjusted.

## 20.3.2. Adjusting scale label format

The following section will show hot to customize the appearance of scale and labels.

The scale labels can, as all other texts in the library have their font and color adjusted. The text property for the labels is accessed through the property

- `OdoScale::label`

Since this is a standard Text object we can adjust the font and color as the following example shows

```
$odo->scale->label->SetFont(FF_FONT1,FS_BOLD);
$odo->scale->label->SetColor('navy');
```

The way the labels are formatted are controlled by a format string with the standard "printf()" syntax, e.g "%d" (as default) displays an integer and "%02f . 1" displays a label zero padded to two positions and with one decimal place.

The format string is applied by using the `OdoScale::SetLabelFormat()` method of the scale class as in

```
$odo->scale->SetLabelFormat(' %d %%');
```

The other aspect of the label formatting is exactly how the labels are positioned within the scale. By using the method

- `OdoScale::SetLabelPos($aPos)`

it is possible to specify the distance from the center to the label edge. By default this is 80% (i.e. specified as 0.8)

The following example shows the effect of applying some of these formatting options.

**Example 20.4. Adjusting the scale format (`odotutex15.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_odo.php");

// Create a new odometer graph (width=250, height=200 pixels)
$graph = new OdoGraph(250,170);

// Setup graph titles
$graph->title->Set('Custom formatting');
$graph->title->SetColor('white');
$graph->title->SetFont(FF_ARIAL,FS_BOLD);

// Add drop shadow for graph
$graph->SetShadow();

// Now we need to create an odometer to add to the graph.
$odo = new Odometer();
$odo->SetColor("lightgray:1.9");

// Setup the scale
$odo->scale->Set(100,600);
$odo->scale->SetTicks(50,2);
$odo->scale->SetTickColor('brown');
$odo->scale->SetTickLength(0.05);
$odo->scale->SetTickWeight(2);

$odo->scale->SetLabelPos(0.75);
$odo->scale->label->SetFont(FF_FONT1, FS_BOLD);
$odo->scale->label->SetColor('brown');
$odo->scale->label->SetFont(FF_ARIAL,FS_NORMAL,10);

// Setup a label with a degree mark
$odo->scale->SetLabelFormat('%dC'.SymChar::Get('degree')));

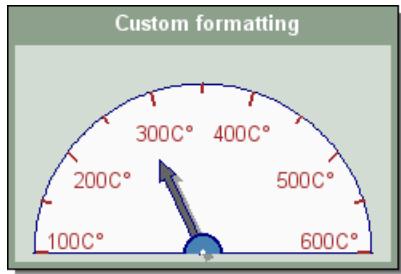
// Set display value for the odometer
$odo->needle->Set(280);

// Add drop shadow for needle
$odo->needle->SetShadow();

// Add the odometer to the graph
$graph->Add($odo);

// ... and finally stroke and stream the image back to the browser
$graph->Stroke();
?>
```

**Figure 20.16. Adjusting the scale format (`odotutex15.php`) [`example_src/odotutex15.html`]**



### Note

As can be seen in Figure 20.16, “Adjusting the scale format (`odotutex15.php`) ” we have used the utility class `SymChar` in order to get hold of the extended character that represents the degree mark. Remember that in order to get the extended character it is necessary to use a TTF scale.

### 20.3.3. Adjusting the start and stop angle for the scale

By default the scale labels start at 180 degree (9'a clock) and ends at 0 degree (3'a clock). To offer full flexibility this can be adjusted.

The method used for this is

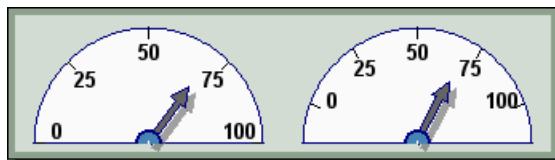
- `OdoScale::SetAngle($aStart, $aEnd)`

the angle is counted clockwise from the 6'a clock position. This means that the normal half odometer has a start angle of 90 degrees and an end angle of 270 degrees.

For full circle odometers the standard start angle is 40 degrees and the end angle is 320. This leaves a sector of 80 degrees free at the bottom of the odometer.

For example we could decide that the scale will start and stop 20 degrees above the horizontal line. The following example shows two odometer plots. The left uses the default start and end angle while the right odometer plot have adjusted start and end angles.

**Figure 20.17. Adjusting start and end angles of scale (`odotutex16.php`) [`example_src/odotutex16.html`]**



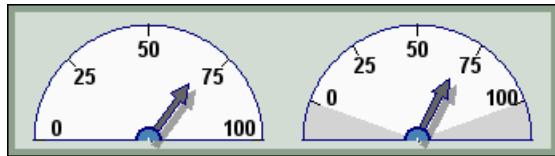
To further emphasize that the scale below the start and stop angles are "invalid" we could add a scale indicator in some gray:ish color that would signal that the area below 0 and 100 is not a valid scale area. We can do this by setting the range of the scale indicator to values outside the scale that would correspond to the horizontal line.

In Figure 20.17, “Adjusting start and end angles of scale (`odotutex16.php`) ” we can accomplish this by adding the following two lines to the second odometer

```
$odo2->AddIndication(-15,0,'lightgray');
$odo2->AddIndication(100,115,'lightgray');
```

The result of this is shown in Figure 20.18, “Adding gray areas below the min and max scale values (odotutex16.1.php)”

**Figure 20.18. Adding gray areas below the min and max scale values (odotutex16.1.php) [example\_src/odotutex16.1.html]**



## 20.4. Adding and positioning multiple odometers to a graph

The library supports unlimited number of odometers plots in a single graph. They can either be positioned manually within the graph by specifying absolute coordinates for the centers of the odometers or it can be done in a semi automatically way to avoid the hassle of calculating exact pixels positions.

### Tip

Remember that there is a caption property of the odometer plot as well as the main graph class and it is suitable to add a separate caption text to every odometer to separate them.

As have been shown in the previous example adding several odometer to the same graph is simply a matter of creating multiple instances of class `Odometer` and then adding them to the graph by calling the `OdoGraph::Add()` method.

### 20.4.1. Manual positioning

To manually position the odometer plot the following method is used

- `OdoPlot::SetPos($ax, $ay)`

Specifies the position of the odometer plot in either absolute coordinates (>1) or as fraction of the width/height when specified as fractions (in range (0.0 to 1.0)

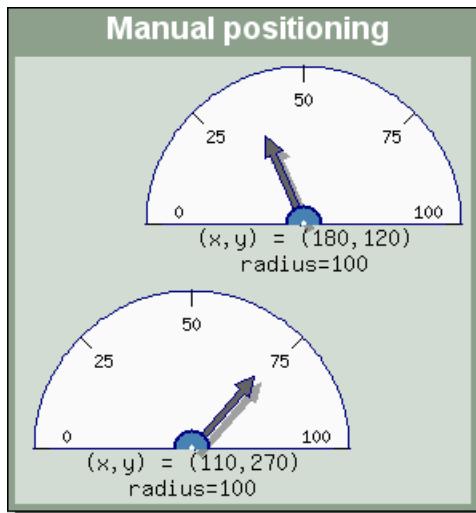
### Note

There is a minor difference when using odometer captions and specifying the y coordinate using fractions in that the y-coordinate and the radius will automatically adjust to the height of the caption size. This guarantees that the total image (odometer+caption) will always occupy the same total height in the image regardless of the size of the caption.

The top left corner of the graph is (0,0)

The following example manually positions two odometer plots in the graph, one at the top right corner and one in the bottom left corner.

**Figure 20.19. Manually specifying the position of odometer plots (`odotutex17.php`) [[example\\_src/odotutex17.html](#)]**



## 20.4.2. Using layout classes

### Note

Those familiar with Java AWT classes will recognize this concept

Normally we don't want to have to calculate the absolute x and y coordinates when positioning odometers. A much better concept would be to just tell the library to position three odometer horizontally or perhaps vertically without us having to figure out the exact coordinates our self.

This is where layout classes come in handy.

There are two types of layout horizontal and vertical. To specify that two odometers should be positioned side by side (horizontal) first a new layout object is created and then the odometer plots are added as object within the layout class. Later on when the object are put to the image the horizontal layout class will take all its objects and spread them out evenly along a horizontal line. The corresponding applies to the vertical layout class with the obvious change in direction.

The layout object are added to the graph in exactly the same was as odometers, by calling the `OdoGraph::Add()` method.

The following line would create a horizontal line of odometer plots

```
<?php
$rowl = new LayoutHor(array($odo1, $odo2, $odo3));
$graph->Add($rowl);
?>
```

If we instead wanted the odometer stacked on top of each other the lines above would have to be changed to

```
<?php
$rowl = new LayoutVert(array($odo1, $odo2, $odo3));
$graph->Add($rowl);
```

```
?>
```

There is no limit to how many object can be added to a layout class.

If it was only possible to add odometers in the creation of layout classes it would be of limited use. The real power of layout classes is that they can be combined.

So for example if we wanted two odometers in the top row and three in the bottom row we can think of this as first creating two horizontal layout object which are then in there turn layout out vertically.

The following code shows how this could be done

```
<?php
$row1 = new LayoutHor(array($odo1, $odo2));
$row2 = new LayoutHor(array($odo3, $odo4, $odo5));

// Combine the two rows in a column
$coll = new LayoutVert(array($row1, $row2));

// The image will now have 5 odometers!
$graph->Add($coll);
?>
```

The following code shows a full example on how this is done

**Example 20.5. Using layout classes for automatic positioning (`odotutex18.php`)**

```

<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_odo.php");

// Create a new odometer graph
$graph = new OdoGraph(500,180);

$odo = array();

// Now we need to create an odometer to add to the graph.
for($i=0; $i < 5; ++$i) {
 $odo[$i] = new Odometer();
 $odo[$i]->SetColor('lightgray:1.9');
 $odo[$i]->needle->Set(10+$i*17);
 $odo[$i]->needle->SetShadow();
 if($i < 2)
 $fsize = 10;
 else
 $fsize = 8;
 $odo[$i]->scale->label->SetFont(FF_ARIAL,FS_NORMAL,$fsize);
 $odo[$i]->AddIndication(92,100,'red');
 $odo[$i]->AddIndication(80,92,'orange');
 $odo[$i]->AddIndication(60,80,'yellow');
}

// Create the layout
$row1 = new LayoutHor(array($odo[0],$odo[1]));
$row2 = new LayoutHor(array($odo[2],$odo[3],$odo[4]));
$coll = new LayoutVert(array($row1,$row2));

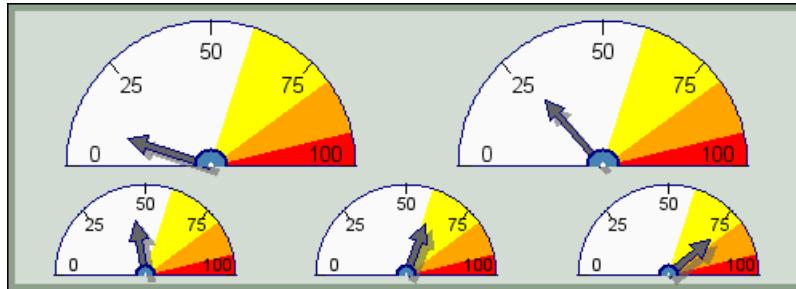
// Add the odometer to the graph
$graph->Add($coll);

// ... and finally stroke and stream the image back to the browser
$graph->Stroke();

?>

```

**Figure 20.20. Using layout classes for automatic positioning (`odotutex18.php`)**  
**[`example_src/odotutex18.html`]**



## 20.5. Adding icon and text objects to the graph

Odometer plots supports the ordinary way of adding icon and text objects to the graph.

### 20.5.1. Adding a text object

Text objects are added by first creating an instance of class `Text` for each text needed and then adding the text to the graph with the usual call to `MatrixGraph::Add()`.

A basic text will only require two additional lines of code

```
<?php
$txt = new Text('Simple string',20,20);
$graph->Add($txt);
?>
```

The following code snippet is slightly more complicated and will create a boxed text in the upper right corner of the graph.

```
<?php
// Add a boxed text
$txt = new Text();
$txt->SetFont(FF_ARIAL,FS_NORMAL,10);
$txt->Set("Arbitrary text\nnon a\nMatrix Plot");
$txt->SetParagraphAlign('center');
$txt->SetPos(0.95,0.15,'right');
$txt->SetBox('lightyellow');
$txt->SetShadow();
$graph->Add($txt);
?>
```

The snippet above adds a text at coordinates X=20, Y=20 using the default lower left corner as the text anchor point.

#### Note

To add a newline you must remember to use double-quotes to enclose the text otherwise the "\n" will only be interpreted literally.

#### Note

Remember that the "text align", as adjusted with `SetAlign()`, specifies the anchor point for the text, i.e. what part of the text is aligned with the specified position.

To add many text strings it is often useful to specify them in an array and then have a loop creating the text object and add the text array of all the created objects to the graph as the following short snippet shows.

```
<?php
//-----
// Add texts to the graph
//-----
$txts = array(
 array('Textstring one ...',$tx1,$ty1),
```

```

array('Textstring two ...', $tx2, $ty2),
array('Textstring three ...', $tx3, $ty3),

$n=count($txts);
$t=array();
for($i=0; $i < $n; ++$i){
 $t[$i] = new Text($txts[$i][0], $txts[$i][1], $txts[$i][2]);
 $t[$i]->SetFont(FF_ARIAL, FS_NORMAL, 12);
 $t[$i]->SetColor('brown');
 $t[$i]->SetAlign('center', 'top');
}
$graph->Add($t);
?>

```

## 20.5.2. Adding icons to the graph

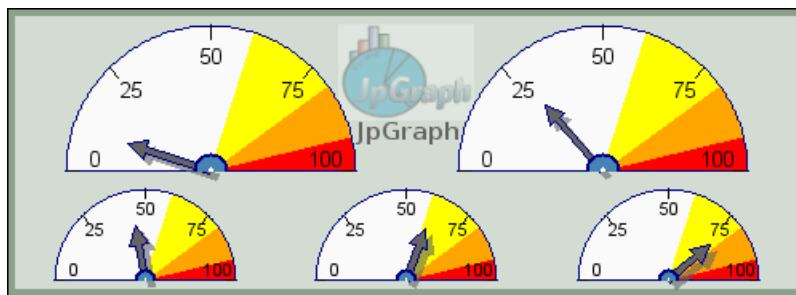
Icons are added as instances of class `IconPlot` to the graph (as usual with a call to `OdometerGraph::Add()`). This means that to use icons the library module "jpgraph\_iconplot.php" must first be included.

The following example shows how to add a small icon in the lower right corner of the graph. For more information on formatting icons that are added to a graph see Section 14.14, "Adding icons (and small images) to the graph"

### Caution

Since Odometer graphs doesn't have a ny concept of linear scale the position of icons can only be specified as absolute pixels or as fractions of the width/height.

**Figure 20.21. Adding an icon and text to a Odometer graph (`odotutex19.php`) [[example\\_src/odotutex19.html](#)]**



We also recall that it is possible to use country flags as icons by making use of the method

- `IconPlot::SetCountryFlag($aFlag, $aX=0, $aY=0, $aScale=1.0, $aMix=100, $aStdSize=3)`

## 20.5.3. Adding background images

In just the same way as for all other plots it is possible to add a background images, background gradients and a background flag to the graph. Since this has been discussed several time previously in the manual we only show a simple example here.

**Note**

Since odometer graphs does not have the concept of plot area the background image positioning alternative BGIMG\_FILLPLOT is the same as BGIMG\_FILLFRAME as we have used in this example below.

# Chapter 21. Windrose

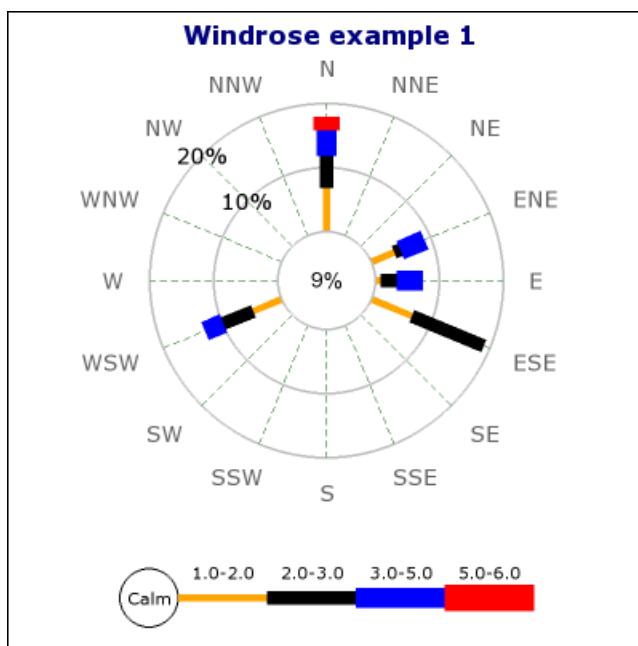
## 21.1. Introduction

### Note

This module is only available in the pro-version of the library.

Windrose plots are usually used to display values which are related to compass directions. For example it could be used to illustrate measured wind strengths over a time period in different directions. Figure 21.1, “An basic Windrose plot” Illustrates a basic example of a windrose graph with one windrose plot.

**Figure 21.1. An basic Windrose plot**



Windrose plots puts data in a number of ranges (or buckets). The number of data points in a certain bucket is visualized by the length of the particular bucket in that direction. The input data specifies the percentage of the overall data that belongs to one bucket in one specific direction. By default there are eight buckets with ranges shown in Table 21.1, “Default windrose buckets”. The default ranges that is shown in the legend is standard values used in displaying wind strengths. The values in the legend (for the buckets) is purely informational since the actual display is controlled by the input data that only specifies a bucket number and not any absolute data values. See more on how to specify input data in ??

**Table 21.1. Default windrose buckets**

Bucket nr	Range of values
0	[0,1]
1	[1,2]
2	[2,3]

Bucket nr	Range of values
3	[3,5]
4	[5,6]
5	[6,10]
6	[10,13.5]
7	[13.5,99]

## Note

The default fonts for Windrose graphs are TTF fonts. This means that to run the examples exactly as shown in this chapter TTF fonts must be installed. If TTF fonts are not available then the font needs to be changed to one of the built-in fonts.

Windrose plots also supports full image anti-aliasing to enhance the look and feel of the graph. The image in Figure 21.1, “An basic Windrose plot” is using anti-alias.

## Caution

To use anti alias TTF fonts must be enabled and used.

### 21.1.1. Features

The Windrose extension module supports all the usual library graph capabilities like frames, shadows, colors, titles, cache. Some of the additional Windrose graphs capabilities are:

- Enhanced anti-alias capability for smooth looking graphs
- Manual or automatic scale
- Supports customizable compass types which allow you to choose between 16,8 and 4 compass directions to be shown.
- Supports two types of basic graphs; Free direction graphs and compass bound graphs.
- Allows data to be specified with both compass direction and angles (in degrees)
- Fully customizable fonts everywhere in the plot
- Fully automatic adaptive positioning of the labels for the scale to avoid collision with data. This can also be turned off to allow manual positioning of the labels.
- Labels have extensive formatting capabilities and supports background colors, rounded boxes and arbitrary fonts.
- Supports multiple Windrose graphs on the same image
- Fully customizable size and positioning of each individual windrose graph.
- Size of windrose plots can be specified both as absolute pixels or as fraction of the graph size.
- Both manual and automatic color specifications

- Both automatic and manual scaling
- Intelligent positioning of angle labels with possibility for client specification of alignment type.
- Windrose graphs may have arbitrary texts blocks added to the graph

## 21.1.2. Different types of Windrose plots

There are two basic types of Windrose plots:

- **Compass type.**

This type of windrose plot has either 4,8 or 16 compass direction axis. Directions in input data is limited to one of the compass directions in the plot. The directions are specified as ordinal number of the direction axis.

The labels on the axis is by default short forms of the compass directions, for example, "E", "N", "W", "S". (See ?? for how to alter the default names of the compass directions). The ordinal number are counted anti-clockwise from East (3 o'clock).

- **Free type.**

This type of windrose plot has no pre-defined direction axis. Directions in input data can be arbitrary angles (or specified as compass directions e.g. "NW"). The 0-angle corresponds to the East direction (3 o'clock)

To specify the type of plot to create the following method is used

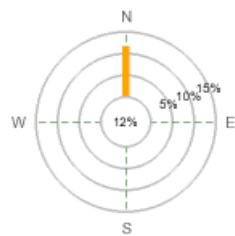
- `WindrosePlot::SetType($aDirection)`

`$aDirection` is one of the following 4 predefined constants

- `WINDROSE_TYPE4`, Compass type with the four core compass directions
- `WINDROSE_TYPE8`, Compass type with eight directions
- `WINDROSE_TYPE16`, Compass type with sixteen directions
- `WINDROSE_TYPEFREE`, Free type. Data directions can be arbitrary angles and there are no predefined labels nor any predefined axis.

The figures below illustrates the basic principle of the different types of windrose plots

**Figure 21.2. WINDROSE\_TYPE4**



**Figure 21.3. WINDROSE\_TYPE8**



**Figure 21.4. WINDROSE\_TYPE16****Figure 21.5. WINDROSE\_TYPEFREE**

The following line would specify a Windrose plot with all 16 compass angles

```
<?php
$wp = new WindrosePlot($data);
$wp->SetType(WINDROSE_TYPE16);
?>
```

### Note

If no type is specified the plot type will default to a compass type with 16 directions

## Specifying input data

Input data is in the form of a data array. Each entry in the array is itself an array which gives the data for one direction. The data should be interpreted as the percentage of the overall data that belongs to a certain direction and bucket. This also means that the total sum of all data entries (for all directions) must be less or equal to 100. It can however be less than 100 (indicating that some data is lost).

The first bucket, bucket 0, specifies the 0-value, i.e. what part of the data in that direction that has no measurement (e.g. it was wind still). Since wind still has no direction all specified bucket 0 are summarized and displayed in the inner circle.

By default the ranges displayed on the legend are:

0-1, 1-2, 2-3, 3-5, 5-6, 6-10, 10-13.5, 13-99

This means that the first element specifies the percentage of 0-1 reading the sum of these readings will be placed in the center of the windrose plot. The second element specifies the percentage of 1-2 readings, the third element the percentage of 2-3 readings and so on.

The direction is specified as the key for that element. The direction can be specified as either

- A string specifying any of the displayed compass directions, e.g. "N", "SW".

```
$data = array (
 array('E' => <east_bucket0>, <east_bucket1> ... <east_bucketN>),
 array('ESE' => <east-southeast_bucket0>, ... <east-southeast_bucketN>),
 ...
 array('NE' => <north-east_bucket0>, ... <north-east_bucketN>),
 array('ENE' => <east-north-east_bucket0>, ... <east-north-east_bucketN>)
);
```

- An ordinal number (integer) in the range 0-15 for regular plots which indicates the axis counting anti-clockwise from East. Please note that only displayed axis count. Ordinal number zero specifies the east direction

```
$data = array (array(0 => <east_bucket0>, <east_bucket1> ... <east_bucketN>),
 array(1 => <east_southeast_bucket0>, ... <east_southeast_bucketN>
 ...
 array(2 => <north_east_bucket0>, ... <north_east_bucketN>),
 array(3 => <east_north_east_bucket0>, ... <east_north_east_bucketN>
```

- A string indicating a number, e.g. '34.5', please note that if the angle is a fraction it must be specified as a string. The library will then automatically recognize that as a valid angle and treat it as expected. (The reason for this is that floating point numbers can't be used as keys in an associative array.)

```
$data = array (array('12.5' => <bucket0>, <bucket1> ... <bucketN>),
 array('22.1' => <bucket0>, ... <bucketN>),
 ...
```

Within the same data set it is also possible to mix the styles.

## Caution

The library makes no data validation apart from checking that the overall sum of the buckets can not be > 100%. It is up to the client to make sure that the data specified makes sense.

If you specify a direction in the data for a compass directions that is not displayed when using a regular Windrose plot you will get an error message to that effect.

Below are some examples of how to specify the data which should clarify how this works

### Example 21.1. Examples of input data for compass (regular) Windrose plots

*(Assuming we have all 16 axis displayed)*

```
$data = array('N' => array(2,5,6));
```

This data array specifies reading in only one direction, "North", the number of 0-readings are 2 percent, number of 0-1 readings are 5 percent and the number of 1-2 readings are 6 percent.

```
$data = array(1 => array(2,5,6), 3 => array(6,3), 'NW' => array(3,2,2,2))
```

This data array specifies readings in three directions. As shown it is possible to mix both ordinal number for the axis as well as the symbolic direction name. However, it is probably best to stick with one method at a time for clarity.

### Example 21.2. Examples of input data for free Windrose plots

What is special with the free type is that angles are no longer restricted to only the 16 compass directions but arbitrary directions as well.

```
$data = array(10 => array(2,5,6), 24 => array(6,3), 137 => array(3,2,2,2))
```

This data array specifies readings in three directions, 10 degrees, 24 degrees and 137 degrees.

```
$data = array('21.6' => array(2,5,6), 'N' => array(6,3), 137 => array(3,2,2,2))
```

This data array specifies readings in three directions, 21.5 degrees, 'North' (or 90 degrees) and finally 137 degrees. Please note that in order to specify a fraction for an angle we must specify the direction as a string value, i.e. '21.6'. This will then be handled automatically by the library.

## 21.2. Creating and formatting basic Windrose graphs

Creation of windrose graphs first requires the inclusion of the library extension module "jpgraph\_windrose.php"

The creation of Windrose graphs otherwise follows the traditional steps in the library of creating a graph and then adding one or several windrose plots to the canvas.

1. Create a basic canvas graph as an instance of `class WindroseGraph`
2. Create an instance of one or several windrose plots as instances of `class WindrosePlot`, set up the scale and appearance and then add them to the graph canvas.
3. Send back the graph to the client with a call to `WindroseGraph::Stroke()`. As usual this can be used to either send back the graph to the client (e.g.browser) or write the graph to a file by specifying a filename as the first argument to `WindroseGraph::Stroke()`.

The example in Figure 21.6, “A basic 16 direction windrose graph (`windrose_ex0.php`)” show a windrose graphs using just the default values for all parameters.

### Example 21.3. A basic 16 direction windrose graph (`windrose_ex0.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

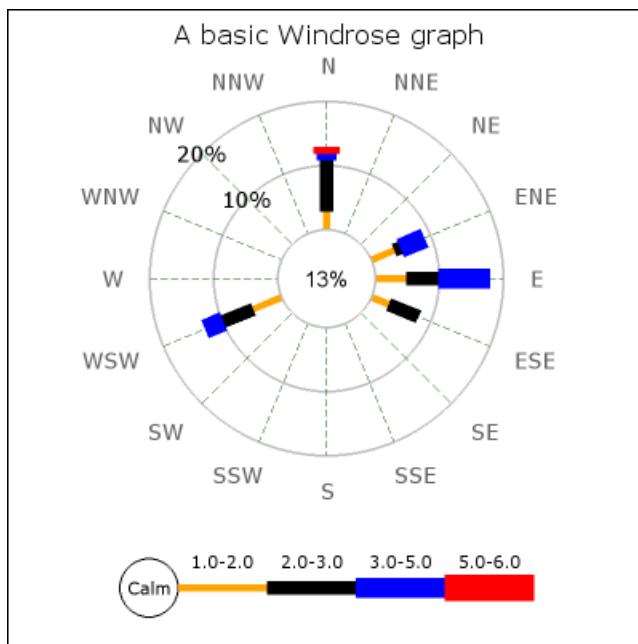
// Data can be specified using both ordinal index of the axis
// as well as the direction label
$data = array(
 0 => array(5,5,5,8),
 1 => array(3,4,1,4),
 'WSW' => array(1,5,5,3),
 'N' => array(2,3,8,1,1),
 15 => array(2,3,5));

// First create a new windrose graph with a title
$graph = new WindroseGraph(400,400);
$graph->title->Set('A basic Windrose graph');

// Create the windrose plot.
$wp = new WindrosePlot($data);

// Add and send back to browser
$graph->Add($wp);
$graph->Stroke();
?>
```

**Figure 21.6. A basic 16 direction windrose graph (`windrose_ex0.php`) [[example\\_src/windrose\\_ex0.html](#)]**



In the same way as for other graph types one or several Windrose plots can be added and positioned freely in the Windrose graph by specifying the position as either absolute coordinates or as fractions of the width/height of the overall graph.

### Tip

An easier way to position Windrose plots is to use layout classes as described in Section 21.6, “Using layout classes to position Windrose plots”

Each instance of the `WindrosePlot` is added to the overall graph with a call to the standard method

- `WindroseGraph::Add($aObject)`

To avoid that all the plots collide in the middle the positioning and sizing methods as shown in the previous sections should be used.

The following script shows how two plots are displayed in the same graph.

**Example 21.4. Adding two windrose plots to the same graph  
(windrose\_2plots\_ex1.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

// Data can be specified using both ordinal index of axis as well
// as the direction label
$data = array(
 1 => array(10,10,13,7),
 2 => array(2,8,10),
 4 => array(1,12,22),
);

$data2 = array(
 4 => array(12,8,2,3),
 2 => array(5,4,4,5,2),
);

// Create a new small windrose graph
$graph = new WindroseGraph(660,400);
$graph->SetShadow();

$graph->title->Set('Two windrose plots in one graph');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);
$graph->subtitle->Set('(Using Box() for each plot)');

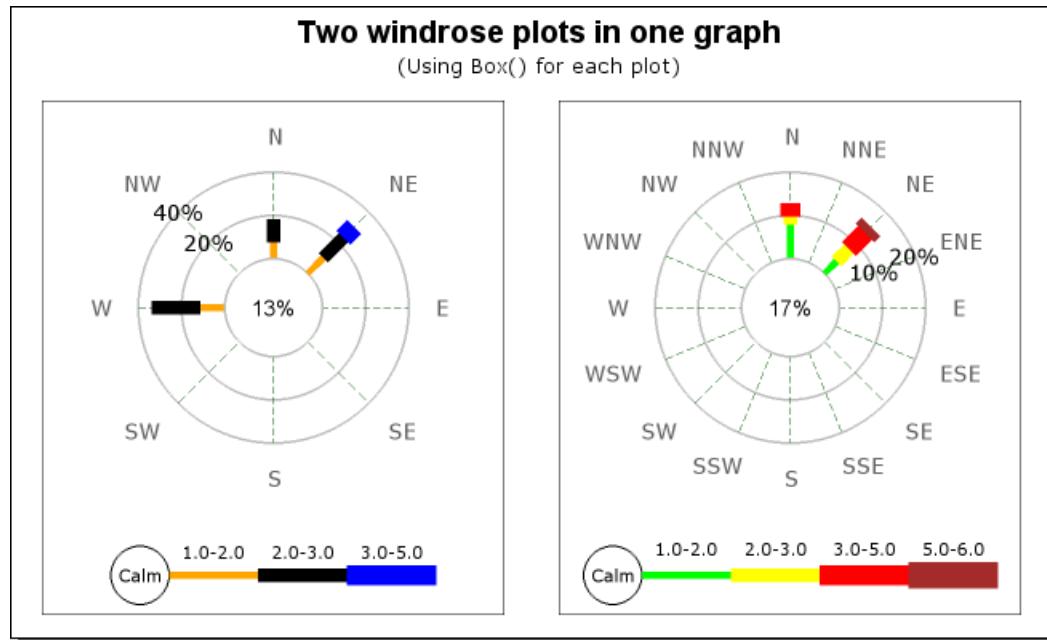
$wp = new WindrosePlot($data);
$wp->SetType(WINDROSE_TYPE8);
$wp->SetSize(0.42);
$wp->SetPos(0.25,0.55);
$wp->SetBox();

$wp2 = new WindrosePlot($data2);
$wp2->SetType(WINDROSE_TYPE16);
$wp2->SetSize(0.42);
$wp2->SetPos(0.74,0.55);
$wp2->SetBox();
$wp2->SetRangeColors(array('green','yellow','red','brown'));

$graph->Add($wp);
$graph->Add($wp2);

$graph->Stroke();
?>
```

**Figure 21.7. Adding two windrose plots to the same graph (windrose\_2plots\_ex1.php) [example\_src/windrose\_2plots\_ex1.html]**



## 21.2.1. Specifying the windrose scale

The scale is a property of each plot and is accessed through the instance variable `WindrosePlot::scale`. The scale consists of the epi-centric circles that radiate from the center of the Windrose plot. The step between each circle can be set manually or it can be done automatically by the library depending on the data values.

(In order to keep the flow of the text this section leaves the full examples to the example section.)

To manually set the scale of the circles the following method is used:

- `WindrosePlotScale::SetStepSize($aMax, $aStepSize=null)`

`$aMax`, The maximum scale value

`$aStepSize`, The step between each scale circle

By default the library tries multiples of 5:s and 2:s for a visually esthetic grid spacing.

Other examples of methods available to adjust the scale are

- `WindrosePlotScale::SetFont($aFontFamily, $aFontStyle=FS_NORMAL, $aFontSize=10)`

Specify font for all scale labels apart from the 0-label (in the center)

- `WindrosePlotScale::SetZFont($aFontFamily, $aFontStyle=FS_NORMAL, $aFontSize=10)`

Specify the font for the 0-label (in the center)

- `WindrosePlotScale::SetFontColor($aColor)`  
Specify font color for all scale labels apart from the 0-label (in the center)
- `WindrosePlotScale::SetZFontColor($aColor)`  
Specify the font for the 0-label (in the center)
- `WindrosePlotScale::SetLabelFillColor($aBkgColor,$aBorderColor=false)`  
Set the background color for the labels
- `WindrosePlotScale::SetLabelAlign($aAlign)`  
Set the label text alignment
- `WindrosePlotScale::SetLabelFormat($aFmt)`  
Set the label format. This is specified as a "printf( )" format string.
- `WindrosePlotScale::Hide($aFlg)`  
Hide the scale labels

Examples of how to use these methods are shown in the following sections.

## Specifying fonts and font colors

Fonts can be specified for both the labels on the scale as well as separately for the label in the center of the plot. The following code snippet shows how to do this (we assume we have already created a windrose plot called "windplot")

```
// Font and color for scale labels
$windplot->scale->SetFont(FF_VERDANA,FS_NORMAL,10);
$windplot->scale->SetFontColor('navy');

// Font and color for the center (Zero circle) label
$windplot->scale->SetZFont(FF_VERDANA,FS_NORMAL,10);
$windplot->scale->SetZFontColor('navy');
```

## Manually specifying the scale

By default the library automatically determines a scale range and step size taking into account the maximum value of data, size of the plot and the font size specified for the scale labels.

However it is possible to force a manual scale and step size with a call to `Set($a.MaxValue,$aStepSize)`. This call sets the maximum scale value as well as the step size. The step size is optional. If the step size is not specified it will be automatically determined.

```
// Specify both maximum value and a step size
$windplot->scale->Set(40,8);
...
// Specify just the maximum value
$windplot->scale->Set(40);
```

This feature is also illustrated in Figure 21.25, "(windrose\_ex7.php)" in the example part of this chapter.

## Note

The automatic scaling tries to first use step sizes as multiples of 5 and then multiples of 2 to achieve a suitable spacing of the scale lines.

## Note

It might be surprising to see that the scale steps may change when the label font is changed. However, this is by design. The reason for this is that there is only a certain amount of space available for the labels between each step. In order for large fonts to fit the space between the grids the step between each circle must be large enough.

## Specifying the label angle

The labels for the scale is placed along a line radiating from the center of the windrose plots. The angle for the labels can be either manually or automatically determined.

The labels for the scale is placed along one of the sixteen compass directions if you choose automatic positioning of the labels. By default the library tries to choose a direction with as little data around it as possible.

To manually specify the angle a call to `SetAngle($aAngle)` with the angle (in degrees) as argument.

The code snippet below shows how to do this

```
// Show the labels at 45 degrees angle
$windplot->scale->SetAngle(45);
```

To specify that the library should do this automatically (which is the default) the angle is specified as the string 'auto' as

```
// Let the library determine a good angle
$windplot->scale->SetAngle('auto');
```

## Specifying number formats for scale labels

In the same way as for other scales in the library it is possible to specify a "printf()" format string to adjust the appearance of the labels. By default the label format is the same as

```
$windplot->scale->SetLabelFormat('%d%%');
```

### 21.2.2. Specifying direction labels

With direction labels we mean the names of the compass directions surrounding the windrose plot. By default they are given the English short form for the 16 discrete compass named directions. This can also be localized (described in ??).

## Specifying fonts and font colors for the axis titles (direction labels)

Fonts for the axis titles are specified with (with the usual arguments)

- `WindrosePlot::SetFont()`
- `WindrosePlot::SetFontColor()`

as the following example shows

```
$windplot->SetFont(FF_TIMES,FS_BOLD,12);
$windplot->SetFontColor('darkgreen');
```

Which for example could give a Windrose plot as shown in Figure 21.22, “(windrose\_ex4.php)”

## Setting arbitrary text for data directions

When using the free direction windrose plots the default label is the direction for the data in degrees. This is fully customizable by using the following method.

- `WindrosePlot::SetLabels($aLabels)`

The input data to this method is an associative array where the keys are the direction and the content is the text to be displayed.

Please note that for regular Windrose plots, i.e. with only compass direction the label can not be changed. This is only available if the type of the windrose plot is `WINDROSE_TYPEFREE`.

The following example specifies a data point at 50 degrees and a text "Point\n#7315" as the label.

### Tip

Labels can contain "\n" which will be interpreted as a line break.

```
$data = array(50 => array(12,12,2));
$labels = array(50 => 'Point #7315');
...
$windplot = new WindrosePlot();
$windplot->SetLabels($labels);
```

An example of this can be seen in Figure 21.24, “(windrose\_ex6.php)”

## Adjusting the alignment and margin

There are two basic ways of adjusting the position of the compass labels around the plot:

- Adjusting the margin between the label and the outer plot circle
- Adjusting the anchor point of the labels.

The easiest way to explain this is to first imagine an invisible circle around the plot. The margins specifies how far from the outer scale circle this imaginative circle is placed.

In order to position the labels on that circle around the plot there are several possible ways to select what point of the text should be positioned on the circle. The easiest way is to select the center (both in X and Y directions) of the text. This works well if all labels are roughly the same size. If there are big difference between the smallest and largest label it might be necessary to have a very large margin to avoid the text to "collide" with the plot.

A better way is to determine the closest point of the text to the plot and then make that point lay on the circle. The closes point depends on what compass direction the label is on, for example for the "East" label the closes point is the middle of the left side of the text and for the "West" the closes point is the middle of the right side.

The library supports both way of positioning through the method

- `WindrosePlot::SetLabelPosition($aPosition)`

---

Valid values for \$aPosition are LBLPOSITION\_CENTER and LBLPOSITION\_EDGE .

The following code snippet shows how to specify both margin and position do this:

```
$windplot->SetLabelMargin(30);
$windplot->SetLabelPosition(LBLPOSITION_EDGE);
```

The red circle in ?? and ?? illustrates the "imaginative" circle on which the labels are placed. The distance from the out most plot circle to the label anchor point is the margin.

**Figure 21.8. Positioning with Figure 21.9. Positioning with  
LBLPOSITION\_CENTER                            LBLPOSITION\_EDGE**




---

An example of this can be seen in Figure 21.24, “(windrose\_ex6.php)”

## 21.3. Formatting the plot

### 21.3.1. Positioning the plot

The position of the graph can be controlled with the method

- `WindrosePlot::SetPos($aXPos, $aYPos)`

The position can be specified as either absolute position (in pixels) or as a fraction of width and height. The anchor for the position is the center of the windrose plot. For example, the following line centers the plot in the graph

```
$windplot->SetPos(0.5,0.5);
```

### 21.3.2. Specifying the size of the plot

In the same way as for the position the size of the windrose plot can be specified as either an absolute value or as a fraction of the minimum of width and height. The size specified refers to the diameter of the plot.

The size is controlled with the the method

- `WindrosePlot::SetSize($aDiameter)`

as in

```
$windplot->SetSize(200);
```

The code above sets the diameter of the plot to 200 pixels, setting the value to 0.7 would set the diameter to  $0.7 * \min(\text{WIDTH}, \text{HEIGHT})$  where width and height are the size of the graph.

### 21.3.3. Specifying the size of the Zero-circle (middle circle)

The size of the middle circle (the zero-range circle) can be adjusted with the method

- `WindrosePlot::SetZCircleSize($aDiameter)`  
\$aDiameter, The diameter can be specified as either an absolute integer value or as a fraction of the windrose plot diameter.

For example the following line sets the size to 1/3 of the windrose plot.

```
$windplot->SetZCircleSize(1/3);
```

### 21.3.4. Formatting the plot legend

The legend is displayed in a fixed position just below the windrose plot as can be seen in all the previous examples.

All legend methods is accessed through the legend property of the windrose plot as in

```
$windroseplot->legend->SetFont(...);
```

There are four basic attributes that can be customizable for the legend as described in the following paragraphs.

#### Specifying legend fonts

To allow for maximum flexibility font can be specified for both the numerical ranges as well as for the text under the legend. In order to avoid making the legend to wide it is often desirable to use a smaller font for the ranges and a slightly larger font for the actual text under the legend (as well as inside the "calm" circle - the indicator for bucket 0).

The fonts for the legend is adjusted by the following methods

- `SetFont()`, The text below the legend
- `SetLFont()`, The font for the labels
- `SetCFont()`, The font for the label inside the "calm" circle
- `SetFont()`, Sets all the font areas above the same font. Note that if this method is called after one of the individual settings previous specified (e.g. `SetCFont()`) they will be overwritten.

The colors for the fonts are specified in an analogue way by using the methods

- `SetFontColor()`
- `SetLFontColor()`
- `SetCFontColor()`

- SetFontColor()

Figure 21.10, “Windrose legend methods” explains visually how these methods are applied

**Figure 21.10. Windrose legend methods**

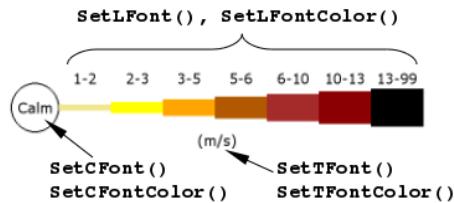
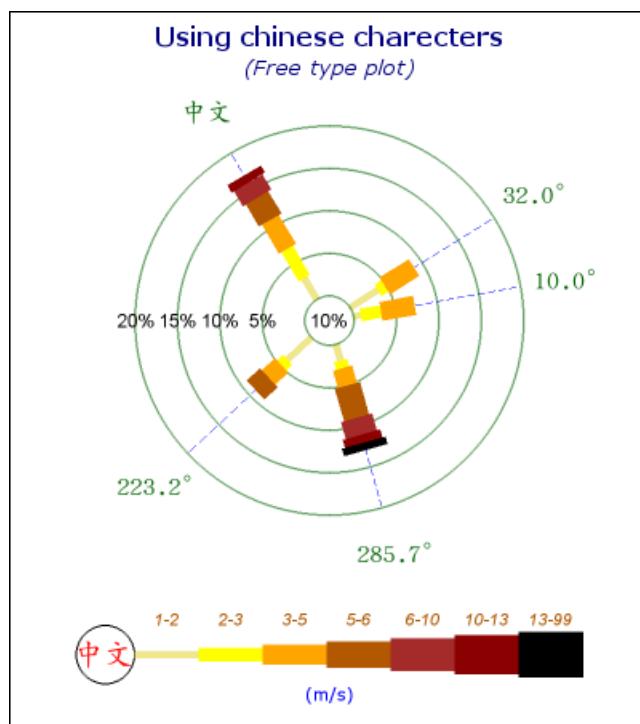


Figure 21.11, “Using chinese fonts (windrose\_ex6.1.php) ” illustrates how two fonts were used for the legend. For the circle we have use a Chinese font (The Kanji characters roughly means "Chinese language")

## Caution

The source file (as every source file in the library) is encoded in utf-8. For technical reasons some browsers might not render the Chinese characters correctly when the source file is viewed.

**Figure 21.11. Using chinese fonts (windrose\_ex6.1.php) [example\_src/windrose\_ex6.1.html]**



## Setting text and colors for the Zero-circle

The "Calm" circle can have its size, line weight and color adjusted. In addition (as was shown in Figure 21.11, “Using chinese fonts (windrose\_ex6.1.php) ”) specify any text within the circle. By default the text "Calm" is printed inside the circle. The following methods are used to adjust these properties.

- SetCircleWeight(\$aWeight)
- SetCircleRadius(\$aRadius)
- SetCircleColor(\$aColor)
- SetCircleText(\$aTxt)

## Specifying number formats for ranges

For the ranges it is possible to adjust how the numbers are formatted with a general format string (in the style of a `printf()` format string). This would allow, for example, to choose the number of decimal values to show in the legend. The format string is specified with a call to

- `Legend::SetFormat($aFmt)`.

The format will be applied to both the lower and upper range value. This means that the actual legend range text in the legend will be printed as [number format]-[number format].

The following example shows how to make sure the ranges are displayed as integers

```
$wp->legend->SetFormat(' %d');
```

## Specifying legend bottom text

Finally it is also possible to specify a text at the bottom of the legend with a call to

- `Legend::SetText($aTxt)`. The text will be displayed just underneath the legend as for example, shown in Figure 21.11, “Using chinese fonts (`windrose_ex6.1.php`)”

## Specifying the numeric values displayed for the ranges

The values displayed in the legend for the ranges can be adjusted with a call to

- `WindRosePlot::SetRanges($aRanges)`. By default the ranges are set to 0,1,2,3,5,6,10,13.5,99.0, this means that the ranges printed will be 1-2, 2-3, 3-5, ....

The following example shows how to change this to the ranges 0.000-0.001, 0.001-0.003, 0.003-0.005

```
$windplot->SetRanges(array(0.001,0.003,0.005));
$windplot->SetFormat('%.3f');
```

As can be seen the first range (bucket 0) is not really directly printed. It is by default indicated by the circle to the left in the legend.

In the example lines above we also changed the format to display the decimals. Without this the default is only to show one decimal and that would show all ranges as 0.0. An example of how this might look can be seen in figure 10 below.

## 21.3.5. Adjusting the grid colors

The colors of both the radial grid and the circular grid can be adjusted. This is done with the method

- `WindrosePlot::SetGridColor($aColor1,$aColor2)`

The first argument specifies the grid circle color and the second argument specifies the radial grid colors.

## 21.3.6. Adjusting the weight of the grid circles

To adjust the weight (thickness) of the grid lines (both circle and radial lines) the following method is used

- `WindrosePlot::SetGridWeight($aWeight1,$aWeight2)`

The first argument specifies the weight of the circle and the second the weight of the radial lines.

### Note

It is possible to have individual colors of the radial grid lines (and line style &weight) by using any of the methods

- `SetRadialColors()`
- `SetRadialStyles()`
- `SetRadialWeight()`

### Note

Due to technical and performance limitations circle weight can only be specified in the range [1,3] (inclusively).

## 21.3.7. Adding a box around the plot

Finally we remember that it is possible to add a rectangular box around the plot by using the method

- `WindrosePlot::SetBox($aColor='black', $aWeight=1, $aStyle='solid', $aShow=true)`

an example of this can be seen in Figure 21.7, “Adding two windrose plots to the same graph (`windrose_2plots_ex1.php`)”.

## 21.3.8. Adjusting the size of the center zero circle

The size of the center circle can be manually adjusted as either specified in absolute number of pixels or as fractions of the windrose plot (not the entire windrose graph). This is done with the method

- `WindrosePlot::SetZCircleSize($aSize)`

## 21.3.9. Localizing the default names for the compass directions

By default the compass cardinal directions are presented in English locale using standard denominations for the 16 compass directions.

Localizing these names are done with a call to the method

- `WindrosePlot::SetCompassLabels($aLabels)`

The order of the names for the directions should start with the corresponding name for "East" and then continue counter-clockwise all the way to "East North East". Adjusting the names for the compass directions means two things:

- If there is named direction in the data (as key) that key must now also be in localized form.

- The displayed names for the compass direction will be shown in localized form.

For example, the code snippet below shows how to localize the compass directions into Swedish.

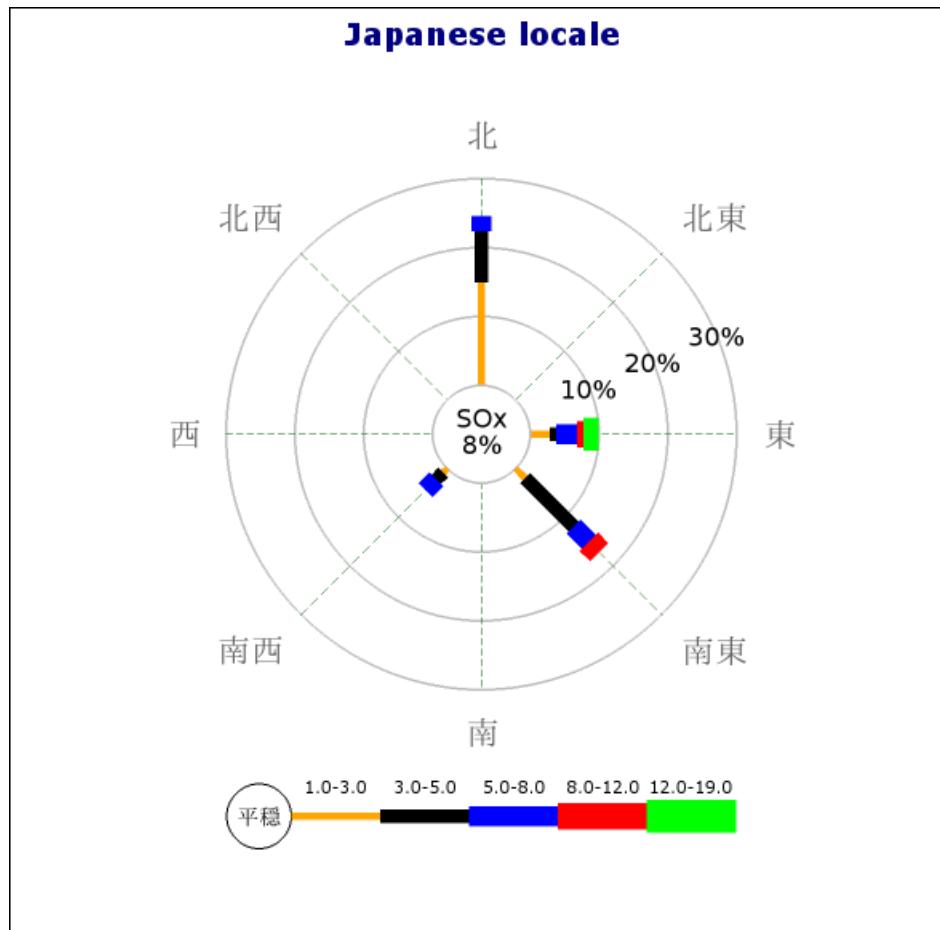
```
...
$data = array(
 'sso' => array(12,8,2,3),
 6 => array(5,4,4,5,4),
);
$se_CompassLbl =
array('O', 'ONO', 'NO', 'NNO', 'N', 'NNV', 'NV', 'VNV', 'V', 'VSV', 'SV', 'SSV', 'S', 'SSO', 'SO');
....
$windroseplot->SetCompassLabels($se_CompassLbl);
...
```

and the following shows how to make a localization into Japanese

## Caution

The source file (as every source file in the library) is encoded in utf-8. For technical reasons some browsers might not render the Japanese characters correctly when the source file is viewed.

**Figure 21.12. Japanese locale (windrose\_ex7.1.php) [example\_src/windrose\_ex7.1.html]**



## 21.3.10. Adjusting the formatting of the windrose plot legs

### Specifying range colors

To adjust the colors used for each range in the plot the method

```
WindrosePlot::SetRangeColors($aRangeColors)
```

The following example shows how to set a red-brown color scale

```
$rangeColors = array('khaki','yellow','orange','orange:0.7','brown','darkred','bla
$windplot->SetRangeColors($rangeColors);
```

### Specifying the width of the plot "legs"

The weight (width) of the legs used for each range in the plot is by default set to the range 2,4,6,8,10,12,14,16,18,20 pixels.

This can be adjusted with a call to the method

- `WindrosePlot::SetRangeWeights($aWeights)`

If fewer weights than ranges used are supplied the ranges will wrap-around. The following example shows how to set a larger difference in size for the legs between the ranges.

```
$windplot->SetRangeWeights(array(2,6,10,14,18));
```

## 21.4. Some more advanced formatting

### 21.4.1. Turning off Anti-aliasing

By default anti-aliasing is enabled. This has a speed penalty together with restrictions that no background images can be used in this version of the library.

If you want to use background images or improve performance anti-aliasing can be turned off with a call to the method

- `WindrosePlot::SetAntiAlias($aFlag)`

### 21.4.2. More on formatting scale-labels

Since scale labels are normal text element all normal text formatting can be used to enhance the look & feel of the scale labels.

### Adding background colors

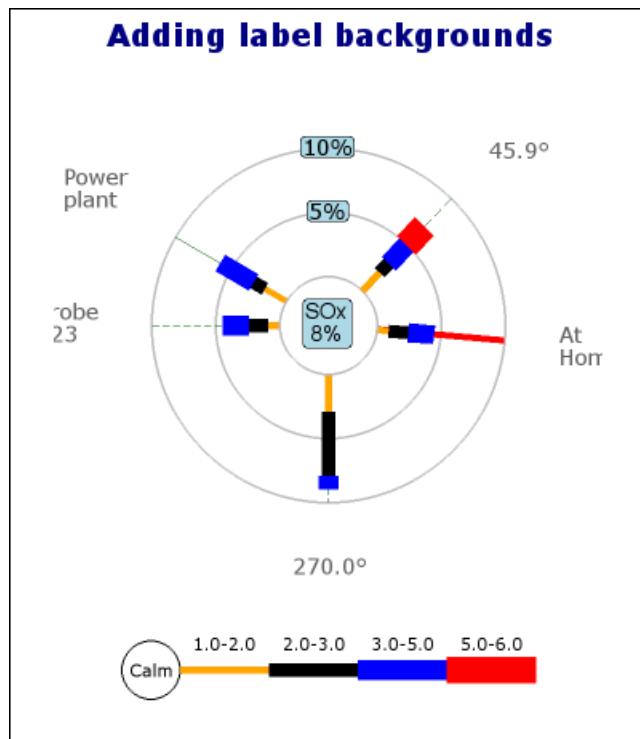
A simple enhancement to make the labels more visible is to add a box and a background color for the label. This is done with a call to the method

- `WindrosePlotScale::SetLabelFillColor($aFillColor, $aColor)`

The following code snippet shows how this could be done

```
$windplot->scale->SetLabelFillColor('lightblue','black');
```

**Figure 21.13. Adding label background (windrose\_ex8.1.php) [example\_src/windrose\_ex8.1.html]**



## Adjusting the alignment of the scale labels

By default the scale labels are centered on the intersection between the radial line and the grid circle. It is possible to also have the label positioned slightly outside the circular grid. This is all controlled by the method

- `WindrosePlotScale::SetLabelAlign($aAlign)`

Possible values for `$aAlign` are

- `LBLALIGN_CENTER`, The default. Center the label
- `LBLALIGN_TOP`, Optional. Position the label slightly outside the circular grid line.

## Specifying number formats

The final formatting option is to specify the number format of the label. This is specified as an ordinary `"printf()"` format string. The format is specified with a call to

- `WindrosePlotScale::SetLabelFormat($aFmt)`

For example the code snippet below makes the label display 1 decimal and adding a '%' sign.

```
$windroseplot->scale->SetLabelFormat('%.1f%%');
```

## Hiding the labels

Finally it is possible to completely hide the scale labels with a call to

- `WindrosePlotScale::Hide($aFlg=true)`

### 21.4.3. Formatting the legend

All the property for the legend is access through the legend property of the windrose plot as shown in the examples below.

#### Turning off the legend

By default the legend is shown for each plot. It can be turned off with a call to

- `LegendStyle::Hide()`

as the following code snippet shows.

```
$windroseplot->legend->Hide();
```

#### Adjusting the legend position

In version 1.0 of Windrose plot it is not possible to fully adjust the position of the plot legend. However, it is possible to manually adjust the margin between

- the Windrose plot and the legend
- ... the bottom of the box around a single plot and the legend

Both these margins are adjusted with a call to

- `Legend::SetMargin($aMarg,$aBottomMargin=5)`

For example by adding the following lines to your script

```
$windroseplot->legend->SetMargin(25);
```

By default this margin to the Windrose plot is 10 pixels and to the bottom of the plot box it is 5 pixels. Note that the legend positioning will automatically adjust for any increase in the font size for the compass directions to ensure that the legend will never collide with the labels when large fonts are used.

#### Adjusting the length of each legend window leg

By default the length of each legend leg is just long enough (with a little bit to spare) to cover the range labels just on top of the legend legs. It can never be set to smaller than that but it can be made larger with a call to

- `LegendStyle::SetLength($aLength)`

#### Adjusting how the range is displayed

By default the range labels is displayed so that the end label of bucket  $n$  is the same as the start label in bucket  $(n+1)$ . This behaviour can be controlled with the method

- `WindrosePlot::SetRangeStyle($aStyle)`

`$aStyle`, is one of two symbolic constants

1. `RANGE_OVERLAPPING`, (The default)
2. `RANGE_DISCRETE`, The range labels will be distinct

Figure 21.14, “Different legend label styles” shows the difference between the two legend styles

**Figure 21.14. Different legend label styles**



## 21.4.4. Highlighting specific compass directions

In order to emphasize certain directions it is possible to individually adjust the radial lines look & feel.

For example, assume that the direction  $134^\circ$  in a free plot is especially important. In order to emphasize this we can make the radial line, solid and red while maintaining a low profile on the other directions.

This is easily accomplished by the use of one or more of the three plot methods

- `SetRadialColors($aColors)`
- `SetRadialWeights($aWeights)`
- `SetRadialStyles($aStyles)`

These method all take arrays as argument. The arrays specifies what direction and what style to use. So, for example to make the "Southwest" grid red one would add the lines

```
<?php
colors = array('sw' => 'red');
$windplot->SetRadialColors($colors);
?>
```

As direction either the (localized) name can be used or the index/direction depending on if it is a free or regular type of windrose type.

Any direction which hasn't been explicitly specified in the array will use the default colors (As Specified by `SetGridColor()`) and the default weight (as specified by `SetGridWeight()`).

## 21.4.5. Including a graph in a PDF

This information assumes that the PDFlib is installed and available in the PHP setup being used. This very short paragraph is in no way a replacement for reading the full PDFlib documentation. It is just intended to give some ideas on how this can be done. This information is to large extent a repetition of what can be read in Appendix C, *FAQ*

Instead of stroking the image to a file or back to the browser you can get hold of the internal GD handle that represents the graph. This is done by specifying the constant `_IMG_HANDLER` as the file name in the call to `Stroke()`. This will tell the `Stroke()` method to return the GD handle.

In order to use the GD handle together with the PDFlib the image must be loaded into memory by a call to

```
pdf_open_memory_image($pdf, $im)
```

The following snippet shows how you can produce and send back a PDF page to the browser. Please note that this is just a code sketch and not final code

```
...
$graph = new WindroseGraph(...);

// Code to create the graph
//
//

// Put the image in a PDF page by first getting the GD handle
$im = $graph->Stroke(_IMG_HANDLER);

$pdf = pdf_new();
pdf_open_file($pdf, "");

// Convert the GD image to something the
// PDFlibrary knows how to handle
$pimg = pdf_open_memory_image($pdf, $im);

// Setup a basic PDF page
pdf_begin_page($pdf, 595, 842);
pdf_add_outline($pdf, "Page 1");
pdf_place_image($pdf, $pimg, 0, 500, 1);
pdf_close_image($pdf, $pimg);
pdf_end_page($pdf);
pdf_close($pdf);

// Prepare to output the PDF page
$buf = pdf_get_buffer($pdf);
$len = strlen($buf);

header("Content-type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=foo.pdf");

echo $buf;

pdf_delete($pdf);
...
```

## 21.4.6. Adding a background image

In the same way as with the other types of plots in the library there is a possibility to add a background image in the graph. However, due to limitations with the way the extended anti-alias in windroses is implemented it is not possible to use background images when this (default) behavior is enabled.

(It is technically possible to correct this limitations but the performance hit of doing this is completely unacceptable since that would require pixel-level manipulation directly in PHP which would be prohibitively CPU expensive).

To use background images the extended anti alias feature in windrose plots must be disabled with a call to `Windrosegraph::SetAntiAlias(false)`

In order to add a background image the normal method `SetBackgroundImage()` is used. The following code snippet shows how this is done.

```
$graph->SetBackgroundImage('mybackground.jpg', BGIMG_FILLFRAME);
```

The above example puts the image covering the entire graph area.

Remember that you can also adjust the mixing level of the background image with a call to `SetBackgroundImageMix($aMixLevel)`. This determines how much of the image is blended with the background color. An argument of 100 takes 100 percent of the image and a value of 0 is the same as not displaying the background image at all.

An example of using a background image is shown in Section 21.7.10, "Example 10"

See Section 14.15, "Adding images and country flags to the background of the graph" for more general information on background images.

### Tip

Remember that the method `Graph::SetBackgroundImageMix($aMix)` can be used to adjust the alpha blending of the background image

## 21.4.7. Adjusting the interpretation for ordinal keys in data

As previously stated the position for data values in the plot, (when the 'key' is specified as an integer), is the index of the displayed axis counted counter-clockwise from "East".

Another possibility which seems to be standard in some application of Windrose plots is to start counting clockwise from the first axis after the "North" axis. Depending on if the plots displays 16, 8 or 3 axis this first axis (with ordinal number 0) would then be either "NNE", "NE" or "E".

To adjust the interpretation the following method is used

- `WindrosePlot::SetDataKeyEncoding($aEncoding)`

Possible values for the parameter are

- `KEYENCODING_CLOCKWISE`, This setting will force the plot to interpret the key in the data as counted clockwise. The axis with 0-index is taken to be the first existing axis just to the right of "North" counted clockwise.
- `KEYENCODING_ANTICLOCKWISE`, This setting will force the plot to interpret the key in the data as counted counter-clockwise. The axis with the 0-index is the "East" axis.

By default the setting is `KEYENCODING_ANTICLOCKWISE`. ?? illustrates how the same data can be interpreted in two different way depending on this setting. The data in both figure a) and b) was given by the array

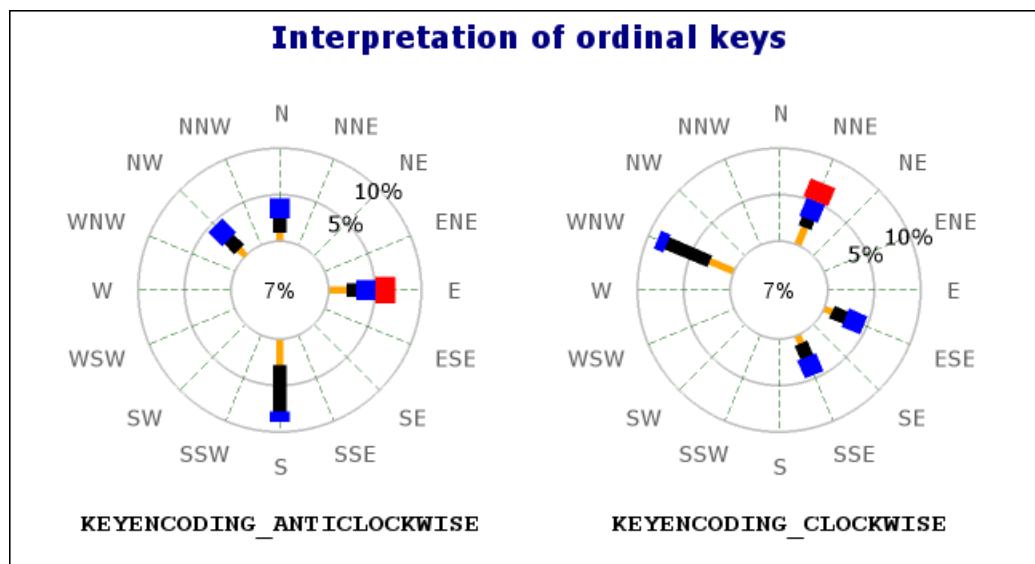
```
$data = array(
 0 => array(3,2,1,2,2),
 4 => array(1,1,1.5,2),
```

```

6 => array(1,1,1.5,2),
12 => array(2,3,5,1),
);

```

**Figure 21.15. Interpretation of ordinal keys (`windrose_ex9.1.php`) [`example_src/windrose_ex9.1.html`]**



## 21.5. Adding icon and text objects to the graph

Windrose plots supports the ordinary way of adding icon and text objects to the graph.

### 21.5.1. Adding a text object

Text objects are added by first creating an instance of class `Text` for each text needed and then adding the text to the graph with the usual call to `WindroseGraph::Add()`.

A basic text will only require two additional lines of code

```

<?php
$txt = new Text('Simple string',20,20);
$graph->Add($txt);
?>

```

The following code snippet is slightly more complicated and will create a boxed text in the upper right corner of the graph.

```

<?php
// Add a boxed text
$txt = new Text();
$txt->SetFont(FF_ARIAL,FS_NORMAL,10);
$txt->Set("Arbitrary text\non a\nWindrose Plot");
$txt->SetParagraphAlign('center');
$txt->SetPos(0.95,0.15,'right');
$txt->SetBox('lightyellow');
$txt->SetShadow();

```

```
$graph->Add($txt);
?>
```

An example of adding text to a graph can for example be seen in Figure 21.23, “(windrose\_ex5.php)”. The snippet above adds a text at coordinates X=20, Y=20 using the default lower left corner as the text anchor point.

## Note

To add a newline you must remember to use double-quotes to enclose the text otherwise the "\n" will only be interpreted literally.

## Note

Remember that the "text align", as adjusted with `SetAlign()`, specifies the anchor point for the text, i.e. what part of the text is aligned with the specified position.

To add many text strings it is often useful to specify them in an array and then have a loop creating the text object and add the text array of all the created objects to the graph as the following short snippet shows.

```
<?php
//-----
// Add texts to the graph
//-----
$txts = array(
 array('Textstring one ...', $tx1, $ty1),
 array('Textstring two ...', $tx2, $ty2),
 array('Textstring three ...', $tx3, $ty3),

$n=count($txts);
$t=array();
for($i=0; $i < $n; ++$i){
 $t[$i] = new Text($txts[$i][0], $txts[$i][1], $txts[$i][2]);
 $t[$i]->SetFont(FF_ARIAL, FS_NORMAL, 12);
 $t[$i]->SetColor('brown');
 $t[$i]->SetAlign('center', 'top');
}
$graph->Add($t);
?>
```

## 21.5.2. Adding icons to the graph

Icons are added as instances of class `IconPlot` to the graph (as usual with a call to `WindroseGraph::Add()`). This means that to use icons the library module "jpgraph\_iconplot.php" must first be included.

The following example shows how to add a small "tornado" icon in the upper left corner of the graph. For more information on formatting icons that are added to a graph see Section 14.14, “Adding icons (and small images) to the graph”

## Caution

Since Windrose graphs doesn't have a ny concept of linear scale the position of icons can only be specified as absolute pixels or fraction of the width/height.

**Example 21.5. Adding a "tornado" icon to the top left corner  
(windrose\_icon\_ex1.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');
require_once ('jpgraph/jpgraph_iconplot.php');

$data = array(
 0 => array(1,1,2.5,4),
 1 => array(3,4,1,4),
 'ws' => array(1,5,5,3),
 'N' => array(2,7,5,4,2),
 15 => array(2,7,12));

// First create a new windrose graph with a title
$graph = new WindroseGraph(400,400);

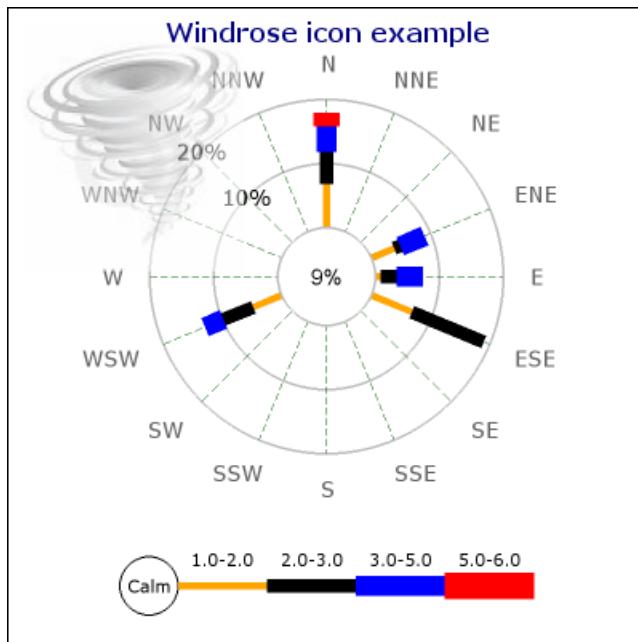
// Create an icon to be added to the graph
$icon = new IconPlot('tornado.jpg',10,10,1.3,50);
$icon->SetAnchor('left','top');
$graph->Add($icon);

// Setup title
$graph->title->Set('Windrose icon example');
$graph->title->SetFont(FF_VERDANA,FS_BOLD,12);
$graph->title->SetColor('navy');

// Create the windrose plot.
$wp = new WindrosePlot($data);

// Add to graph and send back to client
$graph->Add($wp);
$graph->Stroke();
?>
```

**Figure 21.16. Adding a "tornado" icon to the top left corner (windrose\_icon\_ex1.php) to the top [example\_src/windrose\_icon\_ex1.html]**



## 21.6. Using layout classes to position Windrose plots

Normally we don't want to have to calculate the absolute x and y coordinates when positioning multiple Windrose plots in graph as was done in Figure 21.7, “Adding two windrose plots to the same graph (windrose\_2plots\_ex1.php)”. A much better concept would be to just tell the library to position three windrose plots horizontally or vertically without having to figure out the exact coordinates our self.

This is where layout classes come in handy.

There are two types of layouts; horizontal and vertical. To specify that two windroses should be positioned side by side (horizontal) a new horizontal layout object is created and then the two windroses plots are added as object within the horizontal layout class. Later on when the objects are about to be stroked on the graph the horizontal layout class will take all its objects and spread them out evenly along a horizontal line depending on the individual size of each windrose. The same principle applies to the vertical layout class with the obvious change in direction.

The layout object are added to the graph in exactly the same was as odometers, by calling the `WindroseGraph::Add()` method.

The following line would create a horizontal line of three windrose plots

```
<?php
$graph = WindroseGraph($width,$height);

...
// Create three plots
```

```

$w1 = new WindrosePlot($data1);
$w1->SetSize(0.25);
$w2 = new WindrosePlot($data2);
$w2->SetSize(0.25);
$w3 = new WindrosePlot($data3);
$w3->SetSize(0.25);

// Position them even horizontally
$hor = new LayoutHor(array($w1, $w2, $w3));

// add and send back to the client
$graph->Add($hor);
$graph->Stroke();
?>

```

Since both horizontal en vertical layout can be combined it is possible to create almost any rectangular layout. For example to create a 4x4 graph first two horizontal layouts would be created and they would then be included in a vertical layout as the following principle code snippet shows.

```

<?php
$graph = WindroseGraph($width,$height);

...
// Create three plots
$w1 = new WindrosePlot($data1);
$w1->SetSize(0.24);
$w2 = new WindrosePlot($data2);
$w2->SetSize(0.24);
$w3 = new WindrosePlot($data3);
$w3->SetSize(0.24);
$w4 = new WindrosePlot($data4);
$w4->SetSize(0.24);

// Create the two rows
$hor1 = new LayoutHor(array($w1, $w2));
$hor2 = new LayoutHor(array($w3, $w4));

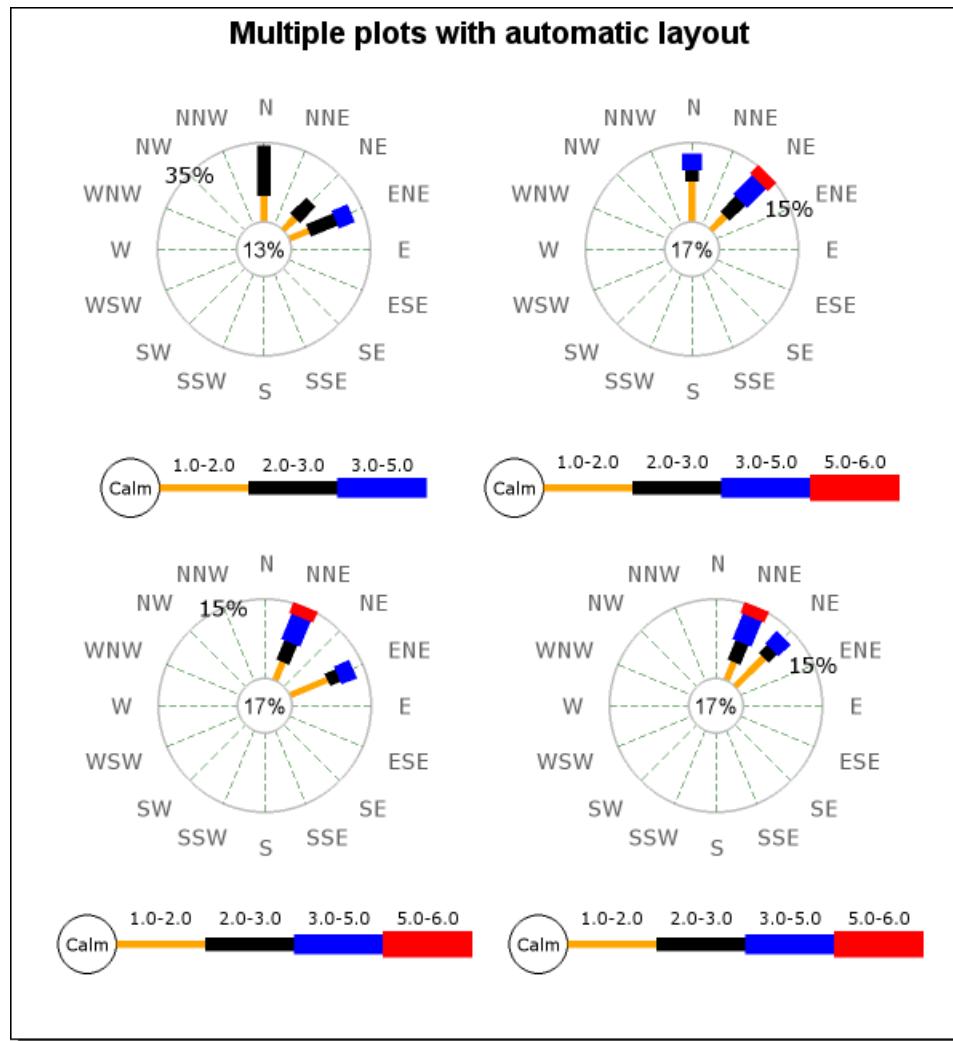
// positioned on top of each other
$vert = new LayoutVert(array($hor1, $hor2));

// add and send back to the client
$graph->Add($vert);
$graph->Stroke();
?>

```

An example of this is shown in Figure 21.17, “Using layout classes to position 4 windrose plots (windrose\_layout\_ex0.php)”

**Figure 21.17. Using layout classes to position 4 windrose plots (windrose\_layout\_ex0.php) [example\_src/windrose\_layout\_ex0.html]**



A slightly more complex example where some more formatting have been added is shown in Figure 21.18, “Using layout classes to position 5 windrose plots (windrose\_layout\_ex1.php)” where two larger plots are in the top row and three smaller plots are shown in the bottom row.

```

<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

// Some data for the five windrose plots
$data = array(
 array(
 1 => array(10,10,13,7),
 2 => array(2,8,10),
 4 => array(1,12,22)),
 array(
 4 => array(12,8,2,3),
 2 => array(5,4,4,5,2)),
 array(
 1 => array(12,8,2,3),
 3 => array(5,4,4,5,2)),
 array(
 2 => array(12,8,2,3),
 3 => array(5,4,4,5,2)),
 array(
 4 => array(12,8,2,3),
 6 => array(5,4,4,5,2))
);

// Legend range colors
$rangecolors = array('green','yellow','red','brown');

// Create a windrose graph with titles
$graph = new WindroseGraph(750,700);

$graph->title->Set('Multiple plots with automatic layout');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);

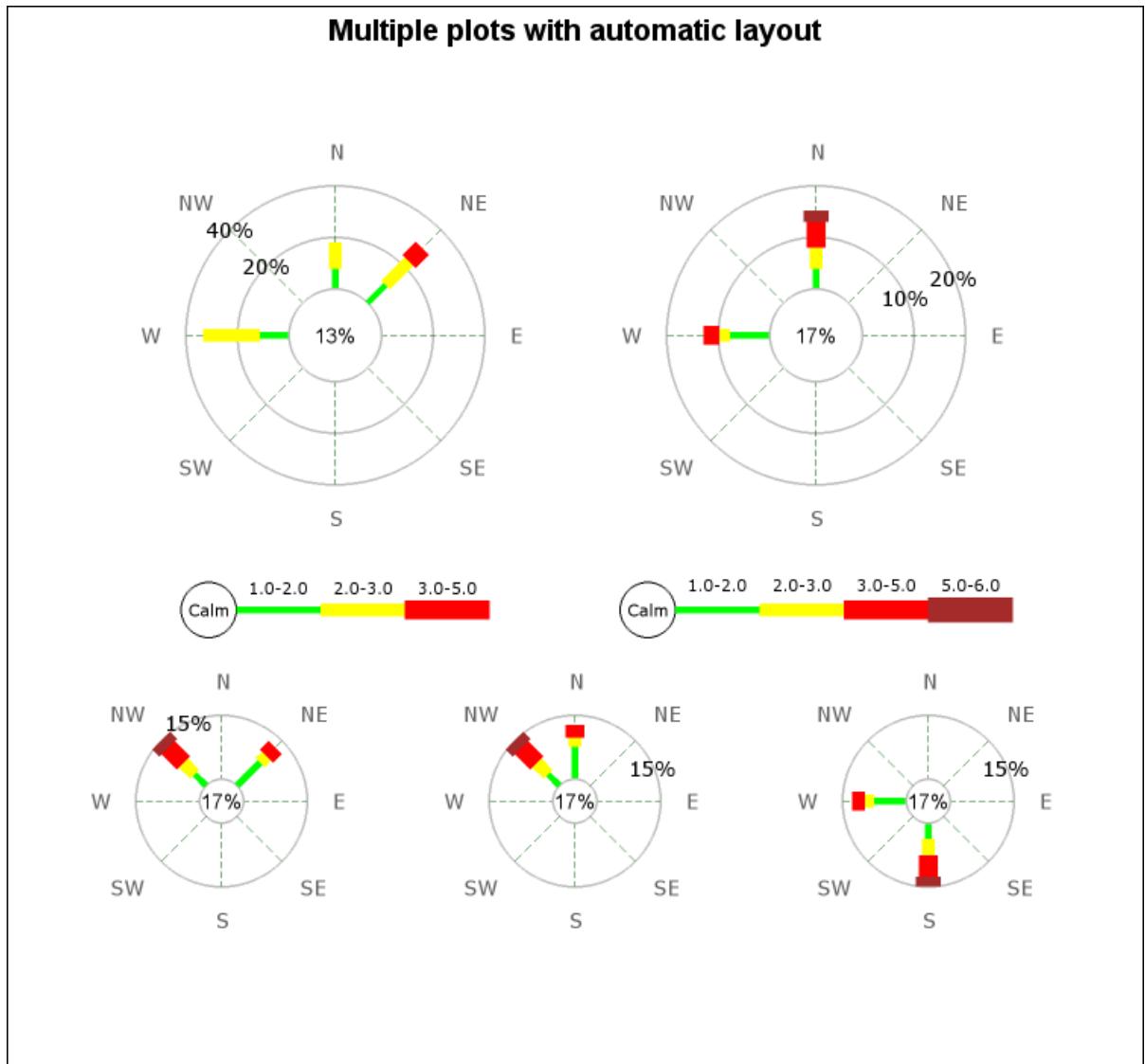
// Setup the individual windrose plots
$wp = array();
for($i=0; $i < 5; ++$i) {
 $wp[$i] = new WindrosePlot($data[$i]);
 $wp[$i]->SetType(WINDROSE_TYPE8);
 if($i < 2) {
 $wp[$i]->SetSize(0.28);
 }
 else {
 $wp[$i]->legend->Hide();
 $wp[$i]->SetSize(0.16);
 $wp[$i]->SetCenterSize(0.25);
 }
 $wp[$i]->SetRangeColors($rangecolors);
}

// Position with two rows. Two plots in top row and three plots in
// bottom row.
$h11 = new LayoutHor(array($wp[0], $wp[1]));
$h12 = new LayoutHor(array($wp[2], $wp[3], $wp[4]));
$v1 = new LayoutVert(array($h11, $h12));

$graph->Add($v1);
$graph->Stroke();
?>

```

**Figure 21.18. Using layout classes to position 5 windrose plots (windrose\_layout\_ex1.php) [example\_src/windrose\_layout\_ex1.html]**



## 21.7. Example section

The following section is a summary with a number of examples that illustrates the usage of the methods described in the previous sections.

### 21.7.1. Example 1

Objectives:

1. Showing a complete but elementary windrose plot using mostly default values.
2. Adding and formatting a title

3. Adding a legend text in order to show the units.
4. Showing how to specify data using both ordinal and textual representation of compass directions.

**Example 21.7. (windrose\_ex1.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

$data = array(
 0 => array(1,1,2.5,4),
 1 => array(3,4,1,4),
 'ws' => array(1,5,5,3),
 'N' => array(2,7,5,4,2),
 15 => array(2,7,12));

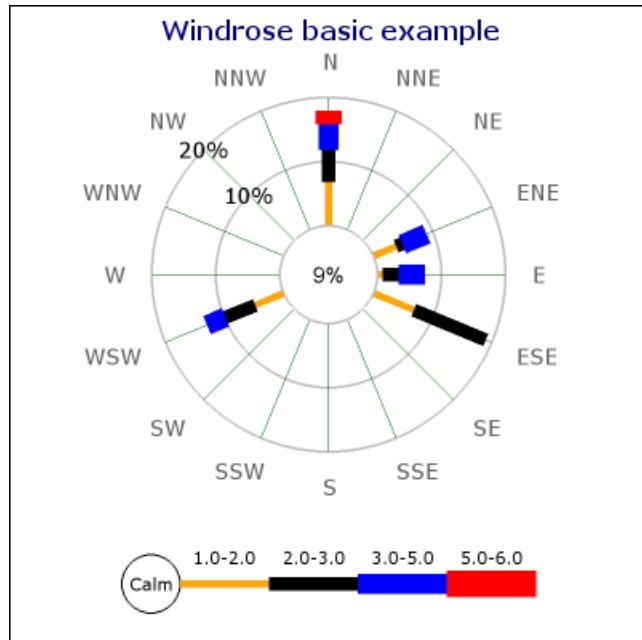
// First create a new windrose graph with a title
$graph = new WindroseGraph(400,400);

// Setup title
$graph->title->Set('Windrose basic example');
$graph->title->SetFont(FF_VERDANA,FS_BOLD,12);
$graph->title->SetColor('navy');

// Create the windrose plot.
$wp = new WindrosePlot($data);
$wp->SetRadialGridStyle('solid');
$graph->Add($wp);

// Send the graph to the browser
$graph->Stroke();
?>
```

**Figure 21.19.** (windrose\_ex1.php)  
[example\_src/windrose\_ex1.html]



## 21.7.2. Example 2

Objectives:

1. Showing how to customize the width of the windrose legs
2. Showing how to only display 8 compass axis
3. Showing how to set a specific size in pixels for the plot
4. Showing how to adjust the size of the center circle
5. Showing how to adjust scale label font &colors
6. Showing how to adjust the font for the compass directions

**Example 21.8. (windrose\_ex2.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

// Data can be specified using both ordinal index of the axis
// as well as the direction label
$data = array(
 0 => array(1,1,2.5,4),
 1 => array(3,4,1,4),
 3 => array(2,7,4,4,3),
 5 => array(2,7,1,2));

// First create a new windrose graph with a title
$graph = new WindroseGraph(400,400);

// Setup title
$graph->title->Set('Windrose example 2');
$graph->title->SetFont(FF_VERDANA,FS_BOLD,12);
$graph->title->SetColor('navy');

// Create the windrose plot.
$wp = new WindrosePlot($data);

// Make it have 8 compass direction
$wp->SetType(WINDROSE_TYPE8);

// Setup the weight of the laegs for the different ranges
$weights = array_fill(0,8,10);
$wp->SetRangeWeights($weights);

// Adjust the font and font color for scale labels
$wp->scale->SetFont(FF_TIMES,FS_NORMAL,11);
$wp->scale->SetFontColor('navy');

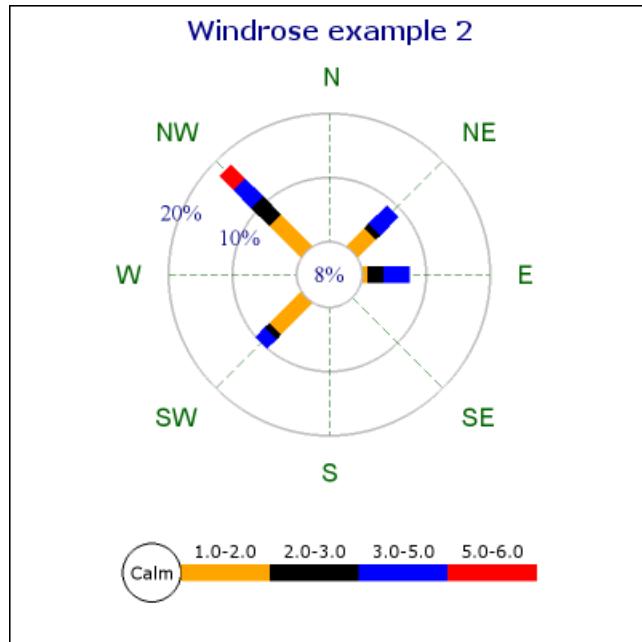
// Set the diametr for the plot to 160 pixels
$wp->SetSize(200);

// Set the size of the innermost center circle to 30% of the plot size
$wp->SetZCircleSize(0.2);

// Adjust the font and font color for compass directions
$wp->SetFont(FF_ARIAL,FS_NORMAL,12);
$wp->SetFontColor('darkgreen');

// Add and send back the graph to the client
$graph->Add($wp);
$graph->Stroke();
?>
```

**Figure 21.20.** (windrose\_ex2.php)  
[example\_src/  
windrose\_ex2.html]



### 21.7.3. Example 3

Objectives:

1. Showing how to add multiple plots to the same graph
2. Showing how to add a subtitle and a drop shadow to the graph
3. Showing how to change size and position for plots
4. Showing how to add legend text

```

"n" => array(1,1,2.5,4),
"ssw" => array(3,4,1,4),
"se" => array(2,7,4,4,3));

// Store the position and size data for each plot in an
// array to make it easier to create multiple plots.
// The format choosen for the layout data is
// (type,x-pos,y-pos,size, z-circle size)
$layout = array(
 array(WINDROSE_TYPE8,0.25,0.55,0.4,0.25),
 array(WINDROSE_TYPE16,0.75,0.55,0.4,0.25));

$legendtxt = array('(m/s) Station 7',(m/s) Station 12');

// First create a new windrose graph with a dropshadow
$graph = new WindroseGraph(600,350);
$graph->SetShadow('darkgray');

// Setup titles
$graph->title->Set('Windrose example 3');
$graph->title->SetFont(FF_VERDANA,FS_BOLD,12);
$graph->title->SetColor('navy');
$graph->subtitle->Set('(Multiple plots in the same graph)');
$graph->subtitle->SetFont(FF_VERDANA,FS_NORMAL,9);
$graph->subtitle->SetColor('navy');

// Create the two windrose plots.
for($i=0; $i < count($data); ++$i) {
 $wp[$i] = new WindrosePlot($data[$i]);

 // Make it have 8 compass direction
 $wp[$i]->SetType($layout[$i][0]);

 // Adjust the font and font color for scale labels
 $wp[$i]->scale->SetFont(FF_TIMES,FS_NORMAL,10);
 $wp[$i]->scale->SetFontColor('navy');

 // Set the position of the plot
 $wp[$i]->SetPos($layout[$i][1],$layout[$i][2]);

 // Set the diameter for the plot to 30% of the width of the graph pixels
 $wp[$i]->SetSize($layout[$i][3]);

 // Set the size of the innermost center circle to 30% of the plot size
 $wp[$i]->SetZCircleSize($layout[$i][4]);

 // Adjust the font and font color for compass directions
 $wp[$i]->SetFont(FF_ARIAL,FS_NORMAL,10);
 $wp[$i]->SetFontColor('darkgreen');

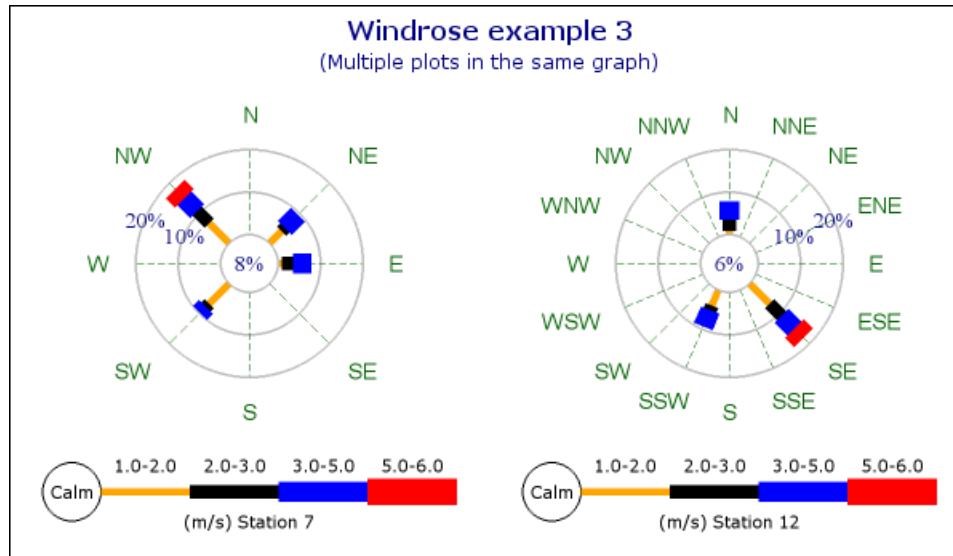
 // Add legend text
 $wp[$i]->legend->SetText($legendtxt[$i]);

 $graph->Add($wp[$i]);
}

// Send the graph to the browser
$graph->Stroke();
?>

```

**Figure 21.21.** (windrose\_ex3.php) [example\_src/windrose\_ex3.html]



## 21.7.4. Example 4

Objectives:

1. Showing how to add arbitrary paragraphs of text to a graph
2. Showing how to manually set the grid distance
3. Showing how to adjust the grid colors
4. Showing how to customize the format for the displayed ranges
5. Showing how to add arbitrary text to the center circle
6. Showing how to adjust the margin for the legend
7. Showing how to increase the margin to the compass directions

```

<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

// Data can be specified using both ordinal index of the axis
// as well as the direction label
$data = array(
 0 => array(1,1,2.5,4),
 1 => array(3,4,1,4),
 3 => array(2,7,4,4,3),
 5 => array(2,7,1,2));

// First create a new windrose graph with a title
$graph = new WindroseGraph(400,400);

// Setup title
$graph->title->Set('Windrose example 4');
$graph->title->SetFont(FF_VARDANA,FS_BOLD,12);
$graph->title->SetColor('navy');

// Create the windrose plot.
$wp = new WindrosePlot($data);

// Adjust the font and font color for scale labels
$wp->scale->SetFont(FF_TIMES,FS_NORMAL,11);
$wp->scale->SetFontColor('navy');

// Set the diameter and position for plot
$wp->SetSize(190);

// Set the size of the innermost center circle to 40% of the plot size
// Note that we can have the automatic "Zero" sum appear in our custom text
$wp->SetZCircleSize(0.38);
$wp->scale->SetZeroLabel("Station 12\n(Calm %d%%)");

// Adjust color and font for center circle text
$wp->scale->SetZFont(FF_ARIAL,FS_NORMAL,9);
$wp->scale->SetZFontColor('darkgreen');

// Adjust the font and font color for compass directions
$wp->SetFont(FF_ARIAL,FS_NORMAL,10);
$wp->SetFontColor('darkgreen');

// Adjust the margin to the compass directions
$wp->SetLabelMargin(50);

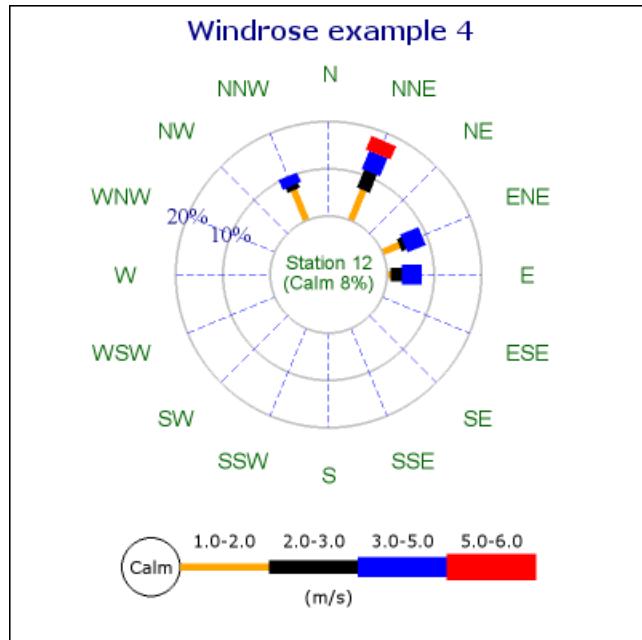
// Adjust grid colors
$wp->SetGridColor('silver','blue');

// Add (m/s) text to legend
$wp->legend->SetText('(m/s)');
$wp->legend->SetMargin(20,5);

// Add and send back to client
$graph->Add($wp);
$graph->Stroke();
?>

```

**Figure 21.22.** (windrose\_ex4.php)  
[example\_src/  
windrose\_ex4.html]



## 21.7.5. Example 5

Objectives:

1. Showing how to add arbitrary paragraphs of text to a graph
2. Showing how to manually set the grid distance
3. Showing how to add a box/frame around the individual plot
4. Showing how to localize the compass directions
5. Showing how to set customized colors for the ranges

```

$graph->title->SetColor('navy');

// Setup graph background color
$graph->SetColor('darkgreen@0.7');

// Setup all the defined text boxes
$n = count($txt);
for($i=0; $i < $n; ++$i) {
 $txtbox[$i] = new Text($txt[$i]);
 $txtbox[$i]->SetPos($txtlayout[$i][0],$txtlayout[$i][1],'right');
 $txtbox[$i]->SetWordwrap($txtlayout[$i][2]);
 $txtbox[$i]->SetParagraphAlign($txtlayout[$i][3]);
 $txtbox[$i]->SetColor($txtlayout[$i][4]);
 $txtbox[$i]->SetBox($txtlayout[$i][5]);
 if(count($txtlayout[$i]) > 6)
 $txtbox[$i]->SetFont($txtlayout[$i][6],$txtlayout[$i][7],$txtlayout[$i][8]);
}
$graph->Add($txtbox);

// Create the windrose plot.
$wp = new WindrosePlot($data);

// Set background color for plot area
$wp->SetColor('lightyellow');

// Add a box around the plot
$wp->SetBox();

// Setup the colors for the ranges
$wp->SetRangeColors($rangeColors);

// Adjust the font and font color for scale labels
$wp->scale->SetFont(FF_ARIAL,FS_NORMAL,9);
$wp->scale->SetFontColor('navy');

// Set the diameter and position for plot
$wp->SetSize(190);
$wp->SetPos(0.35,0.53);

$wp->SetZCircleSize(0.2);

// Adjust the font and font color for compass directions
$wp->SetFont(FF_ARIAL,FS_NORMAL,10);
$wp->SetFontColor('darkgreen');

// Adjust the margin to the compass directions
$wp->SetLabelMargin(50);

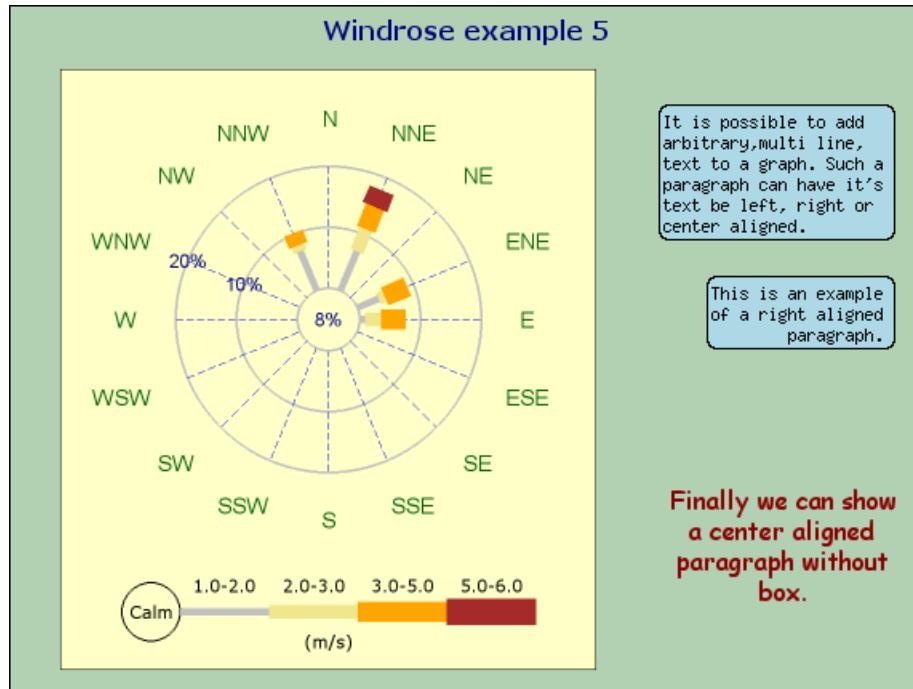
// Adjust grid colors
$wp->SetGridColor('silver','blue');

// Add (m/s) text to legend
$wp->legend->SetText('(m/s)');
$wp->legend->SetMargin(20,5);

// Add plot and send back to client
$graph->Add($wp);
$graph->Stroke();
?>

```

**Figure 21.23.** (windrose\_ex5.php) [example\_src/windrose\_ex5.html]



## 21.7.6. Example 6

Objectives:

1. Set range colors
2. Set grid colors
3. Showing how to adjust the label margin
4. Showing how to change the format for displayed legend values

```

);
// Specify text for direction labels
$labels = array('120.5' => "Plant\n#1275",
 '285.7' => "Reference\n#13 Ver:2");

// Range colors to be used
$rangeColors = array('khaki','yellow','orange','orange:0.7','brown','darkred','bla

// First create a new windrose graph with a title
$graph = new WindroseGraph(400,450);

// Setup titles
$graph->title->Set('Windrose example 6');
$graph->title->SetFont(FF_VERDANA,FS_BOLD,12);
$graph->title->SetColor('navy');

$graph->subtitle->Set('(Free type plot)');
$graph->subtitle->SetFont(FF_VERDANA,FS_ITALIC,10);
$graph->subtitle->SetColor('navy');

// Create the windrose plot.
$wp = new WindrosePlot($data);

// Setup a free plot
$wp->SetType(WINDROSE_TYPEFREE);

// Setup labels
$wp->SetLabels($labels);
$wp->SetLabelPosition(LBLPOSITION_CENTER);
$wp->SetLabelMargin(30);

// Setup the colors for the ranges
$wp->SetRangeColors($rangeColors);

// Adjust the font and font color for scale labels
$wp->scale->SetFont(FF_ARIAL,FS_NORMAL,9);

// Set the diameter and position for plot
$wp->SetSize(230);
$wp->SetZCircleSize(30);

// Adjust the font and font color for compass directions
$wp->SetFont(FF_ARIAL,FS_NORMAL,10);
$wp->SetFontColor('darkgreen');

// Adjust grid colors
$wp->SetGridColor('darkgreen@0.7','blue');

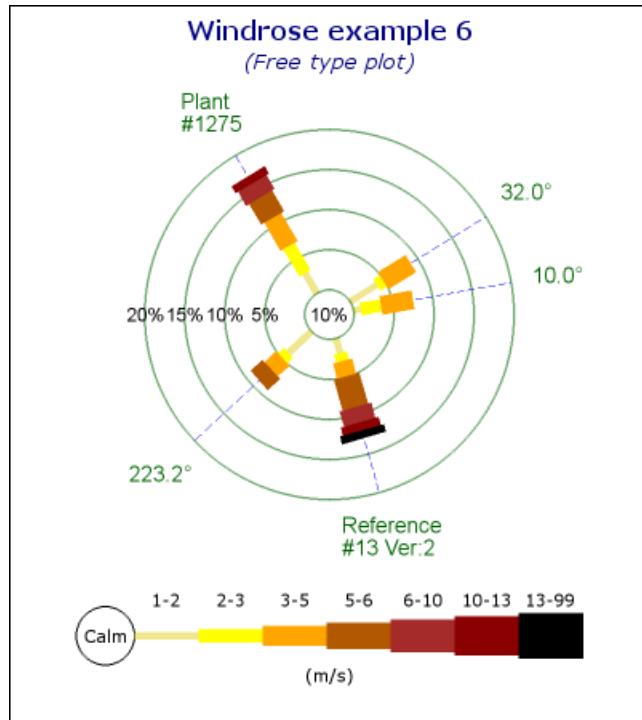
// Add (m/s) text to legend
$wp->legend->SetText('(m/s)');

// Display legend values with no decimals
$wp->legend->SetFormat('%d');

// Add plot to graph and send back to the client
$graph->Add($wp);
$graph->Stroke();
?>

```

**Figure 21.24.** (windrose\_ex6.php)  
[example\_src/  
windrose\_ex6.html]



## 21.7.7. Example 7

Objectives:

1. Showing how to change the direction labels
2. Adjust the displayed ranges
3. Showing how to change the format for displayed legend values
4. Showing how to manually setting the scale range and scale step size

**Example 21.13. (windrose\_ex7.php)**

```

<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

$data = array(
 2 => array(1,15,7.5,2),
 5 => array(1,1,1.5,2),
 7 => array(1,2,10,3,2),
 9 => array(2,3,1,3,1,2),
);

// First create a new windrose graph with a title
$graph = new WindroseGraph(400,450);
$graph->title->Set('Windrose example 7');
$graph->title->SetFont(FF_VERDANA,FS_BOLD,14);
$graph->title->SetColor('navy');

// Create the free windrose plot.
$wp = new WindrosePlot($data);
$wp->SetType(WINDROSE_TYPE16);

// Add some "arbitrary" text to the center
$wp->scale->SetZeroLabel("SOx\n8%");

// Localize the compass direction labels into Swedish
// Note: The labels for data must now also match the exact
// string for the compass directions.
$se_CompassLbl = array('O','ONO','NO','NNO','N','NNV','NV','VNV',
 'V','VSV','SV','SSV','S','SSO','SO','OSO');
$wp->SetCompassLabels($se_CompassLbl);

// Localize the "Calm" text into Swedish and make the circle
// slightly bigger than default
$se_calmtext = 'Lugnt';
$wp->legend->SetCircleText($se_calmtext);
$wp->legend->SetCircleRadius(20);

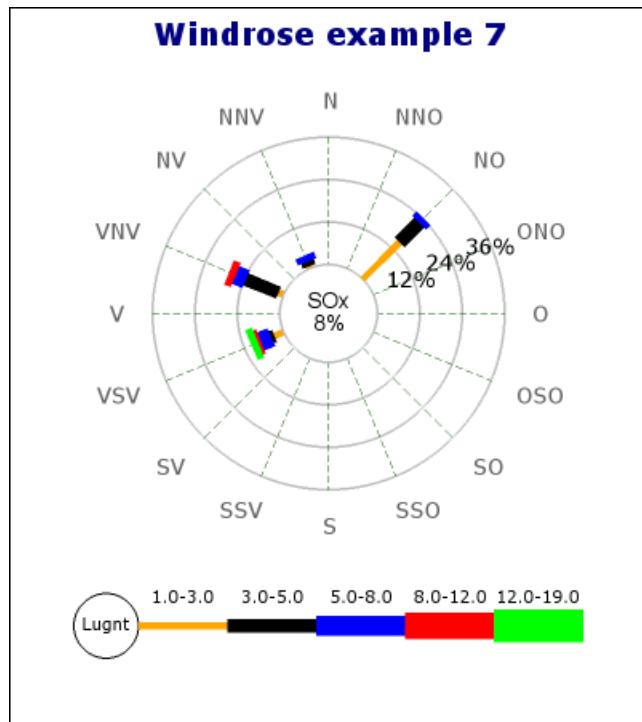
// Adjust the displayed ranges
$ranges = array(1,3,5,8,12,19,29);
$wp->SetRanges($ranges);
// $wp->SetAntiAlias(true);

// Set the scale to always have max value of 30 with a step
// size of 12.
$wp->scale->Set(30,12);

// Finally add it to the graph and send back to client
$graph->Add($wp);
$graph->Stroke();
?>

```

**Figure 21.25.** (windrose\_ex7.php)  
[example\_src/  
windrose\_ex7.html]



## 21.7.8. Example 8

Objectives:

1. Different styles of radial grid lines for a type 8 plot
2. Adding partial labels

**Example 21.14. Show how to set different styles for individual radial grid lines  
(windrose\_ex8.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

// Data can be specified using both ordinal index of the axis
// as well as the direction label.
$data = array(
 '45.9' => array(3,2,1,2,2),
 355 => array(1,1,1.5,2),
 180 => array(1,1,1.5,2),
 150 => array(1,2,1,3),
 'S' => array(2,3,5,1),
);

// Add some labels for afew of the directions
$labels = array(355=>"At\nHome base",180=>"Probe\n123",150=>"Power\nplant");

// Define the color,weight and style of some individual radial grid lines.
$axiscolors = array(355=>"red");
$axisweights = array(355=>8);
$axisstyles = array(355=>'solid',150=>'solid');

// First create a new windrose graph with a title
$graph = new WindroseGraph(400,500);
$graph->title->Set('Windrose example 8');
$graph->title->SetFont(FF_VERDANA,FS_BOLD,14);
$graph->title->SetColor('navy');

// Create the free windrose plot.
$wp = new WindrosePlot($data);
$wp->SetType(WINDROSE_TYPEFREE);

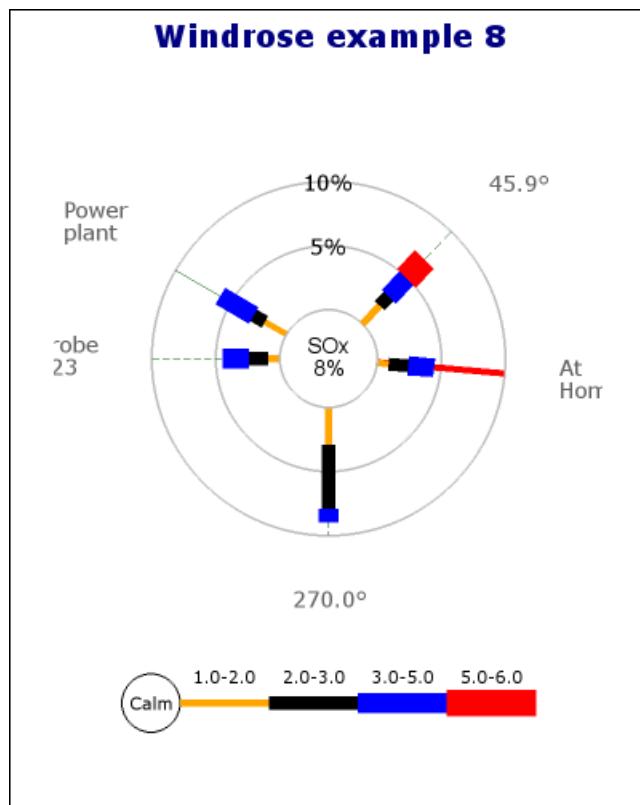
// Specify colors weights and style for the radial gridlines
$wp->SetRadialColors($axiscolors);
$wp->SetRadialWeights($axisweights);
$wp->SetRadialStyles($axisstyles);

// Add a few labels
$wp->SetLabels($labels);

// Add some "arbitrary" text to the center
$wp->scale->SetZeroLabel("SOx\n8%");

// Finally add it to the graph and send back to client
$graph->Add($wp);
$graph->Stroke();
?>
```

**Figure 21.26.** Show how to set different styles for individual radial grid lines (windrose\_ex8.php) [example\_src/windrose\_ex8.html]



## 21.7.9. Example 9

Objectives:

1. Set radial grids for a type 16 plot

**Example 21.15. (windrose\_ex9.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');

// Data can be specified using both ordinal index of the axis
// as well as the direction label.
$data = array(
 'E' => array(3,2,1,2,2),
 'N' => array(1,1,1.5,2),
 'nw' => array(1,1,1.5,2),
 'S' => array(2,3,5,1),
);

// Define the color,weight and style of some individual radial
// grid lines. Axis can be specified either by their (localized)
// label or by their index.
// Note: Depending on how many axis you have in the plot the
// index will vary between 0..n where n is the number of
// compass directions.
$axiscolors = array('nw'=>'brown');
$axisweights = array('nw'=>8); // Could also be specified as 6 => 8
$axisstyles = array('nw'=>'solid');

// First create a new windrose graph with a title
$graph = new WindroseGraph(400,500);
$graph->title->Set('Windrose example 9');
$graph->title->SetFont(FF_VERDANA,FS_BOLD,14);
$graph->title->SetColor('navy');

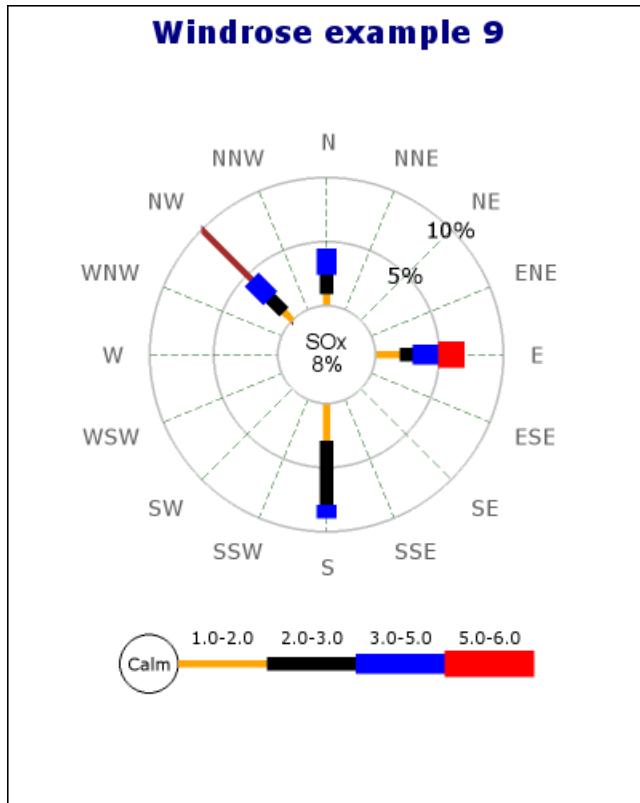
// Create the free windrose plot.
$wp = new WindrosePlot($data);
$wp->SetType(WINDROSE_TYPE16);

// Specify colors weights and style for the radial gridlines
$wp->SetRadialColors($axiscolors);
$wp->SetRadialWeights($axisweights);
$wp->SetRadialStyles($axisstyles);

// Add some "arbitrary" text to the center
$wp->scale->SetZeroLabel("SOx\n8%%");

// Finally add it to the graph and send back to the client
$graph->Add($wp);
$graph->Stroke();
?>
```

**Figure 21.27.** (windrose\_ex9.php)  
[example\_src/windrose\_ex9.html]



## 21.7.10. Example 10

Objectives:

1. Showing how to add a background image which is one of the country flags

**Example 21.16. (windrose\_bgimg\_ex1.php)**

```

<?php // content="text/plain; charset=utf-8"
require_once ('jpgraph/jpgraph.php');
require_once ('jpgraph/jpgraph_windrose.php');
require_once ('jpgraph/jpgraph_flags.php');

// Data can be specified using both ordinal index of axis as well
// as the direction label
$data2 = array(
 'vsv' => array(12,8,2,3),
 6 => array(5,4,4,5,4),
);

$se_CompassLbl = array('O', 'ONO', 'NO', 'NNO', 'N', 'NNV', 'NV', 'VNV', 'V', 'VSV', 'SV', 'S');

// Create a new small windrose graph
$graph = new WindroseGraph(400,400);
$graph->SetMargin(25,25,25,25);
$graph->SetFrame();

$graph->title->Set('Example with background flag');
$graph->title->SetFont(FF_VERA,FS_BOLD,14);

// $graph->SetBackgroundImage('bkgimg.jpg',BGIMG_FILLFRAME);
// $graph->SetBackgroundImageMix(90);
$graph->SetBackgroundCFlag(28,BGIMG_FILLFRAME,15);

$wp2 = new WindrosePlot($data2);
$wp2->SetType(WINDROSE_TYPE16);
$wp2->SetSize(0.55);
$wp2->SetPos(0.5,0.5);
$wp2->SetAntiAlias(false);

$wp2->SetFont(FF_ARIAL,FS_BOLD,10);
$wp2->SetFontColor('black');

$wp2->SetCompassLabels($se_CompassLbl);
$wp2->legend->SetMargin(20,5);

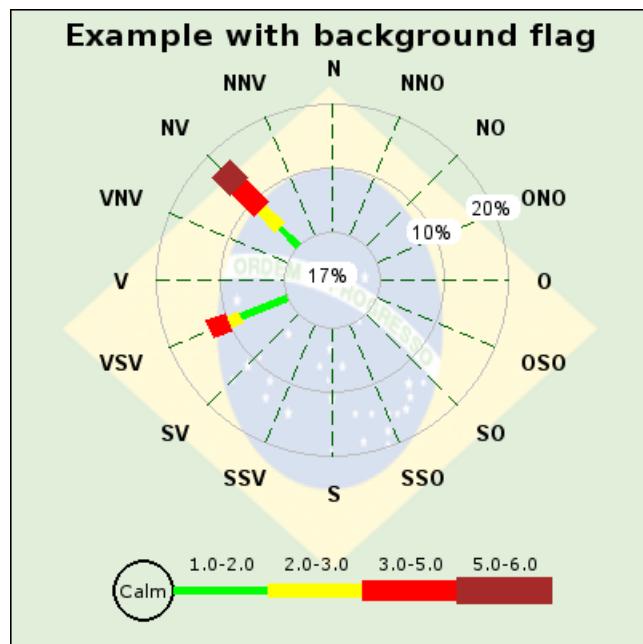
$wp2->scale->SetZFont(FF_ARIAL,FS_NORMAL,8);
$wp2->scale->SetFont(FF_ARIAL,FS_NORMAL,9);
$wp2->scale->SetLabelFillColor('white','white');

$wp2->SetRangeColors(array('green','yellow','red','brown'));

$graph->Add($wp2);
$graph->Stroke();
?>

```

Figure 21.28. (windrose\_bgimg\_ex1.php) [example\_src/windrose\_bgimg\_ex1.html]



---

# Chapter 22. Matrix graphs

## 22.1. Introduction

### Note

This module is only available in the pro-version of the library.

Matrix graphs are used to visualize the content of a rectangular matrix. Each entry in the matrix is mapped to a specific color which are then displayed in a rectangular plot corresponding to the size of the input matrix. The size of each module (corresponding to a matrix entry) is user customizable. There are two types of possible modules; circular and rectangular shape.

In order to achieve high quality rendering when circles are used the matrix module makes use of an optional (user settable) super-sampling to achieve an anti-alias effect to take the edge of the filled circles.

The color assignment to each value is controlled by the selected color map. The library first establish the min and max value in the matrix and then equates the min value with the "lower" end of the specified color map and the max value with the "higher" end of the color map. All values in between is linearly interpolated to have a color between the "low" and "high" end in relation to the value (which are guaranteed to be between the min and max value previously established). There are several pre-defined color maps available as well as completely user configurable color maps.

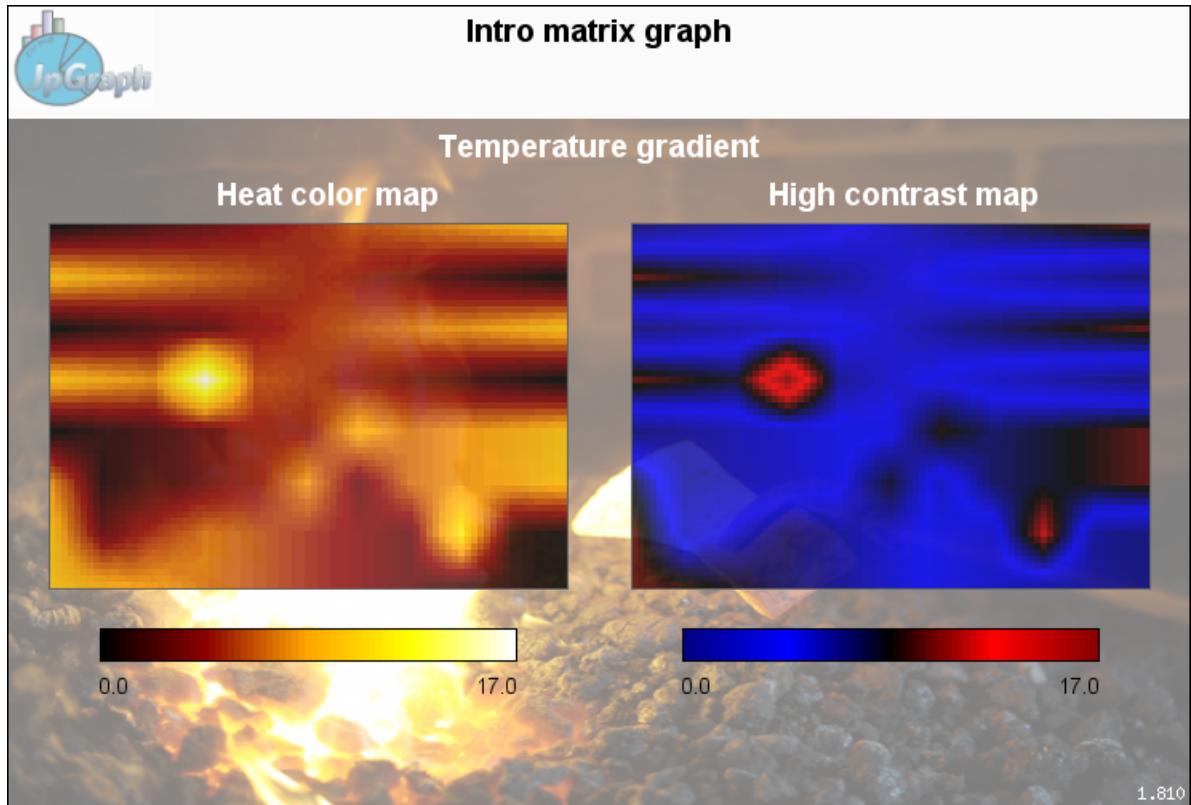
The (optional) legend shows the range associated to each color in the corresponding plot.

### Tip

The library supports mesh interpolation. Mesh interpolation which by itself doesn't produce any new data but can help produce smoother plots by creating "false" interpolated values in between the original entries in the matrix. This is similar to the Matlab (tm) command **interp2**

The overall structure and capabilities of matrix graphs follows that of other graphs in the library, e.g. Windrose plots, in that a graph can have multiple plots, title, footers, texts, icons, image backgrounds etc. Figure 22.1, “A medium complex example to shows some capabilities of matrix plots (matrix\_introex.php)” shows a medium complex example of two matrix plots in a matrix graphs which uses both a background image, icon (the logo in the top left corner), timing of the graph and free positioned text. We have also used a small degree of alpha blending in the plots just to let the background “shine” through a little bit. The two plots shows the same data but uses different color maps.

**Figure 22.1.** A medium complex example to shows some capabilities of matrix plots (`matrix_introex.php`) [[example\\_src/matrix\\_introex.html](#)]



The library offers an extensive range of formatting for the appearance of the plot when it comes to adjusting exactly how the colors are used. There are several built in color maps as well as the possibility to specify manual color maps and adjusting the contrast (scale range) of the plot.

## 22.1.1. Features of the matrix plots

- Both manual and automatic scale range setting
- Customizable legend position and layout
- Row and column legends (edge labels)
- Several built in color maps (22)
- Option for manual specified color maps
- Adjustable color contrast
- Layout classes to position multiple plots in the sam graph
- Both circular and rectangular module type
- Mesh interpolation of input data
- Flexible sizing of matrix plots both manual, automatic and a mix of them

- Alpha blending support for matrix plot
- All the usual graph support for background images, text objects and icons

## 22.2. Creating and formatting a basic matrix graph

In order to create a matrix plot the extension module "jpgraph\_matrix.php" must be included in the script along with the core module "jpgraph.php".

The creation of Matrix graphs otherwise follows the traditional steps in the library of first creating a matrix graph which acts as a canvas for one or several matrix plots. The principle steps are:

1. Create a basic matrix graph as an instance of `class MatrixGraph` (in all our example we use the variable `$graph` to hold this instance)
2. Create an instance of one or several matrix plots as instances of `class MatrixPlot`, make any necessary modifications to the color map and appearance and then add the plots to the matrix graph canvas.
3. Send back the graph to the client with a call to `MatrixGraph::Stroke()`. As usual this can be used to either send back the graph to the client (e.g.browser) or write the graph to a file by specifying a filename as the first argument to `MatrixGraph::Stroke()`.

The example in Figure 22.2, “A basic matrix graph with all default values (matrix\_ex0.php)” shows a matrix plot using just the default values for all parameters.

**Example 22.1. A basic matrix graph with all default values (`matrix_ex0.php`)**

```

<?php
require_once('jpgraph/jpgraph.php');
require_once('jpgraph/jpgraph_matrix.php');

// Some (random) matrix
$data = array(
 array(0,1,2,3,4,5,6,7,8,9,10),
 array(10,9,8,7,6,5,4,3,2,1,0),
 array(0,1,2,3,4,5,6,7,8,9,10),
 array(10,9,8,17,6,5,4,3,2,1,0),
 array(0,1,2,3,4,4,9,7,8,9,10),
 array(8,1,2,3,4,8,3,7,8,9,10),
 array(10,3,5,7,6,5,4,3,12,1,0),
 array(10,9,8,7,6,5,4,3,2,1,0),
);

// Setup a basic matrix graph and title
$graph = new MatrixGraph(400,300);
$graph->title->Set('Basic matrix example');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);

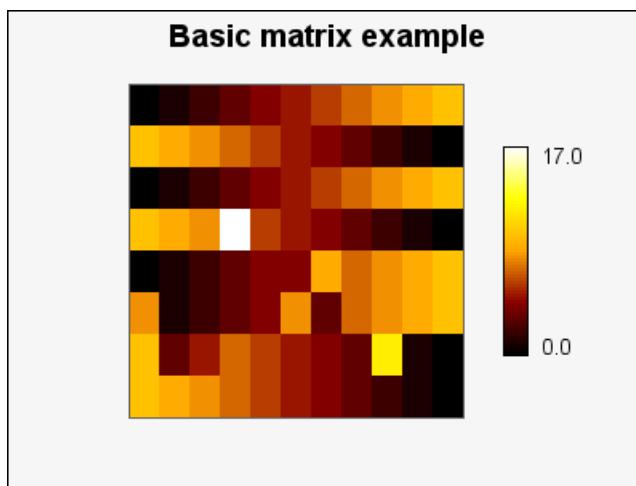
// Create a matrix plot using all default values
$mp = new MatrixPlot($data);
$graph->Add($mp);

$graph->Stroke();

?>

```

**Figure 22.2. A basic matrix graph with all default values (`matrix_ex0.php`)**  
**[[example\\_src/matrix\\_ex0.html](#)]**



In the same way as for other graph types one or several Matrix plots can be added and positioned freely in the Matrix graph by specifying the position as either absolute coordinates or as fractions of the width/height of the overall graph.

## Tip

An easier way to position Matrix plots is to use layout classes as described in Section 22.7, “Using layout classes to position matrix plots”

## Tip

If the data for the matrix is available in a file a convenient way to get hold of the dat in the file is to use the utility class `ReadFileData` to get hold of the data using the method

- `ReadFileData::FromMatrix($aFile, $aSeparator= ' ')`

which read the matrix from a file. Each row of the matrix must be a separate line and each cell is separated with the character specified as the second argument. By default a space is used as separator. All values read back are converted to floating point numbers (double precision). The following short example shows how easy this is to use

```
$data = ReadFileData::FromMatrix('matrixdata.txt');
```

We will not further discuss all the standard graph formatting options such as the ability to add title(s), footers etc. since this is covered in numerous other places in this manual.

## 22.3. Mesh interpolating of input data

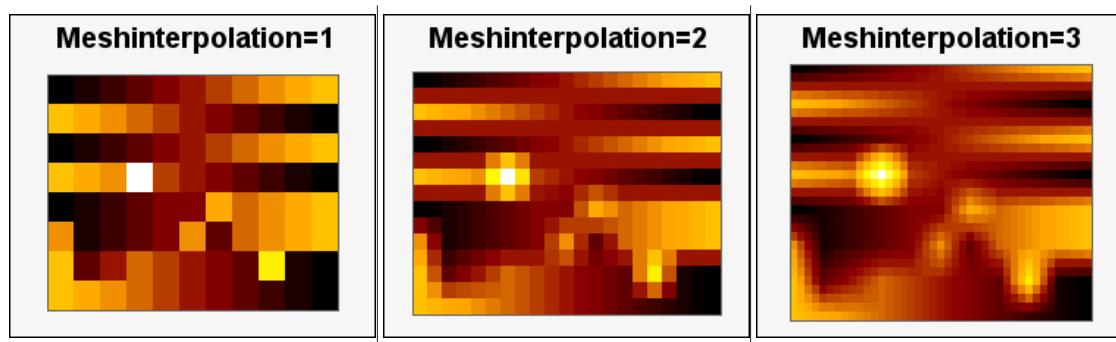
By using mesh interpolation it is possible to obtain a "smoother" looking matrix plot by creating a "in-between" values in the original matrix by linear interpolation.

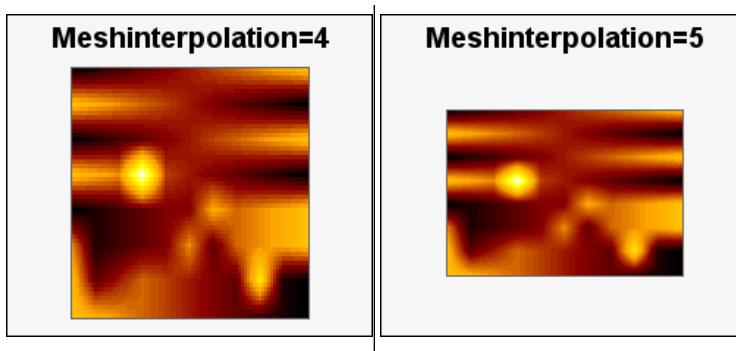
## Tip

This is also used in contour plots. See Section 15.6.4, “Understanding mesh interpolation” for a more thorough discussion on mesh interpolation and the implication of CPU usage.

The interpolation factor specifies how many times, recursively, the interpolation should be done. Practical value ranges from 2-6. While it is possible to specify larger values than 6 the time it takes to do the interpolation will grow exponentially in the interpolation factor. It is also important to remember that this interpolation dos not create any "more" information than what is already available in the matrix. In addition it needs to be verified that such a linear interpolation of data is at all valid for the underlying data in the matrix.

As an example the following figures show the effect of doing a 1-5 times interpolation of the original data (same as interpolation = 1). With the chosen graph size it is no point of interpolating further since doing 5 times interpolating will force the module to be 1x1 pixel in order to fit within the constraints of the graph. (The original data was 8x11 and interpolating it 5 times creates a 113x161 matrix)





The different sizes of the plot is due to the fact that each cell in the matrix must have an integer number of pixels. In the graphs above we have used the largest module size while still fitting in the image. Hence the different appearances.

There are two ways of doing this interpolation.

1. When the matrix plot is created by specifying the interpolation factor as the second argument to the plot constructor, i.e.

```
$matrixplot = new MatrixPlot($data, 4); // 4 times interpolation
```

2. If many plots share the same data it is more efficient to do it once in the beginning instead of doing the interpolation each time a new matrix plot object is created. This can be done by using the utility function

- `doMeshInterpolate(&$aData, $aFactor)`

As can be seen from the declaration this is a call by reference method where the data is replaced by the new data that has been interpolated the specified number of times. This avoids unnecessary data copying for large matrices.

### Note

Those familiar with Matlab (tm) will recognize a similar mesh interpolation in the `interp2()` function.

## 22.4. Formatting the matrix plot

### 22.4.1. Color maps

The color map is a property of the matrix plot and all aspects of the color map is accessed through the "MatrixPlot::colormap" property which is an instance of class `MatrixColormap`.

To adjust how the color map is used the following methods are available

- `ColorMap::SetMap($aMap, $aInvert=false)`

This is a polymorphic method that can take either a symbolic constant as argument and in that case specifies one of the built in color maps or it can be an array of colors which in that case specifies a manual color map. There are several built in color maps of different types which are referenced by an integer in range [0, 19]. A list of the built in color maps can be found in Section 22.8, "Built in color maps"

For example the specification

```
$mp->colormap->SetMap(0);
```

will make the library use the heat color map which is standard color map that goes from black for the min value up through red, orange, yellow and finally white for the highest value. This could be thought of as the color of a heated iron rod (the hottest iron is white glowing).

The following manual specification uses three color plateau that the map touches.

```
$map = array('navy', 'gray', 'red');
$mp->colormap->Set($map);
```

would create a color map that would use 'navy' color for the min value and 'red' for the max value and linearly interpolate all other values for a color range continuously moving from 'navy'-'gray'-'red'. The color interpolation made between these values is a linear interpolation of the corresponding RGB values which creates the illusion of a continuous color change. The array must have at least two colors. There is no limit on the maximum number of base colors specified. However using more than ~7-10 colors to specify the color map is probably not to be recommended.

If \$aInvert is set to true then the color map will be reversed so that the lowest color in the map will be the highest and vice versa.

- `ColorMap::GetCurrMap()`

Return the current set color map

- `ColorMap::SetRange($aMin,$aMax)`

This is used to manually specify the min and max values to be used for mapping matrix values to a color. Any values in the matrix that is lower than \$aMin will be set to the min color and any values above \$aMax will be set to the maximum color.

The matrix values in between will be mapped to the entire color scale. Another way of looking at this is that this will compress the dynamic range used or the colors. If the min and max values specified are much higher and lower than the actual content of the matrix the result is that most of the colors in the matrix will have "middle" colors (and hence lower the contrast). By default the values are (0,0) which means that the library will automatically determine the min/max value based on the input data and use the entire range of colors specified in the color map.

```
<?php
// This will lower the contrast by roughly 50 percent
$mindataaval = ... ; // The minimum data value
$maxdataaval = ... ; // The maximum data value
$contrast = -0.5; // Reduce the contrast by 50%
$adj = ($maxdataaval-$mindataaval+1)*$contrast/2;
$matrixplot->colormap->SetRange($mindataaval+$adj, $maxdataaval-$adj);
?>
```

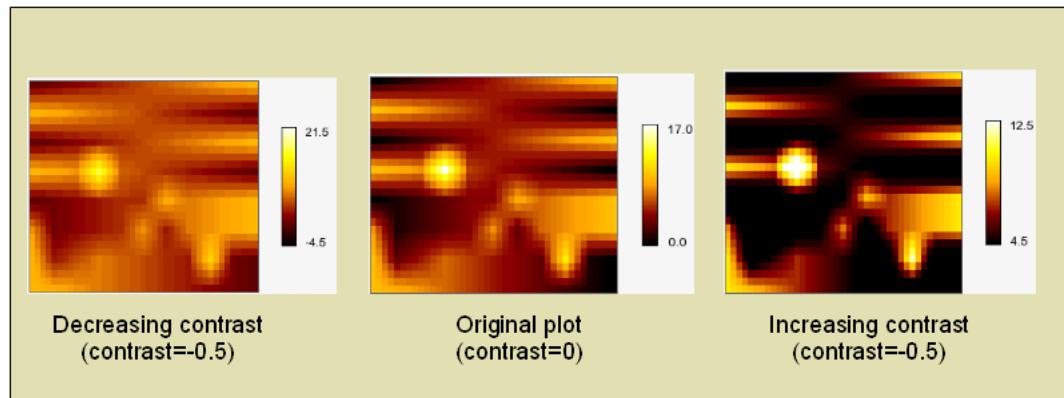
In a similar way the following code will instead increase the "contrast" 50% by letting a large part of the data values become mapped to the min and max color values.

```
<?php
// This will increase the contrast by roughly 50 percent
$mindataaval = ... ; // The minimum data value
$maxdataaval = ... ; // The maximum data value
$contrast = 0.5; // Increase the contrast by 50%
$adj = ($maxdataaval-$mindataaval+1)*$contrast/2;
$matrixplot->colormap->SetRange($mindataaval+$adj, $maxdataaval-$adj);
```

?&gt;

The three plots in Figure 22.3, “The effects of changing the value range for the colormap” shows the effect of increasing and decreasing the contrast with this method.

**Figure 22.3. The effects of changing the value range for the colormap**



## Tip

When using the auto ranging (the default) the contrast can be adjusted with a call to

- `MatrixPlot::SetAutoContrast($aContrast)`

as in

```
$matrixplot->SetAutoContrast(-0.3);
```

- `ColorMap::SetNumColors($aNum)`

This is used to specify the number of discrete color steps used in the map. By default the scale range is divided in approximately 64 color buckets that all matrix entries are mapped into. Depending on the actual color map the specified value might be adjusted up or down up to  $+/- (p-1)$ . Where  $p=number\ of\ base\ colors$ . Valid ranges are 3-128 but is also dependent on the actual color map.

The reasons for these restriction and the adjustments are

- The minimum number of colors are the number of base colors in the current color map
- The number of colors must satisfy the equation

$$n = p + k*(p-1), k = 0, 1, 2, \dots \text{ (eq. 1)}$$

where

- $n$  = number of colors
- $p$  = number of base colors that specifies the map

The specified number of colors will be adjusted to the closest number that satisfies (eq. 1).

- `ColorMap::SetNullColor($aColor)`

If specified this determines what color will be used for any values in the matrix that are null

## 22.4.2. Changing the module type (rectangle vs. circle)

By default the module type (the shape that represents one cell in the matrix) is a rectangle. As was mentioned in the introduction this can also be a circle. This is controlled by the method

- `MatrixPlot::SetModuleType($aType)`

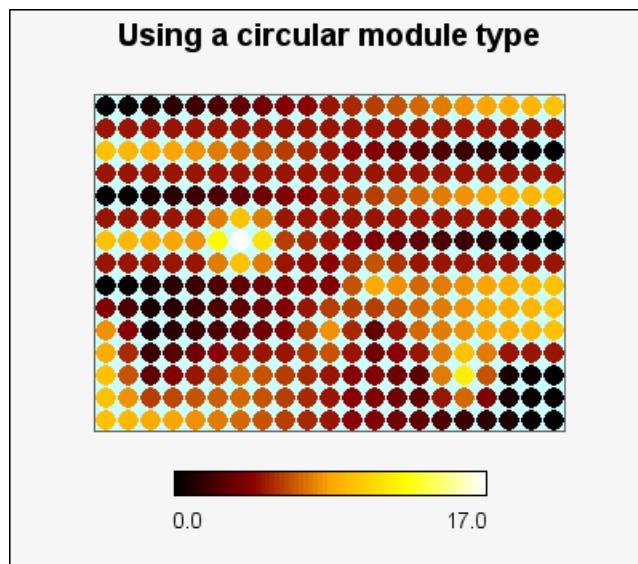
`$aType`, 0 = Use rectangle, 1 = Use a circle

When using circular module type it might also be useful to specify a separate background color for the plot since there will be some space between the circles where the background can be seen. The plot background is specified with the method

- `MatrixPlot::SetBackgroundColor($aColor)`

An example of using circular modules can be seen in Figure 22.4, “Using a circular module type (`matrix_ex05.php`)”

**Figure 22.4. Using a circular module type (`matrix_ex05.php`)**  
[[example\\_src/matrix\\_ex05.html](#)]



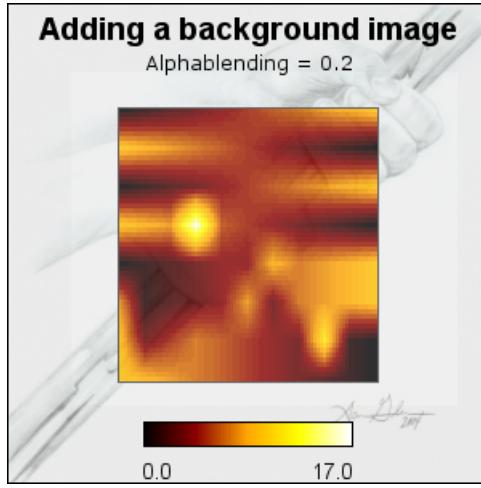
## 22.4.3. Adjusting the alpha blending of the plots

By default the plots are filled with solid colors from the chosen color map. By specifying an alpha value it is possible to let the background shine through the matrix plot. The alpha blending is chosen by the method

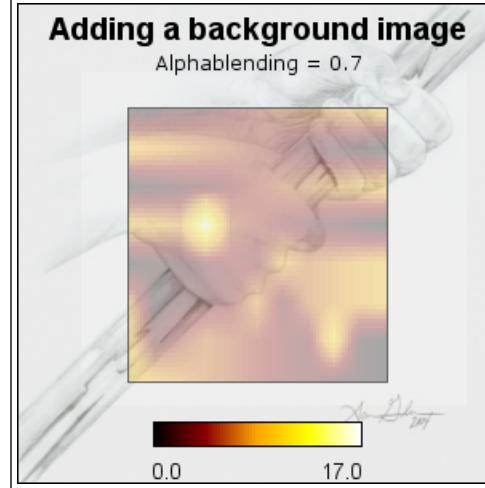
- `MatrixPlot::SetAlpha($aAlpha)`

`$aAlpha` , 0=No transparency, 1=Full transparency (not so useful since then only the background can be seen)

**Figure 22.5. Matrix alpha blending=0.2**  
**(matrix\_ex04.1.php)**  
**[example\_src/**  
**matrix\_ex04.1.html]**



**Figure 22.6. Matrix alpha blending=0.7**  
**(matrix\_ex04.2.php)**  
**[example\_src/**  
**matrix\_ex04.2.html]**



### Note

As can be seen in the figures only the plot area is adjusted. The legend is always shown with no transparency.

## 22.4.4. Specifying the size

There are three and disjunct way to specify the size of the matrix plot. However the size can not be set to any pixel value. Since the matrix plot is a visualization of a matrix the width and height must always be an even multiple of the number of rows and columns since each cell in the matrix have an integer number of pixels as width and height (the module size). This will sometimes force the library to adjust a specified size so that it is an even multiple of the number of rows and columns in the input data matrix.

For example; if the matrix has 50 columns this means that the width will only grow and shrink by multiples of 50 pixels since each cell has an equal number of pixels. The minimum width for such a matrix is 50 pixels.

There are three ways to specify the size:

1. by setting the width and height explicitly (in number of pixels).

Note that the actual rendered size might be different depending on the input matrix size since each cell must have an integer number of pixels and not all sizes will be even dividable with the input data matrix size.

2. by setting the width and height as fractions of the overall width and height of the graph

Note that the actual rendered size might be different depending on the size of the input data matrix.

3. by specifying the width and height of each rendered cell (a.k.a the module size)

This specifies the size (in pixels) of each module. The minimum size is 1x1 pixels.

The size is adjusted by the following two methods

- `MatrixPlot::SetSize($aW,$aH)`

```
MatrixPlot::SetSize($aW)
```

If the two arguments are numbers in range [0.0, 1.0] it will be interpreted as specifying the size as fractions of the overall graph width and height. If the number are > 1 they will be interpreted as the absolute size (in pixels). It is perfectly possible to mix teh two ways. For example the following is a valid size specification

```
$matrixplot->SetSize(250,0.6); // 250px wide and height will be ~60% of the graph
```

If only one argument is specified it will set both the width and height to the specified size. If the single size is specified as a fraction the smallest of the graph width/height will be used as the base.

- `MatrixPlot::SetModuleSize($aW,$aH)`

The two argument specifies the module size, i.e. the size of each cell in the plot (in pixels).

### Note

The size does not effect the legend that belongs to a matrix plot.

## 22.4.5. Specifying the position of the plot on the graph

The position of the plot by is by default centered vertically and slightly move to the left of the vertical center in order to compensate for the default legend that is shown n the right of the plot. The position of the plot is specified by the method

- `MatrixPlot::SetCenterPos($aX,$aY)`

Specifies the center of the plot to be the given x and y-coordinates. The position can be specified in either absolute pixels or as fractions of the width and height (or as a combination).

### Tip

In order to position multiple plots on the same graph it is easier to user the layout classes (as described in Section 22.7, “Using layout classes to position matrix plots”)

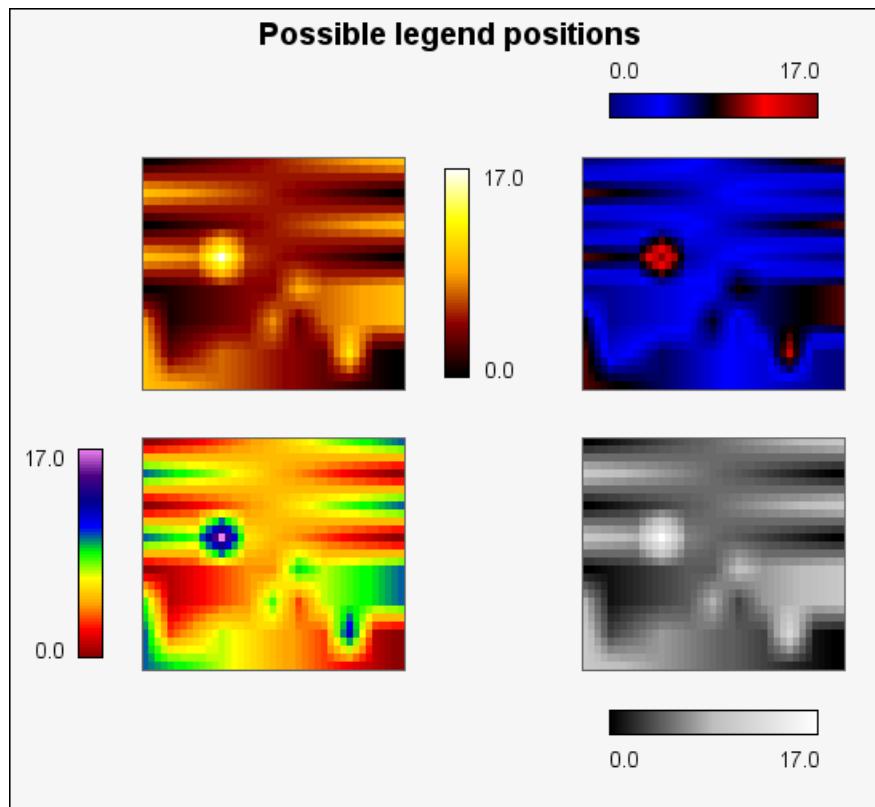
## 22.4.6. Adjusting the legend

The legend belongs to the matrix plot and not the graph. All legend are instances of `class MatrixLegend` and is accessed through the property

- `MatrixPlot::legend`

By default the legend is enabled and positioned to the right of the plot. Both the size and position of the legend can be manually adjusted.

There are four possible positions of the legend as shown in Figure 22.7, “Matrix legend positions”, on each of the four sides of the plot. The labels of the legend will be automatically adjusted to face aways from the plot.

**Figure 22.7. Matrix legend positions**

To position the legend the following `MatrixPlot` method is used

- `MatrixPlot::SetLegendLayout ($aPos)`

The position of the legend is specified as an integer in range [0-3] where 0 is the right side of the plot and the remaining positotins follow clockwise from the right, (i.e. 1 is the bottom, 2 is the left and 3 is the top side).

The following methods can be used to fine tune and adjust the legend

- `MatrixLegend::Show ($aFlg=true)`

Used to enable/disable the legend. The following code line would hide the legend

```
$matrixplot->legend->Show(false);
```

- `MatrixLegend::SetModuleSize ($aBucketWidth, $aBucketHeight=5)`

This specifies the size of each color bucket in the legend.

- `MatrixLegend::SetSize ($aWidth, $aHeight=5)`

This is an alternative way to specify the size of the legend compare with the individual bucket specification. With this method the overall width and height of the legend bar can be adjusted. The size can be specified s either absolute pixels or as fraction of the width height of the entire graph.

- `MatrixLegend::SetMargin ($aMargin)`

Specifies the margin (in pixels) between the matrix plot and the legend.

- `MatrixLegend::SetLabelMargin($aMargin)`

Specifies the margin between the min/max label and the legend bar

- `MatrixLegend::SetFont($aFamily,$aStyle,$aSize)`

Specifies the font for the label on the legend

- `MatrixLegend::SetFormatString($aStr)`

Specifies the format string (in `printf()` format) to be used when rendering the legend label

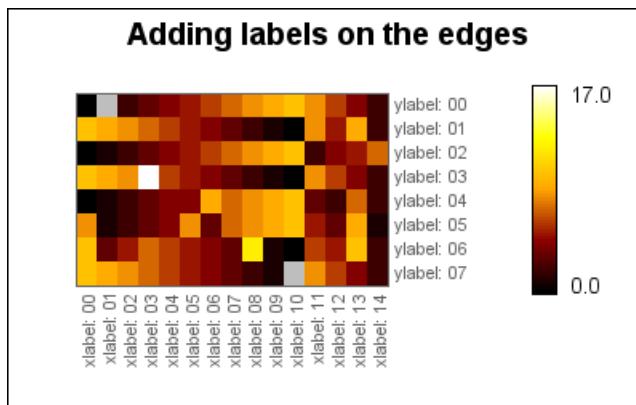
## 22.4.7. Adding row and column legends

### Note

This feature is only available in v3.0.4p and above.

When using matrix plots to display micro arrays it is often desirable to have legends for each row and column. Figure 22.8, “Adding row and column legends to a matrix plot (`matrix_edgeex01.php`)” shows an example of this which helps understand this concept.

**Figure 22.8. Adding row and column legends to a matrix plot (`matrix_edgeex01.php`) [example\_src/matrix\_edgeex01.html ]**



In the library this is modelled by the class `EdgeLabel` which is instantiated in the matrix plot as

- `MatrixPlot::collabel`

Instance for horizontal labels

- `MatrixPlot::rowlabel`

Instance for vertical labels

To adjust the appearance of the labels the following methods can be used:

- `EdgeLabel::SetFont($aFF,$aFS,$aSize)`

Specify the font to be used. Keep in mind that the font size should not be larger than the module size chosen to be able to fit within the row/column it specifies. For the default module size a font size of 8-9 for a TTF font is usually fine. By default the labels are set in FF\_ARIAL, 8 pt size.

- **EdgeLabel::SetFontColor(\$aColor)**

Specifies the font color.

- **EdgeLabel::SetMargin(\$aMargin)**

Set the margin from the edge of the matrix in pixels.

- **EdgeLabel::SetSide(\$aSide)**

Specifies on what side the labels should be drawn. For horizontal (x) labels the possible sides are "left" or "right" and for vertical (y) the possible sides are "top" and "bottom".

- **EdgeLabel::Set(\$aLabels)**

Specifies the 1-dimensional array that holds the labels to be used.

- **EdgeLabel::SetAngle(\$aAngle)**

Specify the angle to draw the label at. By default row labels are drawn at 0 degree and column labels are drawn at 90 degrees angle.

The simplest use of labels is to use the default values for all parameters and just set the labels, i.e.

```
<?php
$data = array(...) ;
$collabels = array(...) ;
$rowlabels = array(...) ;

$mp = new MatrixPlot($data);

$mp->collabel->Set($collabels);
$mp->rowlabel->Set($rowlabels);
?>
```

Below is a final example of adding row and column labels to a matrix graph

```
#=matrix_edgeex02|Adding row and column legends to a matrix plot##
```

## 22.5. Adding icon and text objects to the graph

Matrix plots supports the ordinary way of adding icon and text objects to the graph.

### 22.5.1. Adding a text object

Text objects are added by first creating an instance of class `Text` for each text needed and then adding the text to the graph with the usual call to `MatrixGraph::Add()`.

A basic text will only require two additional lines of code

```
<?php
```

```
$txt = new Text('Simple string',20,20);
$graph->Add($txt);
?>
```

The following code snippet is slightly more complicated and will create a boxed text in the upper right corner of the graph.

```
<?php
// Add a boxed text
$txt = new Text();
$txt->SetFont(FF_ARIAL,FS_NORMAL,10);
$txt->Set("Arbitrary text\nnon a\nMatrix Plot");
$txt->SetParagraphAlign('center');
$txt->SetPos(0.95,0.15,'right');
$txt->SetBox('lightyellow');
$txt->SetShadow();
$graph->Add($txt);
?>
```

The snippet above adds a text at coordinates X=20, Y=20 using the default lower left corner as the text anchor point.

## Note

To add a newline you must remember to use double-quotes to enclose the text otherwise the "\n" will only be interpreted literally.

## Note

Remember that the "text align", as adjusted with `SetAlign()`, specifies the anchor point for the text, i.e. what part of the text is aligned with the specified position.

To add many text strings it is often useful to specify them in an array and then have a loop creating the text object and add the text array of all the created objects to the graph as the following short snippet shows.

```
<?php
//-----
// Add texts to the graph
//-----
$txts = array(
 array('Textstring one ...',$tx1,$ty1),
 array('Textstring two ...',$tx2,$ty2),
 array('Textstring three ...',$tx3,$ty3),

$n=count($txts);
$t=array();
for($i=0; $i < $n; ++$i){
 $t[$i] = new Text($txts[$i][0],$txts[$i][1],$txts[$i][2]);
 $t[$i]->SetFont(FF_ARIAL,FS_NORMAL,12);
 $t[$i]->SetColor('brown');
 $t[$i]->SetAlign('center','top');
}
$graph->Add($t);
?>
```

## 22.5.2. Adding icons to the graph

Icons are added as instances of class `IconPlot` to the graph (as usual with a call to `MatrixGraph::Add()`). This means that to use icons the library module "jpgraph\_iconplot.php" must first be included.

The following example shows how to add a small icon in the lower right corner of the graph. For more information on formatting icons that are added to a graph see Section 14.14, "Adding icons (and small images) to the graph"

### Caution

Since Matrix graphs doesn't have a ny concept of linear scale the position of icons can only be specified as absolute pixels or as fractions of the width/height.

#### Example 22.2. Adding an icon to the lower right corner (`matrix_ex03.php`)

```
<?php
require_once('..../jpgraph.php');
require_once('..../jpgraph_matrix.php');
require_once('..../jpgraph_iconplot.php');

$data = array(
 array(0,1,2,3,4,5,6,7,8,9,10),
 array(10,9,8,7,6,5,4,3,2,1,0),
 array(0,1,2,3,4,5,6,7,8,9,10),
 array(10,9,8,17,6,5,4,3,2,1,0),
 array(0,1,2,3,4,4,9,7,8,9,10),
 array(8,1,2,3,4,8,3,7,8,9,10),
 array(10,3,5,7,6,5,4,3,12,1,0),
 array(10,9,8,7,6,5,4,3,2,1,0),
);

// Do the meshinterpolation once for the data
doMeshInterpolate($data,3);
$r=count($data);$c=count($data[0]);

$width=400; $height=400;
$graph = new MatrixGraph($width,$height);
$graph->title->Set('Adding an icon to the background');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);

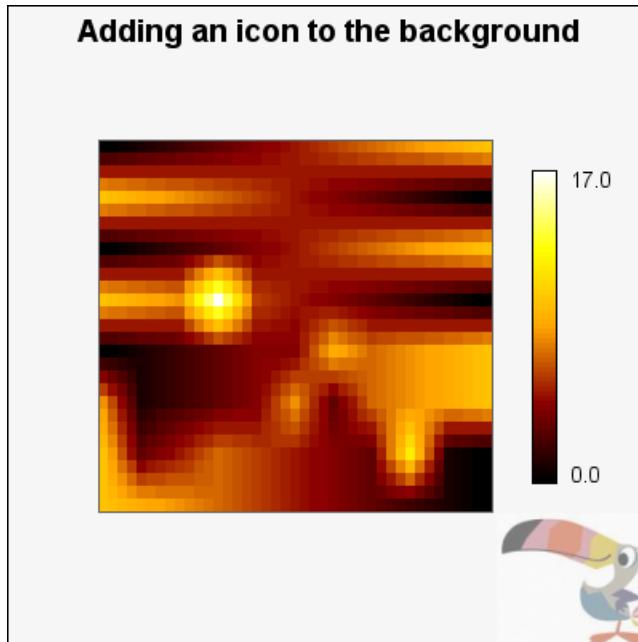
$mp = new MatrixPlot($data,1);
$mp->SetSize(0.6);

$icon = new IconPlot('icon.jpg',$width-1,$height-1,0.8,50);
$icon->SetAnchor('right','bottom');
$graph->Add($icon);

$graph->Add($mp);
$graph->Stroke();

?>
```

**Figure 22.9. Adding an icon to the lower right corner (`matrix_ex03.php`)  
[[example\\_src/matrix\\_ex03.html](#)]**



We also recall that it is possible to use country flags as icons by making use of the method

- `IIconPlot::SetCountryFlag($aFlag,$ax=0,$ay=0,$aScale=1.0,$aMix=100,$aStdSize=3)`

### 22.5.3. Adding background images

In just the same way as for all other plots it is possible to add a background images, background gradients and a background flag to the graph. Since this has been discussed several time previously in the manual we only show a simple example here.

#### Note

Since Matrix graphs does not have the concept of plot area the background image positioning alternative `BGIMG_FILLPLOT` is the same as `BGIMG_FILLFRAME` as we have used in this example below.

**Example 22.3. Adding a background image to the matrix graph  
(matrix\_ex04.php)**

```
<?php
require_once('.../jpgraph.php');
require_once('.../jpgraph_matrix.php');
require_once('.../jpgraph_iconplot.php');

$data = array(
 array(0,1,2,3,4,5,6,7,8,9,10),
 array(10,9,8,7,6,5,4,3,2,1,0),
 array(0,1,2,3,4,5,6,7,8,9,10),
 array(10,9,8,17,6,5,4,3,2,1,0),
 array(0,1,2,3,4,4,9,7,8,9,10),
 array(8,1,2,3,4,8,3,7,8,9,10),
 array(10,3,5,7,6,5,4,3,12,1,0),
 array(10,9,8,7,6,5,4,3,2,1,0),
);

// Do the meshinterpolation once for the data
doMeshInterpolate($data,4);
$r=count($data);$c=count($data[0]);

$width=400; $height=400;
$graph = new MatrixGraph($width,$height);
$graph->title->Set('Adding a background image');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);

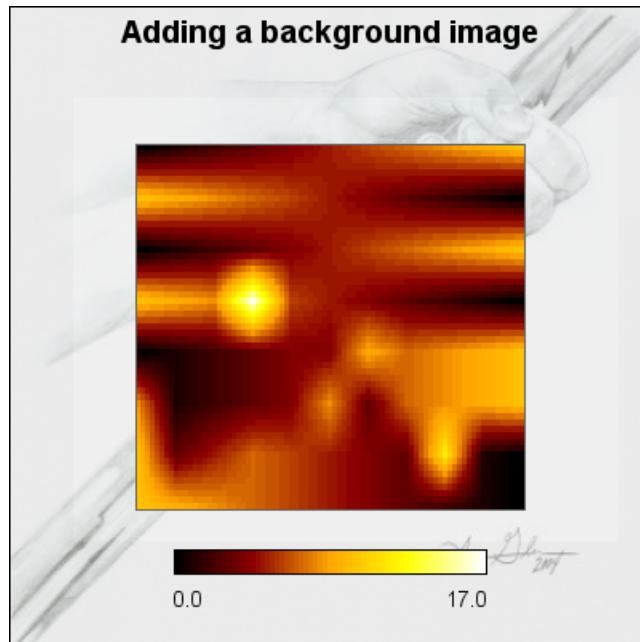
// Add a stretched background image
$graph->SetBackgroundImage('ironrod.jpg',BGIMG_FILLFRAME);
$graph->SetBackgroundImageMix(50);

$mp = new MatrixPlot($data,1);
$mp->SetSize(0.6);
$mp->SetCenterPos(0.5,0.5);
$mp->SetLegendLayout(1);

$graph->Add($mp);
$graph->Stroke();

?>
```

**Figure 22.10. Adding a background image to the matrix graph (matrix\_ex04.php) [example\_src/matrix\_ex04.html]**



## 22.6. Adding marker lines to the matrix plot

### Note

This feature was added in 3.1.1p

In order to mark important divisions in the plot it is possible to add an instance of class `PlotLine` (either horizontal or vertical line) to each plot in the graph. For example, to add one vertical and one horizontal plot line the following lines can be added to the matrix script

```
<?php
$mp = new MatrixPlot($data);

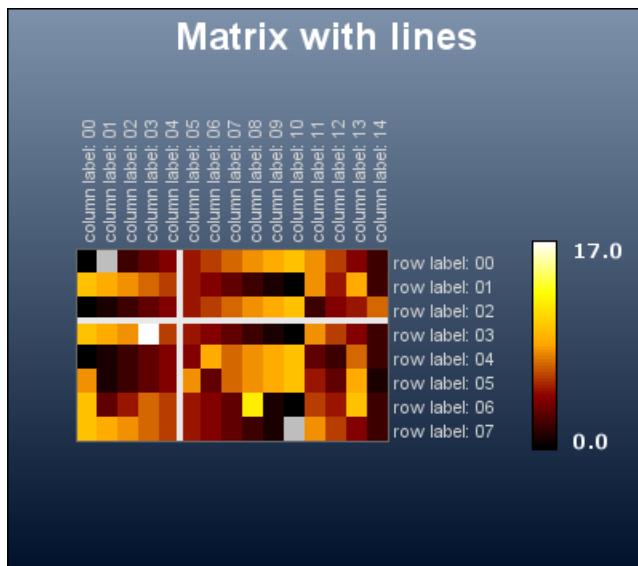
// Create two lines to add as markers
$l1 = new PlotLine(VERTICAL, 5, 'lightgray:1.5', 4);
$l2 = new PlotLine(HORIZONTAL, 3, 'lightgray:1.5', 4);

// Add lines to the plot
$mp->AddLine($l1);
$mp->AddLine($l2);
// this could also be done as
// $mp->AddLine(array($l1,$l2));

...
?>
```

The example in Figure 22.11, “Adding plot lines to the matrix plot (matrix\_ex06.php)” shows an example of this

**Figure 22.11. Adding plot lines to the matrix plot (`matrix_ex06.php`) [[example\\_src/matrix\\_ex06.html](#)]**



## 22.7. Using layout classes to position matrix plots

Normally we don't want to have to calculate the absolute x and y coordinates when positioning multiple Matrix plots in graph. A much better concept would be to just tell the library to position three matrix plots horizontally or vertically without having to figure out the exact coordinates our self but just evenly distribute them in the specified direction.

This is where layout classes come in handy.

There are two types of layouts; horizontal and vertical. To specify that two matrix plots should be positioned side by side (horizontal) a new horizontal layout object is created and then the two matrix plots are added as object within the horizontal layout class. Later on when the objects are about to be stroked on the graph the horizontal layout class will take all its objects and spread them out evenly along a horizontal line depending on the individual size of each matrix plot (including the specified margin). The same principle applies to the vertical layout class with the obvious change in direction.

The layout object are added to the graph in exactly the same way as ordinary matrix plots, by calling the `MatrixGraph::Add()` method.

The following line would create a horizontal line of three matrix plots

```
<?php
$graph = new MatrixGraph($width,$height);

// additional graph formatting

$mp1 = new MatrixPlot($mat1);
$mp2 = new MatrixPlot($mat1);
$mp3 = new MatrixPlot($mat1);
```

```
$hor = new LayoutHor(array($mp1, $mp2, $mp3));

$graph->Add($hor);
$graph->Stroke();
?>
```

A complete example of this can be seen in Figure 22.12, “Using layout classes with Matrix plots (matrix\_layout\_ex1.php)”

**Example 22.4. Using layout classes with Matrix plots  
(matrix\_layout\_ex1.php)**

```
<?php
require_once('..../jpgraph.php');
require_once('..../jpgraph_matrix.php');

$data = array(
array(0,1,2,3,4,5,6,7,8,9,10),
array(10,9,8,7,6,5,4,3,2,1,0),
array(0,1,2,3,4,5,6,7,8,9,10),
array(10,9,8,17,6,5,4,3,2,1,0),
array(0,1,2,3,4,4,9,7,8,9,10),
array(8,1,2,3,4,8,3,7,8,9,10),
array(10,3,5,7,6,5,4,3,12,1,0),
array(10,9,8,7,6,5,4,3,2,1,0),
);

doMeshInterpolate($data,4);

$graph = new MatrixGraph(850,580);
$graph->title->Set('Matrix layout example');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,14);

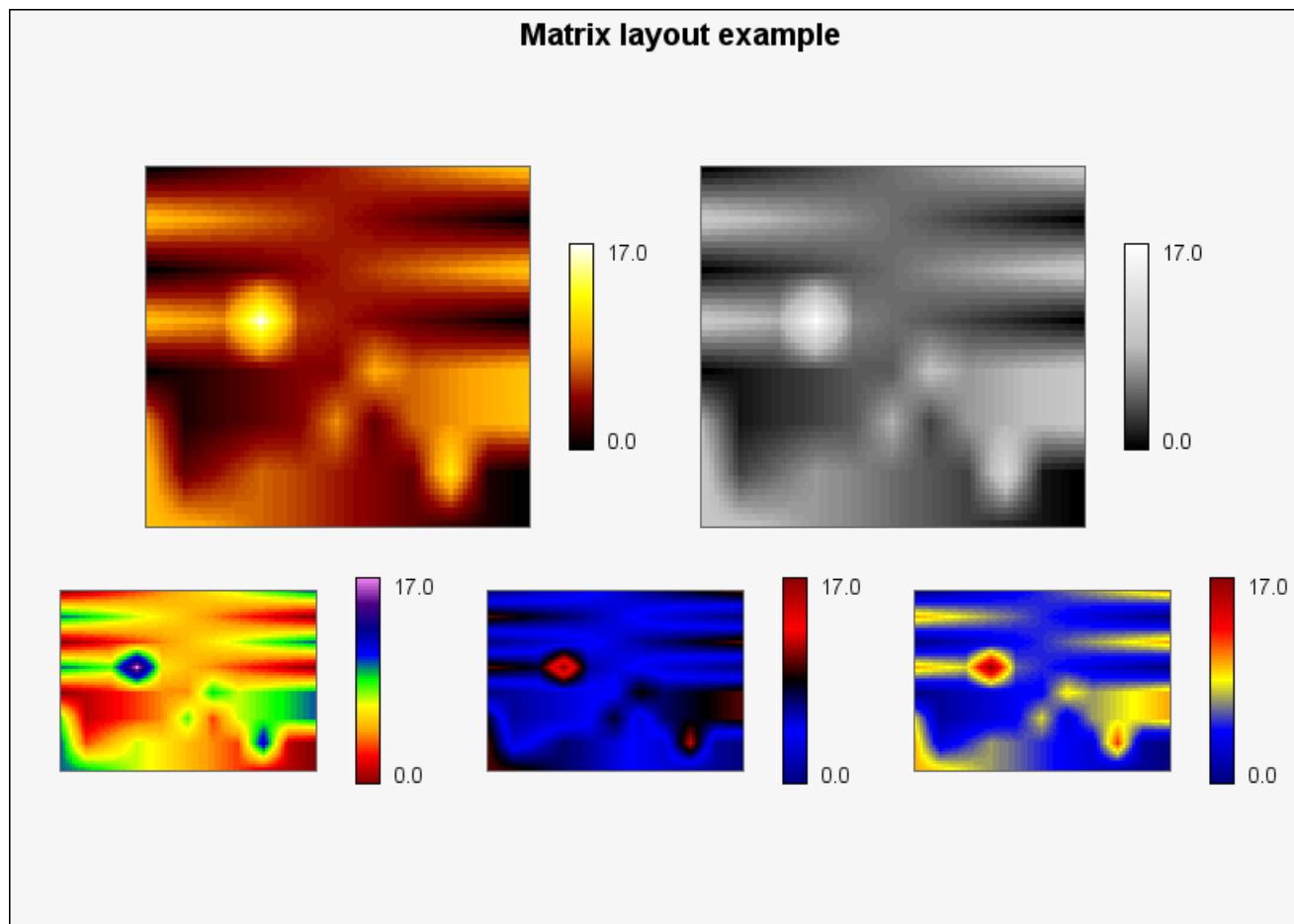
$tmp = array();
$n = 5;
for($i=0; $i < $n; ++$i){
 $tmp[$i] = new MatrixPlot($data);
 $tmp[$i]->colormap->SetMap($i);
 if($i < 2)
 $tmp[$i]->SetSize(0.35);
 else
 $tmp[$i]->SetSize(0.21);
 // We need to make the legend a bit smaller since by
 // defalt has a ~45% height
 $tmp[$i]->legend->SetModuleSize(15,2);
}

$hor1 = new LayoutHor(array($tmp[0],$tmp[1]));
$hor2 = new LayoutHor(array($tmp[2],$tmp[3],$tmp[4]));
$vert = new LayoutVert(array($hor1,$hor2));
$vert->SetCenterPos(0.45,0.5);

$graph->Add($vert);
$graph->Stroke();

?>
```

**Figure 22.12. Using layout classes with Matrix plots  
[matrix\_layout\_ex1.php]  
[example\_src/matrix\_layout\_ex1.html]**



When using layout classes there is one parameter of the matrix plot that can be adjusted and that is the margin around each matrix plot. This margin is controlled by a call to

- `MatrixPlot::SetMargin($aMargin)`

By default the margin is 20 pixels on each side. This margin also takes the legend into consideration so the margin is counted from the edge of the legend.

## 22.8. Built in color maps

There are three types of built in color maps

### 1. Standard maps

Includes rainbow spectrum and heat maps and combination of plain red-green-blue maps

### 2. Normalized center

These maps have a white-ish center. This is mostly useful to visualize the spread from a center value

### 3. Continues

The maps have one base color where the hue changes

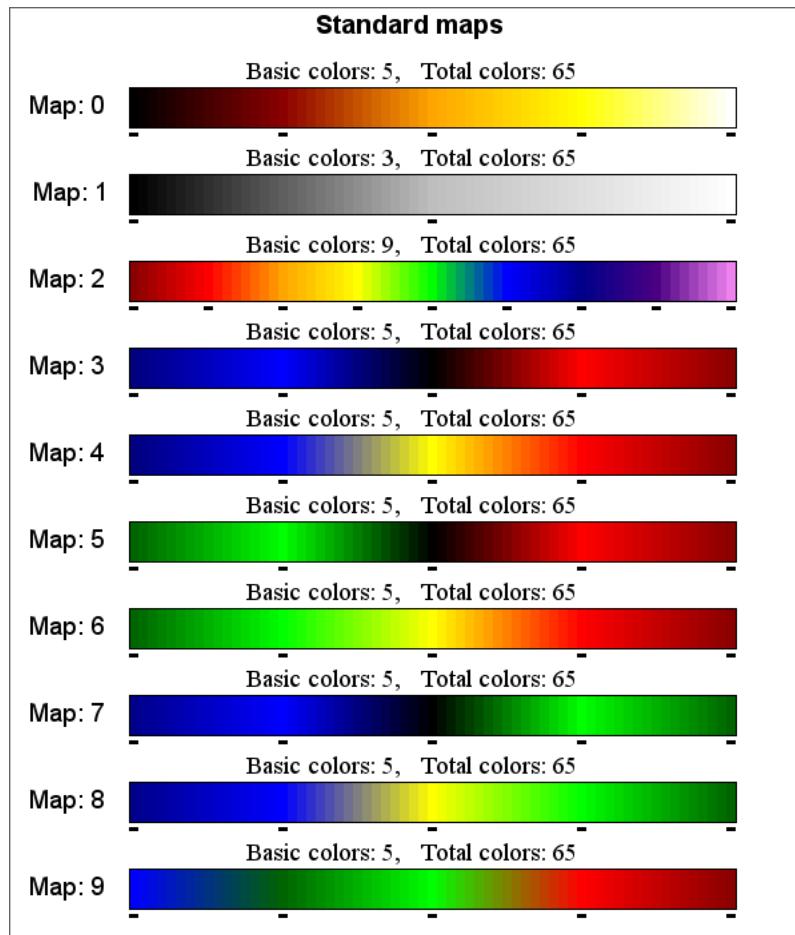
In the color maps in the following section the black bars under some colors in the color map shows the discrete colors (the plateaus) that the color map is made up of. All colors in between two "barred" colors are linear interpolations.

The script that was used to generate these color maps can be found under the example directory as "colormaps.php".

## 22.8.1. Standard maps

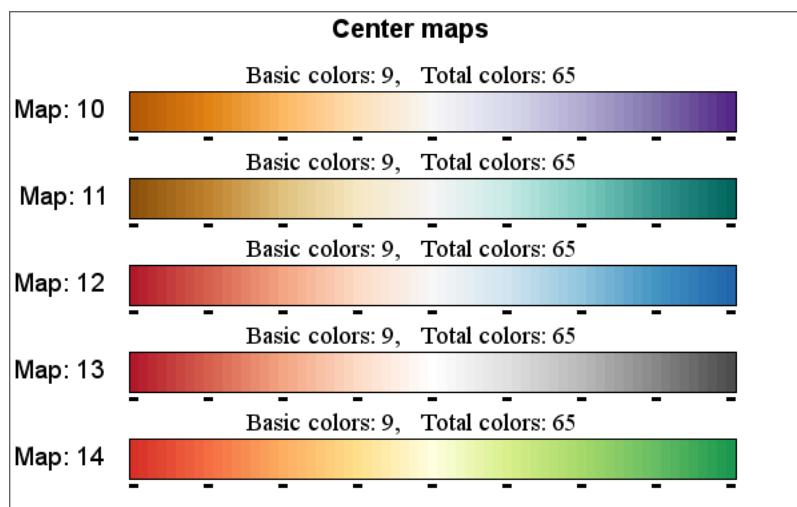
To show heat gradient map 0 is the standard color map. To highlight the min and max values (to get a high contrast) map 3 is a good choice. Map two is a standard "rainbow" spectrum from red all up to violet.

**Figure 22.13. Standard color maps**



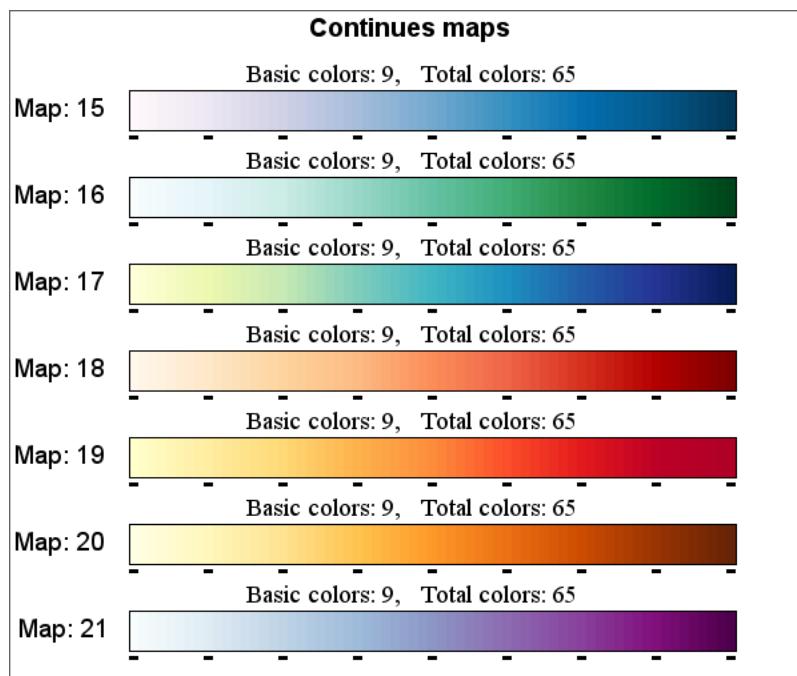
## 22.8.2. Normalized center

These maps have a neutral (white) center and feathers out to the min/max value at each end.

**Figure 22.14. Centered color map**

### 22.8.3. Continues map

These maps changes continues from the min color to the max color.

**Figure 22.15. Continues color map**

## 22.9. Using CSIM with matrix plots

### Note

This feature was added in v3.1.1p

In the same was as most other plot types matrix plots also supports the usage of Client Side Image Maps. (See Chapter 10, *Using CSIM (Client side image maps)* for a full description on the usage of CSIM in general the library.)

The possible hotspot areas in a matrix graph are:

- The title

This is set (as usual with a call to `MatrixGraph::title::SetCSIM()`)

- Each cell in the matrix itself.

This is set with a call to `MatrixPlot::SetCSIM()`. The input is specified with a matrix of the same size as the input data. An error message will be given if the sizes differ.

- Each row and column label text. The input must be an array of the same length as the number of labels.

This is specified with either (or both)

```
MatrixGraph::rowlabel::SetCSIM()
```

```
MatrixGraph::collabel::SetCSIM()
```

The following example shows how to add both label and data CSIM. As usual the graph must be stroked with a call to `MatrixGraph::StrokeCSIM()` when using CSIM functionality.

```

$nx = count($data[0]);
$ny = count($data);

for($i=0; $i < $ny; ++$i) {
 for($j=0; $j < $nx; ++$j) {
 $csimtargets[$i][$j] = '#'.sprintf('%02sd',$i)."-".sprintf('%02sd',$j);
 }
}

for($i=0; $i < $nx; ++$i) {
 $collabels[$i] = sprintf('column label: %02d',$i);
 $collabeltargets[$i] = '#'.sprintf('collabel: %02d',$i);

}
for($i=0; $i < $ny; ++$i) {
 $rowlabels[$i] = sprintf('row label: %02d',$i);
 $rowlabeltargets[$i] = '#'.sprintf('rowlabel: %02d',$i);
}

// Setup a nasic matrix graph
$graph = new MatrixGraph(400,350);

$graph->SetBackgroundGradient('lightsteelblue:0.8','lightsteelblue:0.3');
$graph->title->Set('CSIM with matrix');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,16);
$graph->title->SetColor('white');

// Create one matrix plot
$mp = new MatrixPlot($data,1);
$mp->SetModuleSize(13,15);
$mp->SetCenterPos(0.35,0.6);
$mp->colormap->SetNullColor('gray');

// Setup column labels
$mp->collabel->Set($collabels);
$mp->collabel->SetSide('top');
$mp->collabel->SetFont(FF_ARIAL,FS_NORMAL,8);
$mp->collabel->SetFontColor('lightgray');

// Setup row labels
$mp->rowlabel->Set($rowlabels);
$mp->rowlabel->SetSide('right');
$mp->rowlabel->SetFont(FF_ARIAL,FS_NORMAL,8);
$mp->rowlabel->SetFontColor('lightgray');

$mp->rowlabel->SetCSIMTargets($rowlabeltargets);
$mp->collabel->SetCSIMTargets($collabeltargets);

// Move the legend more to the right
$mp->legend->SetMargin(90);
$mp->legend->SetColor('white');
$mp->legend->SetFont(FF_VERDANA,FS_BOLD,10);

$mp->SetCSIMTargets($csimtargets);

$graph->Add($mp);
$graph->StrokeCSIM();

?>

```

## **22.10. Matrix graph examples**

### **22.10.1. Example 1**

### **22.10.2. Example 2**

### **22.10.3. Example 3**

### **22.10.4. Example 4**

# Chapter 23. Filled contour graphs

## 23.1. Filled Contour graphs

### Note

This section describes additional functionality available in the pro-version v3.1p and above.

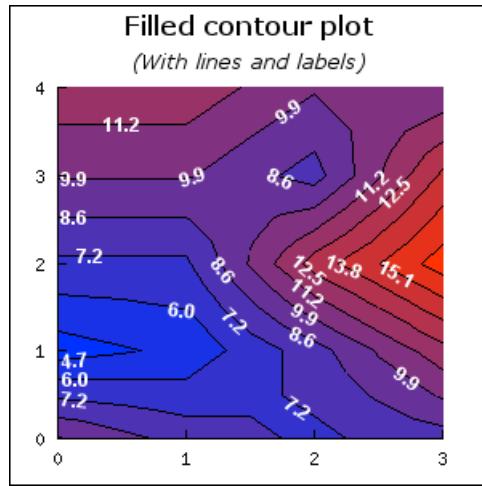
The use of filled ContourPlots requires the inclusion of the module "jpgraph\_contourf.php". The filled version of contour graphs supports the following additional features compared to the regular contour graphs

- Filled contour plots
- Labelling of isobar lines in the contour plots
- Two user selectable adaptive algorithms to determine the contours, one rectangular and one triangular based adaptive mesh algorithm.

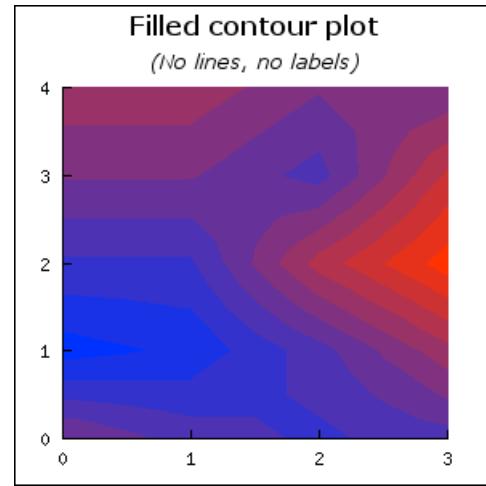
The enhanced contour plot is contained in the new plot class `FilledContourPlot`.

As a quick introduction the two examples below shows two contour graphs which makes use of these additional features

**Figure 23.1. Filled contour with labels (contour2\_ex1.php)**  
[[example\\_src/contour2\\_ex1.html](#)]



**Figure 23.2. Filled contour with no isobar lines (contour2\_ex2.php)**  
[[example\\_src/contour2\\_ex2.html](#)]



All basic formatting, such as specifying colors and number/location of the isobar lines are handled in exactly the same way as described in the previous section for basic contours.

### 23.1.1. Creating a filled contour graph

The filled contour graph follows the exact same principles as for the non filled graph described above. The core difference is that a filled graph is created as an instance of class `FilledContourPlot`, i.e.

```
<?php
$filledContour = new FilledContourPlot($data);
?>
```

## 23.1.2. Enabling and disabling contour lines

Contour lines are controlled with the method

- `FilledContourPlot::ShowLines($aFlg,$aColorWithFilled='black')`

The `$aFlg` is a boolean that determines if contour lines should be shown or not. by default they are enabled. The second argument `$aColorWithFilled` determines what color the contour lines should have when the contour is filled. If the contour is not filled then the contour lines will have the color of the isobar.

## 23.1.3. Specifying a filled contour

In order to have the contour filled the method

- `ContourPlot::SetFilled($aFlg)` Determines if the contour should be filled or not.

should be called.

Since there are  $n$  isobar lines and there needs to be  $n+1$  colors to fill the graph there is a choice of what side should the color for isobar line  $n$  fill. The area above or the area below the isobar. In this library we use the convention that the color for isobar line  $n$  is used to fill the area up to the next highest isobar  $n+1$ .

If the colors are specified manually it is important that one more color than the number of isobar lines are specified. The convention is such that the color specified in the first position (position = 0) in the color array is used to fill the area below the lowest specified isobar.

An example will make this clear.

The graph below only uses three isobar lines (to keep things simple) which means that we need four colors. To create such a graph we use the lines

```
<?php
$data = array(...);

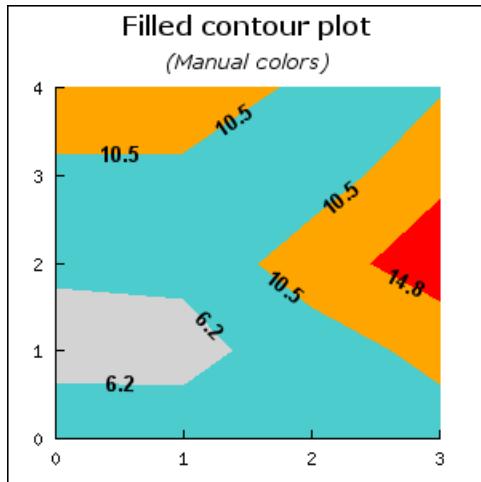
$isobar_colors = array('lightgray','teal:1.3','orange','red');

// Create a new contour graph with three isobar lines
$cp = new ContourPlot($data,3);

// Specify the colors manually
$cp->SetIsobarColors($isobar_colors);

?>
```

**Figure 23.3. Manual colors for contour (contour2\_ex3.php)**  
[example\_src/contour2\_ex3.html]



as can be seen from the labels in the graph the algorithm has resulted in three isobar lines (as specified) at values  $6.2$ ,  $10.5$  and  $14.8$ . The first color "gray" is used to fill the area below the smallest isobar ( $6.2$ ).

It should be noted that by specifying `SetFilled(false)` a non filled contour graph can be created. However, the visual appearance compared with the class `ContourPlot` will be slightly different. The reason is that the filled contour plot class uses an adaptive algorithm that gives better fidelity than the simpler non adaptive algorithm used in the standard (non-filled) contour plot class. The contour corresponds to creating a contour with the plain contour plot algorithm (class `ContourPlot`) using an interpolation factor of ~3-4.

### 23.1.4. Adjusting the labels in the contour

In order to make it easier to follow the contour labels can be added inside the contour. The labels are placed using a heuristic algorithm that tries to place enough labels to make the plot easy to read but not as many labels as to make it crowded.

There are two main characteristic of the labels that can be controlled.

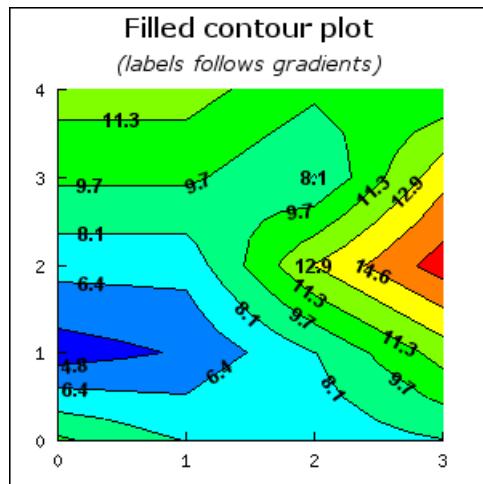
- The font and color of the labels which are controlled by the two methods `FilledContourPlot::SetFont()` and `FilledContourPlot::SetFontColor()`
- Whether or not the label should follow the gradient of the isobar line at the point where the label is displayed or if it should always be drawn horizontally.

To enable or disable the display of the labels as well as determining if the label should follow the gradient or not the method

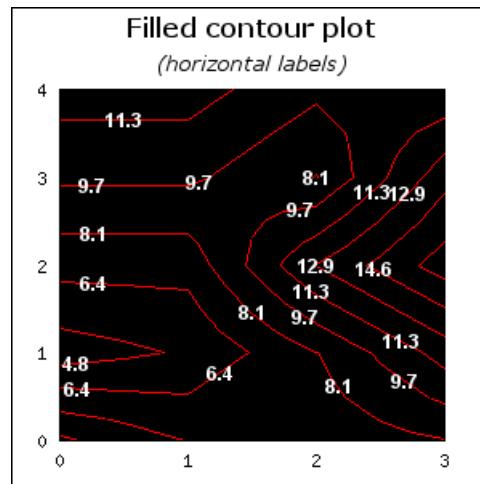
- `FilledContourPlot::ShowLabel($aFlg,$aFollowGradient)`

is used. The two examples below show two variants of the same contour where the left graph have labels which follows the gradient and the right graph have labels that are always oriented horizontally.

**Figure 23.4. Labels that follows Figure 23.5. Labels that are always the gradients (contour2\_ex4.php) [example\_src/contour2\_ex4.html]**



**Figure 23.5. Labels that are always horizontal. In this example we have also shown how to change the colors. (contour2\_ex5.php) [example\_src/contour2\_ex5.html]**



## 23.1.5. Selecting the adaptive method to use

### Note

This section can be skipped without loss of continuity. The default method "rectangular" recursive adaptation is good enough for most circumstances.

In order to fully understand the effect of selecting which adaptive method to use it is first necessary to understand some background on how contours are created algorithmically.

Determining general contours for a function of two independent variables is equivalent to the problem of finding the equivalence class of all coordinate pairs  $(x,y)$  for which for  $C=f(x,y)$  which in general is a non-linear problem. This is an example of a problem that is computationally very expensive to mathematically solve correct but quite tractable if we view a sampled version of the function. This means that we in general do not have access to  $f(x,y)$  instead we only know of its values at a number of grid points. This is also the cause of visually different contour plots for the same input data.

The difference lay in the core problem that in order to draw a continues contour we need to interpolate the unknown values in between the known sample points. This is therefore a degree of freedom where the actual choice of how we do the interpolation will cause the contour to be visually different depending on our choice.

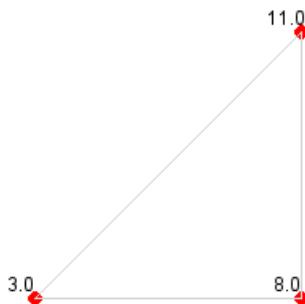
While it would take to far in this manual to describe the full adaptive algorithm in all details we will show the different principles that underlying the two methods hat the user can select between.

In order to determine the isobars the given data points are considered to be vertices in a grid where each vertex is connected with an edge. There are two basic ways to connect the points with edges; rectangular and triangular. In the first way four points are connected to form a square and in the second way three point are connected to form a triangle. The so created square or triangles will be referred to as a submesh consisting of four and three sides respectively.

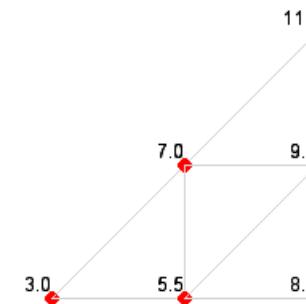
To determine the isobars each submesh is recursively divided until each side is small enough to only have one crossing isobar. The exact position of the crossing is determined by linear interpolation between the two corresponding vertices.

The figures below shows the sequences for the triangle recursive division. The recursion continues until each edge has exactly one crossing of an isobar.

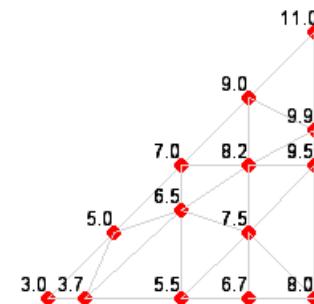
**Figure 23.6. Triangulation  
step 0**



**Figure 23.7. Triangulation  
step 1**

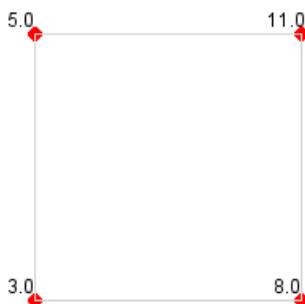


**Figure 23.8. Triangulation  
step 2**

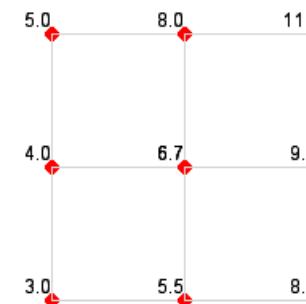


In the same way the sequence of figures below shows the principle for the quadratic recursive subdivision of the mesh for first three steps.

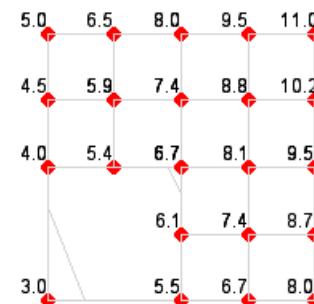
**Figure 23.9. "Rectangularization"  
step 0**



**Figure 23.10. "Rectangularization"  
step 1**



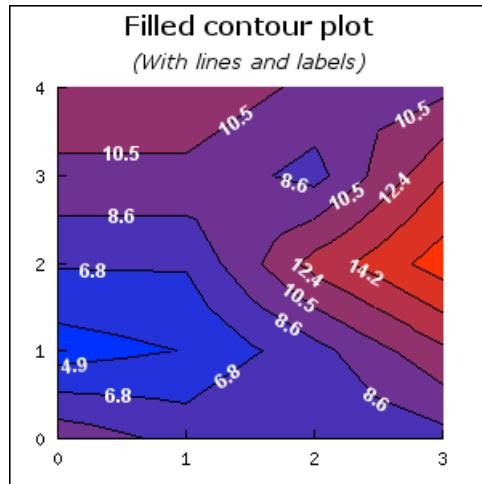
**Figure 23.11. "Rectangularization"  
step 2**



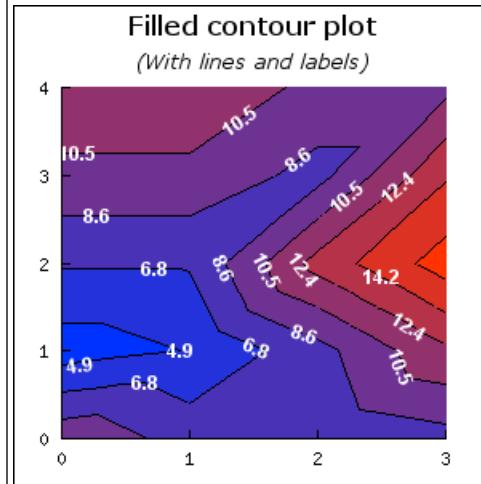
Needless to say these two variants give slightly different visual appearance of the resulting contour plot and neither of the two method can be considered "the right method". Triangularization tends to work better (and faster) for high frequency contour plots and gives a more "edgy" result. The quadratic subdivision might require more recursive steps for high frequency contours but in general gives a smoother look of the resulting contour.

By default the library uses a rectangular mesh division and that is the method that has been used in all the previous examples. As an illustration the effect of the method have the two figures below shows the same data as we have used in the previous figures to display a contour plot with 7 isobars. The left figures uses a rectangular division (the default) and the right figure uses a triangular division.

**Figure 23.12. 7 Isobars, "rect" method (contour2\_ex6.php) [example\_src/contour2\_ex6.html]**



**Figure 23.13. 7 Isobars, "tri" method (contour2\_ex7.php) [example\_src/contour2\_ex7.html]**



As can be seen in this case the triangular method favour elongated contours from southwest to northeast while the rectangular method favours circular contours.

The way to choose what method to use is by the method

- `FilledContourPlot::SetMethod($aMethod)`

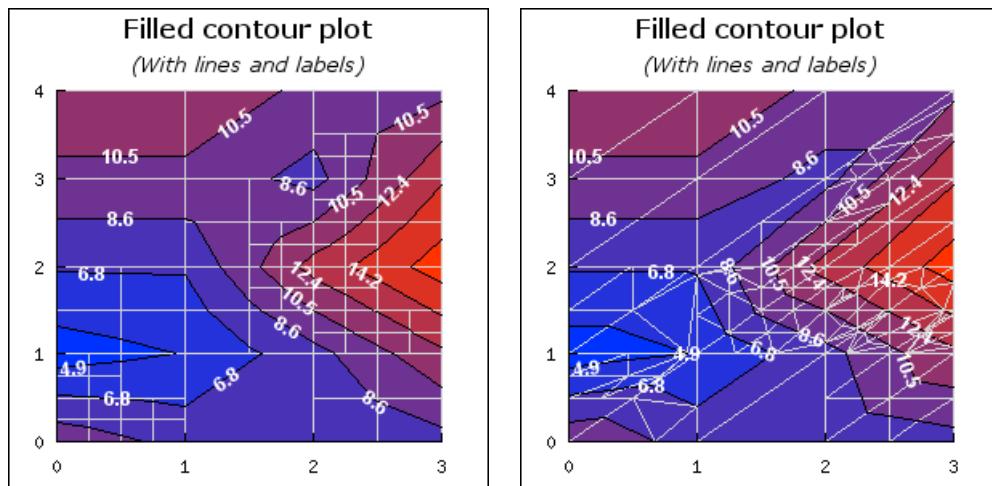
The possible options for `$aMethod` are

1. "rect" - Rectangular algorithm (default)
2. "tri" - Triangular algorithm

### Tip

It is possible to show the triangulation used by calling the method `FilledContourPlot::ShowTriangulation($aFlg)`. In the two figures below the triangulation for the "rect" and "tri" method in the previous examples are shown.

**Figure 23.14.** Rectangular sub- **Figure 23.15.** Triangular sub-division



---

## **Part VI. Barcodes**

For a good historic overview of barcodes and there usage we refer to Wikipedia [http://en.wikipedia.org/  
wiki/Barcode](http://en.wikipedia.org/wiki/Barcode)

---

# Chapter 24. Linear Barcodes (One Dimensional Barcodes)

## 24.1. Introduction

### Note

This module is only available in the pro-version of the library.

With the linear (One-dimensional) barcode extension it is possible to create bar codes using all the commonly accepted symbologies. The resulting bar code can be created as either an image (in PNG or JPEG format) or as a Postscript file ready for printing on high resolution printers.

The bar code extension provides extensive data verification which makes sure that the created barcode follows the official applicable standards.

### Note

All generated bar codes (except CODE 11 which have little support in Europe) have been extensively verified using Metrologic CCD-47 handheld scanner.

In order to create a linear barcode the module "`jpgraph_barcode.php`" must be included.

There are several types of linear barcode even though they all use the same principle. The reason for several different types (or symbologies as it is known) is partly to handle different requirements (for example encoding just numerical data or both numerical and alphabetical data) and partly because they were initially created by different companies to solve similar problems. The library supports most of the common linear barcode. These types of barcode look like what can be usually seen on consumer goods.

The different types of barcodes are often referred to as different symbologies.

Most of the commonly used symbologies has been elevated to ISO/IEC standards. Some barcode which have not yet been accepted as ISO standard are available from AIM (<http://www.aimglobal.org>). In USA the organization responsible for issuing retail codes used in barcodes are the *Uniform Code Council*, (UCC) <http://www.uc-council.org> [<http://www.uc-council.org/>].

While these standards are very comprehensive they are of little interest to end user of the barcode. For an end user the three most important question when selecting a barcode symbology are

1. What characters can be encoded in this symbology?
2. How efficient is the symbology, i.e. how large will the barcode be for a given input data string?

Linear barcodes typically encode alphanumerical strings up to maximum of ~20 characters.

### Note

While many of the barcode symbologies can handle, in theory unlimited, string lengths there are practical limitation to how wide barcode a given reader/scanner can interpret. Most handheld scanners can usually not reliable read a barcode wider than ~10cm.

3. How strong is the tolerance against physical damage, i.e. how large percentage of the barcode can be destroyed while still be readable by a scanner?

The tolerance for linear barcode are in general low. If one or more of the bars (making up the barcode) is unreadable the whole barcode is in practice unreadable. While some barcode symbologies include a check digit this is not enough to re-create damaged data. It is only enough to verify with some confidence that the data is intact, however there is still no guarantee since two errors might, by coincidence, make the check digit correct.

A common way to strengthen linear barcodes is to make them physically larger/taller.

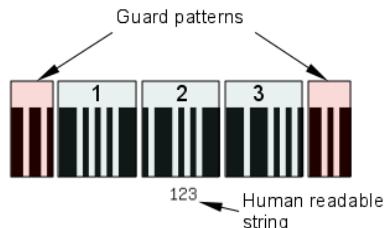
While linear barcodes are still widely used in legacy applications there are almost no new applications that uses linear barcode due in part to the limitations listed above (poor fault tolerance and low capacity). Most new applications uses either RFID tags or two dimensional barcodes (described in the following chapters) which have much higher data capacity and fault tolerance.

## 24.2. How does linear barcodes work?

Linear barcodes are created by translating the supported characters that should be displayed into combination of narrow and wide bars which are combined into a barcode. To identify the start and end of a barcode special "guard" patterns are used to indicate to the scanner that the barcode starts and also identify what type of symbology is used.

To illustrate this Figure 24.1, "Understanding linear barcodes. Example with Code 25 symbology" shows an enlarged and amended barcode that uses the Code 25 symbology and encodes the numeric string "123". In order to illustrate the different parts we have colored the start and stop pattern and the three digits to make it clear how it is encoded.

**Figure 24.1. Understanding linear barcodes. Example with Code 25 symbology**



We have chosen to illustrate the principle with the code 25 symbology since it is easy to understand. Each character is encoded with 5 black elements with spaces in between, 2 wide and three narrow elements, hence the name 2 of 5 (or 25). The width of the thinnest element is also known as the module width. Other symbologies have more complicated patterns to achieve higher density (more encoded characters in a given width) but the principle still stands.

The typical scanner sends out light which is reflected and is then measured as the scanner swipes from left to right. The relative size of the black and white areas are measured and translated back to the encoded data.

The human readable string, usually at the bottom of the barcode is strictly speaking not a part of the barcode. It is only there to help a human reader interpret the barcode.

Some barcode symbologies also add a check digit at the end of the data. This check digit (or digits) is used to verify that the interpreted data has been read correctly.

## 24.3. Barcode symbologies

### 24.3.1. Numerical only barcode symbologies

The following supported symbologies only supports numerical data and in some cases special characters like "-,+,:,\\$". A more detailed description and example of each barcode is given in Section 24.9, "Short description of supported symbologies"

- Codabar. Older code not widely used today apart from some library system primarily in USA.
- Code 11. Used primarily for labeling telecommunications equipment
- EAN-13 (ISO/IEC 15420). European Article Numbering, 13-digits. International retail product code  
(Also known as UPC-13 in USA)
- EAN-8. European Article Numbering, 8-digits. Compressed version of EAN code for use on small products
- Industrial 2 of 5. Older low density code not commonly used today
- Interleaved 2 of 5 (ISO/IEC 16390). Compact numeric code, widely used in industry, driving licenses, transportation
- UPC-A. Universal product code seen on almost all retail products in the USA and Canada
- UPC-E. Compressed version of UPC-A code for use on small products
- Bookland. Used to encode ISBN (International Standard Book Number) numbers used in book and magazines

### 24.3.2. Alphanumeric barcode symbologies

The following supported symbologies supports alpha numerical data and in some cases special characters like "-,+,&,#,!". A more detailed description and example of each barcode is given in Section 24.9, "Short description of supported symbologies"

- Code 128 (ISO/IEC 15417). A flexible high capacity code in wide use. There are variants of Code 128 (e.g EAN-128) that impose a strict structure of the data.
- EAN 128 (ISO/IEC 15420). Structured variant of Code 128. This is not really a barcode symbology in itself but rather a structure for how to format the input data to Code 128.
- Code 39 (ISO/IEC 16388). General purpose code. Used worldwide.

### 24.3.3. Which symbology should be used?

Usually the application dictates what standard should be followed and there is no choice other than following the applicable industry standard.

However, if the usage is strictly internal it is possible to chose any symbology. However for generic usage we would recommends using either Code 39 or Code 128 since they offer great flexibility, supports large character sets and is fairly efficient in terms of space/datasize ratio.

## 24.4. Features

This is a summary of the features available in the JpGraph barcode extension.

- Supports the following symbologies (with data validation)
  1. UPC A
  2. UPC E
  3. EAN 128
  4. EAN 13
  5. EAN 8
  6. CODE 11 (USD-8)
  7. CODE 39
  8. CODE 128
  9. Industrial 2 of 5
  10. Interleaved 2 of 5
  11. Codabar
  12. Bookland (ISBN)
- Input data is verified against the official specification of each symbologies and makes it impossible to create invalid bar codes.
- Output format
  1. Image format (either PNG or JPG encoding)
  2. Postscript
  3. Encapsulated postscript
- Formatting options
  1. User specified module width. This controls the width of the smallest unity in the barcode.
  2. Automatic calculation of optional checksum for symbologies where the checksum is optional
  3. User specified scaling of resulting bar code
  4. Horizontal or vertical bar code layout
  5. Suppression of human readable text on barcodes
  6. Selectable font for human readable text

## Note

The term "symbology" is the term used to describe the combination of encodation method (how characters are translated to bars) and the layout of the bars specified by a particular barcode standard. All of the listed standard symbologies above are still widely used. Many of the symbologies overlap in terms of functionality and multiple symbologies might be possible to use for a particular application unless there is a strict standard that stipulates that a particular symbology should be used.

The reason for the existence of so many symbologies with overlapping functionality is in some cases the result of different companies early on specifying their own patented barcodes in order to gain market shares or in some cases lock customers to a specific brand of scanner. In other cases it is simply the result of that earlier barcodes could not handle new demands.

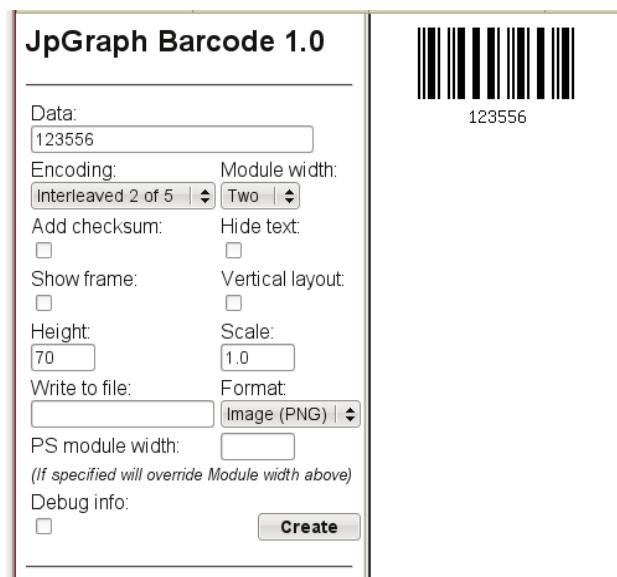
### 24.4.1. Sample application

Even though primarily the JpGraph bar code extension is meant, and designed, to be used as a library within a larger system there is a small demo web application to allow easy creation of linear barcodes bar codes. This application is primarily included as a demo of the features available and not as a finalized product.

The application is available under the directory "barcode/demoapp" as "index.html" in the main library directory

Figure 24.2, "Linear Barcode Demo application (screen shot from running in WEB-browser)" shows a screen shot of the demo application

**Figure 24.2. Linear Barcode Demo application (screen shot from running in WEB-browser)**



We strongly suggest running this demo application to get a feel for what formatting options and how the different symbologies behave.

## Tip

There is also a command line utility that allows the creation of barcodes from the command line "mkbarcode.php"

## 24.5. Creating barcodes - quick start

The creation of all linear barcode follows the same schema:

1. Create an instance of the encoder for the chosen symbology
2. Create an instance of the backend for the chosen output format (image or postscript)
3. Encode the data and generate the barcode

Normally only three lines of code is needed to create a basic barcode.

For example, the following code will create a barcode encoded as an image representing the data string "ABC123" using symbology "CODE 39".

```
<?php
require_once('jpgraph_barcode.php');

$symbology = BarcodeFactory::Create (ENCODING_CODE39);
$barcode = BackendFactory ::Create(BACKEND_IMAGE, $symbology);
$barcode ->Stroke('ABC123');
?>
```

The generated barcode is shown in

**Figure 24.3. Encoding "ABC123" with CODE 39.**



\*ABC123\*

As can be seen from the code above the basic interface to the library makes use of two abstract factories which creates the appropriate encoder and output backend. This design makes the addition of new output formats and new symbologies transparent for the end user of the library.

If instead we wanted to encode the data string using symbology "CODE 128" instead, it would only be necessary to modify the first line in the above code so instead it would become.

```
<?php
require_once('jpgraph_barcode.php');

$symbology = BarcodeFactory::Create (ENCODING_CODE128);
$barcode = BackendFactory ::Create(BACKEND_IMAGE, $symbology);
$barcode ->Stroke('ABC123');
?>
```

the result of this script is shown in Figure 24.4, "Encoding "ABC123" with CODE 128."

**Figure 24.4. Encoding "ABC123" with CODE 128.**



ABC123

As can be seen in the examples above both the backend and the symbology is specified by means of a symbolic constant. The following list shows the symbolic constants available to specify the different supported symbologies.

1. ENCODING\_EAN128
2. ENCODING\_EAN13
3. ENCODING\_EAN8
4. ENCODING\_UPCA
5. ENCODING\_UPCE
6. ENCODING\_CODE39
7. ENCODING\_CODE128
8. ENCODING\_CODE25
9. ENCODING\_CODEI25
10. ENCODING\_CODABAR
11. ENCODING\_CODE11
12. ENCODING\_BOOKLAND

The usage and typical application of each symbology is discussed in Section 24.9, “Short description of supported symbologies”.

## 24.6. Error handling

The barcode module uses the standard library error handling which mean that in case of an error (most likely that the data validation fails) an exception will be raised. Errors can be handled in two ways.

1. By enclosing the script in a `try { ... } catch { ... }` statement
2. By installing a custom default error handler with `set_exception_handler()` when the library throws an error the specified error handler will be called with an instance of the `JpGraphException` class.

The following code snippet shows an example of using a try-catch statement

```
<?php
try {
 $encoder = BarcodeFactory::Create(ENCODING_CODE39);
 $e = BackendFactory::Create(BACKEND_IMAGE,$encoder);
 $e->Stroke('abc123');
} catch(JpGraphException $e) {
 echo 'Error: ' . $e->getMessage() . "\n";
}
?>
```

The code when run will give the error

```
Error: Data validation failed. Can't encode [abc123] using encoding "CODE 39"
```

The problem with the input data string is that Code 39 does not support encoding lower case letters. In the case some error handling is still needed but the image error should be displayed it is possible to re-raise the original exception as the following example shows

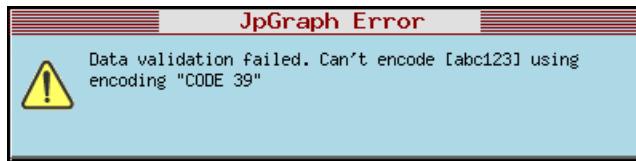
```
<?php
try {
 $encoder = BarcodeFactory::Create(ENCODING_CODE39);
 $e = BackendFactory::Create(BACKEND_IMAGE,$encoder);
 $e->Stroke('abc123');
} catch(JpGraphException $e) {
 JpGraphError::Raise($e->getMessage());
}
?>
```

The line

```
JpGraphError::Raise($e->getMessage());
```

will display the image error as shown in Figure 24.5, “Image error - Failed barcode data validation”

**Figure 24.5. Image error - Failed barcode data validation**



In the command line barcode utility ( Section 24.7, “Generating barcodes on the command line”) the alternative method of installing a different default error handler can be seen.

## 24.7. Generating barcodes on the command line

The library includes a command line utility as an additional example on how to use the library.

- `mkbarcode.php`

This utility can be found in the "barcode /" directory. This utility should be used from the command line as the following example shows

```
$>php mkbarcode.php -b code39 ABC123 > barcode.png
```

The line above will create a barcode from the data string "abc123" using the Code 39 symbology and store the resulting image in the file "barcode.png".

Using the command utility as above assumes that the client version of PHP is available at the command line.

This utility has the following syntax

```
mkbarcode.php -b <symbology> [-r -h -c -o <output format> -m <width> -s <scale>\\
-y <height> -f <filename>] datastring
```

```
Create the specified barcode
-b What symbology to use, one of the following strings (case insensitive)
 UPCA
 UPCE
 EAN128
 EAN13
 EAN8
 CODE11
 CODE39
 CODE128
 CODE25
 CODE125
 CODABAR
 BOOKLAND
-c Add checkdigit for symbologies where this is optional
-o Output format. 0=Image, 1=PS, 2=EPS
-m Module width
-s Scale factor
-h Show this help
-f Filename to write to
-r Rotate barcode 90 degrees
-y height Set height in pixels
-x Hide the human readable text
--silent Silent. Don't give any error messages
```

If no filename is given the image will be sent back to the console which means it should be redirected to a file.

## 24.8. Format options for barcodes

### 24.8.1. Adjusting the encodation process

For the encoding there is basically only one modification available.

For those symbologies that include an optional checksum it is possible to enable or disable this calculation.

The following symbologies may have optional checksum character(s)

1. Industrial 2 of 5
2. Interleaved 2 of 5
3. Code 39
4. Code 39 Extended
5. Code 11

Turning on/off checksum calculation for other symbologies will have no affect. Checksum calculation is enabled with a call to `AddChecksum()` on the chosen backend.

For example to augment the previous "CODE 39" example to include the checksum the code would be changed to

```
<?php
$symbology = BarcodeFactory::Create (ENCODING_CODE39);
$symbology->AddChecksum();
$barcode = BackendFactory ::Create('IMAGE' , $symbology);
$barcode->Stroke('ABC123');
?>
```

Which would give the result shown in Figure 24.6, “Encoding "ABC123" with CODE 39 adding checksum (checksum=4).”

**Figure 24.6. Encoding "ABC123" with CODE 39 adding checksum (checksum=4).**



## 24.8.2. Selecting output format

The output format can be adjusted by specifying/creating the appropriate backend. The library supports image and postscript (and encapsulated postscript) backends.

The backend is created by calling the static factory method

- `BackendFactory::Create($aBackend, $aEncoder, $aReport=false)`

So to create an image backend the following code is needed

```
$barcode = BackendFactory::Create (BACKEND_IMAGE , $symbology);
```

Where "\$symbology" is the chosen symbology as created by the `BarcodeFactory::Create()` factory method. Please note that both factory functions are called as static methods.

The output format is specified by using one of the following symbolic defines

1. `BACKEND_IMAGE`, Creates a standard JPEG or PNG (default) image
2. `BACKEND_PS`, Creates a standard postscript file as output. It is possible to modify this output to become EPS (Encapsulated postscript) by calling the `SetEPS()` method on the backend as the following code snippet shows

```
$barcode = BackendFactory::Create (BACKEND_PS,$symbology);
$barcode->SetEPS();
```

Please note that for the postscript backend the postscript code is returned as a string from the `Stroke()` method.

To send the created stream (either image or postscript) back to the browser or to a file the `Backend::Stroke()` method shall be used. The parameter to the `Stroke()` method shall be the string to be encoded.

Assume we want to create an image that is sent back to the browser. We would then use the following code

```
<?php
$symbology = BarcodeFactory::Create (ENCODING_CODE128);
```

```
$barcode = BackendFactory ::Create(BACKEND_IMAGE, $symbology);
$barcode->Stroke('ABC123');
?>
```

It is also possible to write the barcode directly to a file by specifying a second argument to the `Stroke()` method above. So if we instead wanted the barcode to be stored in the file "/tmp/barcode.png" we could write

```
$symbology = BarcodeFactory::Create(ENCODING_CODE128);
$barcode = BackendFactory ::Create(BACKEND_IMAGE, $symbology);
$barcode->Stroke('ABC123', '/tmp/barcode.png');
```

### Note

There is no automatic added extension to the file name.

Again, please note that for the Postscript background the `Backend::Stroke()` method normally returns the postscript file as a string if everything went well.

## 24.8.3. Writing barcodes to a file

This is done by adding a second argument, the file name, to the `Backend::Stroke()` method. This works for all backends. The file name should be an absolute path name. Since it is the PHP process that writes the file the permissions must allow the PHP process to write to the directory if PHP is called from a browser. If the command line version of PHP is used this does of course not apply.

## 24.8.4. Hiding the human readable text

- `Backend::HideText($aHide=true)`

The human readable text is the string that can optionally be displayed at the bottom of the bar. By default this is enabled.

## 24.8.5. Adjusting the module width

- `Backend::SetModuleWidth($aWidth)`

There are however some subtle facts regarding the module width and backend that needs to be explained.

1. For image type backends the module width specifies the number of pixels used for a module.
2. For Postscript (and Encapsulated PS) backends the module width specifies the width in points (i.e. 1/72 inch).

This also means that for image type backends only integer values makes sense.

### Caution

Depending on the quality of the printer (and paper) very small module width might not be readable with all bar code readers. For images it is therefore recommended to use "2" pixels as the minimum module width and for postscript output the minimum recommended width is "0.8" pt.

The following code shows how to both change the module width to 2 pixels and hide the human readable text

```
<?php
$symbology = BarcodeFactory::Create (ENCODING_CODE39);
$barcode = BackendFactory ::Create('IMAGE', $symbology);
$barcode ->SetModuleWidth (2);
$barcode ->HideText();
$barcode ->Stroke('ABC123');
?>
```

which would give the result shown in Figure 24.7, “Encoding “ABC123” with CODE 39, hiding the text.” below

**Figure 24.7. Encoding “ABC123” with CODE 39, hiding the text.**



## 24.8.6. Setting vertical or horizontal layout

- `Backend::SetVertical($aVertical=true)`

Will rotate the barcode 90 degrees to create a vertical view of the barcode.

## 24.8.7. Adjusting height of bar code

- `Backend::SetHeight($aHeight)`

The height of the bar codes is specified with the `Backend::SetHeight()` method. For images the height is interpreted as pixels and for postscript files it is interpreted as points (1 pt = 1/72 inch)

## 24.8.8. Scaling of bar codes

- `Backend::SetScale($aScaleFactor)`

The scale factor is real number and specifies a scale factor for the overall barcode image.

## 24.8.9. Add frame around bar code

- `Backend::ShowFrame($aFlag=true)`

This method will enable a frame around the edges of the barcode image

## 24.8.10. Examples of adjusting the output

The following example outputs a postscript file representing the bar code with a module width of 1.2 pt, using a vertical layout and scaling the image 2 times. For this example we are using CODE 39 with a checksum (which is automatically generated)

```
<?php
$symbology = BarcodeFactory::Create (ENCODING_CODE128);
$barcode = BackendFactory ::Create(BACKEND_PS, $symbology);
$barcode->SetVertical(true);
```

```
$barcode->Scale(2);
$barcode->SetModuleWidth(1.2);
$barcode ->Stroke('ABC123')) {
?>
```

## 24.9. Short description of supported symbologies

In the following section we will describe the requirements to encode in each of the supported symbologies. Some symbologies have strict standard for the data to be encoded. Trying to encode illegal (or not supported data) will result in a "Data validation" error as shown in Figure 24.5, "Image error - Failed barcode data validation".

### Warning

Please note that the information given here is in no way a replacement for the official AIM and ISO standards these encoding are based on. It is assumed that the reader have access to the official standards describing each of these encodings before applying a particular symbology.

### 24.9.1. UPC A

UPC A = Universal Product Code Version A.

UPC A encodes 12 numeric digits. The first digit identifies the number system used. The next group of 5 digits identifies the manufacturer. This number is assigned by the Uniform Code Council (UCC [<http://www.uc-council.org>]). The next 5 digits identify the particular product and are assigned by the manufacturer. The final digit is a check digit.

### Usage

Used for consumer goods. Is being slowly replaced by EAN13. UPCA is a special case of EAN 13 where the first digit is always 0.

### Input data/character set

Eleven digit. First digit is always 0.

### Checksum

Automatic, mandatory.

### Example

Encoded string = "03456781233"

**Figure 24.8. UPC A Example**



## 24.9.2. UPC E

UPC E = Universal Product Code Version E.

### Usage

Same general use as UPC A. Can be considered a subset and a more compact version of UPC A. Used where UPC A bar codes are too wide.

The 6 resulting digits are taken from the UPC A according to the following 4 rules:

1. If a manufacturer's number ends in 000 or 100 or 200, he has available to him 1,000 item numbers between 00000 and 00999. The six characters are obtained from the first two characters of the manufacturer's number followed by the last three characters of the item number, followed by the third character of the manufacturer's number.
2. If a manufacturer's number ends in 300, 400, 500, 600, 700, 800 or 900, he has available to him 100 item numbers between 00000 and 00099. The six characters are obtained from the first three characters of the manufacturer's number followed by the last two characters of the item number, followed by "3".
3. If a manufacturer's number ends in 10, 20, 30, 40, 50, 60, 70, 80 or 90, he has available to him 10 item numbers between 00000 and 00009. The six characters are obtained from the first four characters of the manufacturer's number followed by the last character of the item number, followed by "4".
4. If a manufacturer's number does not end in zero, then five item numbers between 00005 and 00009 are available. The six characters are obtained from all five of the manufacturer's identification number followed by the last character of the item number.

### Input data/character set

Eleven digits. First digit is always 0. Data input must follow UPC rule for construction of UPC E data.

### Checksum

Automatic, mandatory

### Example

Encoded string = "05510000120"

As an example of the smaller size of UPC E the same data string is also encoded with UPC A below.

**Figure 24.9. UPC E Encoding of "05510000120"**



**Figure 24.10. UPC A Encoding of "05510000120"**



## 24.9.3. EAN 8

General purpose short barcodes

## Usage

A shorter version of EAN13 using only 7 digits + checksum digit.

## Input data/character set

Seven digits.

## Checksum

Automatic, mandatory

## Example

Encode string = "3776221". Note the automatically added check digit.

**Figure 24.11. EAN 8 Example**



## 24.9.4. EAN 13

The name comes from that this symbology encodes 13 characters. It is widely used in the manufacturing industry.

The first two or three digits are a country code which identify the country in which the manufacturer is registered. The country code is followed by 9 or 10 data digits (depending on the length of the country code) and a finally a single digit checksum.

## Usage

Generic code for consumer goods. Is replacing UPC-A as worldwide standard.

## Input data/character set

Twelve digits.

## Checksum

Automatic, mandatory

## Example

Encoded string = "377622153812"

**Figure 24.12. EAN 13 Example**



## 24.9.5. EAN 128

### Note

The GS1 ([www.gs1.com](http://www.gs1.com)) organization has renamed several barcode standards built on Code 128 such as EAN-128 to GS1-128.

### Usage

EAN 128 is a CODE 128 where the data structure is strictly regulated. Please refer to the official EAN-128 specifications for details.

### Input data/character set

An alphanumeric data string following EAN 128 rules. This means for one thing that the string must start with a FUNC1 character which is encoded using ASCII 128 in JpGraph Barcode available as constant EA\_FUNC1. The validation routines checks these rules and will fail any data string not following the EAN 128 rules.

### Checksum

Automatic, mandatory.

### Example

Encoded string = "3125134772"

**Figure 24.13. EAN 128 Example**



## 24.9.6. Industrial 2 of 5

The names comes from that each character is encoded in 5 bars and 2 of those are 3 modules wide and the rest 1 module wide

### Usage

Old low density standard. For newer applications it should not be used. Use interleaved 2 of 5 instead since this is a higher density code.

### Input data/character set

Digits.

### Checksum

Optional.

## Example

Encoded string = "13729"

**Figure 24.14. Industrial 2 of 5, without check digit**



13729

**Figure 24.15. Industrial 2 of 5, with check digit**



137294

## 24.9.7. Interleaved 2 of 5

The name comes from the that each data character is composed of 5 elements, either 5 bars or 5 spaces. Two elements are 3 modules wide and three elements have a width of 1 module. Adjacent characters are interleaved, mixing the spaces from one character with the bars of the other.

### Note

This symbology is sometimes referred to as "Code 25" but since this could also refer to "Industry 2 of 5" symbology this name is ambiguous without further discriminating information.

## Usage

Various. Relative hight density numeric code. Used for example on some driving licenses.

## Input data/character set

Even length numeric string when not using checksum. Odd length string when using checksum.

## Checksum

Optional.

## Example

Encoded string = "137291"

Since using a checksum requires an odd number of digits we add a '0' in the beginning to get a data string with an even number of digits.

**Figure 24.16. Industrial 2 of 5, without check digit**



137291

**Figure 24.17. Industrial 2 of 5, with check digit**



01372915

## 24.9.8. CODE 11 (USD 8)

### Warning

#### THIS IS AN EXPERIMENTAL ENCODER

Due to the lack of CCD scanners supporting this code in Europe we have not been able to verify this Code against any scanning equipment.

### Usage

Numeric only code. Used primarily on telecommunication equipment. Not recommended for general usage. Note: This code is not widely supported by handheld CCD scanners.

### Input data/character set

Numeric including the dash "-" character.

### Checksum

Optional.

### Example

Encoded string = "0137291"

**Figure 24.18. Code 11, without check digit** | **Figure 24.19. Code 11, with check digit**



## 24.9.9. CODE 39

This is also known as "Code 3 of 9" or "USD 3" . This name comes from that each character consists of 9 elements. 5 bars and 4 spaces. Three of those elements are 2 modules wide and the rest one module wide. Code 39 is defined in *American National Standards Institute* (ANSI) standard MH10.8M-1983.

The full 128 ASCII character set can be encoded in Code 39.

Code 39 is a very widely used barcode and basically every scanner/reader on the market is able to decode Code 39 barcodes which makes it for a very good selection as a generic linear barcode.

### Usage

General alphanumeric data (Capitals only)

### Input data/character set

Alphanumeric (CAPITALS only) including the special characters "\$", "/", "+", "%" and "\*"

## Checksum

Optional.

## Example

Encoded string = "GRAPH12"

**Figure 24.20. Code 39, without check digit** | **Figure 24.21. Code 39, with check digit**



## 24.9.10. CODE 39 Extended

Code 39 Extended was developed to provide a means of encoding additional characters that are not normally part of the Code 39 character set (lower case characters and symbols).

## Usage

Same as Code 39 with the change that it supports lower case characters ("a,b,c,d,...") but do not support the special characters "+/%\$+\*".

## Input data/character set

Alphanumeric data.

## Checksum

Optional.

## Example

Encoded string = "Code39"

**Figure 24.22. Code 39 Extended, without check digit** | **Figure 24.23. Code 39 Extended, with check digit**



## 24.9.11. CODE 128

Named so since it can encode the entire 128 ASCII character set (from ASCII 0 to ASCII 128.)

## Usage

Generic high density code which supports the full ASCII set.

## Input data/character set

Alphanumeric characters.

## Checksum

Automatic, mandatory.

## Example

Encoded string = "Code128"

**Figure 24.24. Code 128**



## 24.9.12. CODABAR

Codabar is primarily used in libraries in the USA. It is also sometimes used in health care and transportation systems.

## Usage

Old, numeric + some alpha capability, code. Primarily used in the USA.

## Input data/character set

Numbers 0-9, Special characters "-\$:/.+"

## Checksum

None.

## Example

Encoded string = "12354"

**Figure 24.25. Codabar**



## 24.9.13. Bookland (ISBN)

This is a specially formatted EAN13 Code. This is primarily used to encode the *International Standard Book Number* (ISBN) on magazines and books.

### Usage

Encodes ISBN codes in books and magazines.

### Input data/character set

A valid ISBN number without last digit (check digit) and "-" signs. An ISBN is a 10-digit number which is made up of

- Group identifier (1 digit, typically 0 or 1 for English-speaking countries)
- Publisher identifier
- Title identifier
- Check digit

### Checksum

Automatic, mandatory.

### Example

The input data must be a valid ISBN (*International Standard Book Number*)

Encoded string = "0-486-63926-6"

**Figure 24.26. Bookland (ISBN)**



### Note

The ISBN check digit "6" is automatically removed and replaced by the EAN 13 check digit.

### Note

A bookland code will always have the digits "978" as the first three digits (the standard also allows "979" but that is currently not in use)

### Note

ISBN numbers are organized in Europe by:

International Standard Book Number Agency,  
Staatsbibliothek Preussischer Kulturbesitz

---

Linear Barcodes (One  
Dimensional Barcodes)

---

Posdamer Strasse 33  
D-1000 Berlin 30 Germany

---

# Chapter 25. PDF417 (2D-Barcode)

## 25.1. Principle of PDF417 Barcodes

### Note

This module is only available in the pro-version of the library.

### Caution

In order to use the PDF417 barcode module it is necessary for the PHP installation to support the function **bcmod()**. This is enabled when compiling PHP by making sure that the option `--enable-bcmath` is given when configuring PHP at compile time.

This first section gives a very brief explanation of the general structure of PDF417 barcodes and some capacity figures.

PDF417 was one of the first publicly available high density (capable of storing up to 2710 data characters) two dimensional barcodes. It was originally published by Symbol Technologies, Inc. but has since become an ISO standard. PDF417 belongs to the early two dimensional barcodes which internally consists of a number of linear barcodes stacked on top of each other. This is in contrast to the more modern two dimensional barcodes like *Datamatrix* and *QR code* which are truly two dimensional in that they have moved away from the row thinking in the internal construction of the barcode.

PDDF417 barcodes are extensively used for example within aviation, automobile industry and health care.

Strictly speaking it is not necessary to know this level of detail to use the PDF417 barcode module but we would recommend to read through this at least once since some parameters (like number of columns - explained below - that are used adjustable)

PDF417 is an acronym for *Portable Data Format 4 of 17* where 4 of 17 describes the structure of how a single data character is encoded (4 bars and 4 spaces in a 17 module wide structure).

### 25.1.1. PDF417 standard

The PDF417 is high capacity two dimensional barcode and is fully described in the official standard ISO/IEC 15438:2001 available for purchase from ISO Standard Organization [<http://www.iso.ch/iso/en/CombinedQueryResult.CombinedQueryResult?queryString=pdf417>].

### 25.1.2. Data capacity

PDF417 is a row based 2 dimensional barcode that consists of a maximum of 90 rows and 30 columns. The maximum number of data is dependent on

- The compaction mode used
- The number of columns (and rows)
- The error correction level

The maximum data size is dependent on both the compaction mode as well as the input data. The figures listed below will give some idea on the capacity

- 2710 digits in numeric compaction mode
- 1850 characters in text compaction mode
- 1108 bytes in byte compaction mode

One barcode can hold up to a maximum of 929 codewords (data count + data + error correction)

### 25.1.3. Structure of PDF417 barcodes

A high level overview of the structure of a PDF417 barcode is shown in Figure 25.1, “PDF417 Structure - Overview”. A PDF417 barcode can be thought of as a number of linear barcode stapled on top of each other. Each row in the barcode is constructed in a similar way.

Each data word (symbol character) consists of 4 bars and four spaces in a 17 module structure, hence the name PDF417. A more detailed explanation of a real PDF417 barcode is shown in Figure 25.2, “PDF417 Structure - Details of a real barcode”

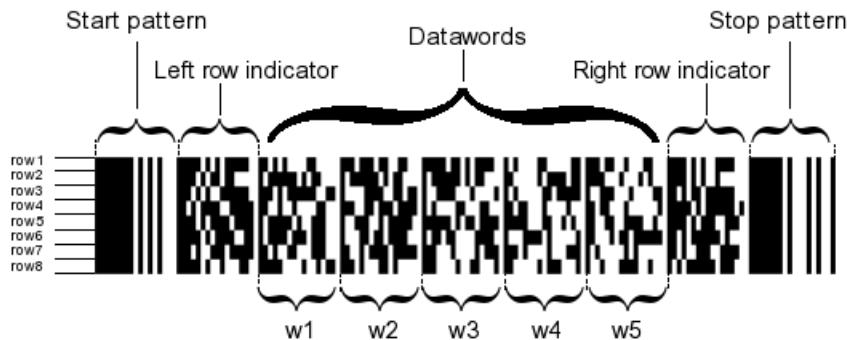
There are three distinct areas in a barcode:

1. **Start and stop pattern (light red background color).** Used to help the scanner find the start and beginning of the barcode. These patterns are static and are the same for all barcodes.
2. **Left and right row indicators.** Used to help the scanner orient itself in the barcode. These patterns are dependent on the actual data in the barcode to achieve maximum contrast.
3. **Data and data count.** This is unique for each barcode and represents the encoded data. PDF417 specifies several ways to encode the characters in the input data to achieve maximum compression level based on the knowledge (and restriction) on the input alphabet. For example, if the data is known to be only numeric the encodation can take advantage of this and make the compaction schema more efficient than if also alphabetical letters have to be encoded.
4. **Error correction codewords.** Each PDF417 have a user selectable error correction level. Since the barcode have a specified size this means that the more error correction words that are used the less data can fit. The error correction words are added to the end of the payload data, each barcode has a minimum of 2 error detection codewords. Up to 510 additional error correction codewords can be added for maximum data correction.

**Figure 25.1. PDF417 Structure - Overview**

Start pattern	Left row indicator	Data count	Datawords	Right row indicator	Stop pattern
Start pattern	Left row indicator		Datawords	Right row indicator	Stop pattern
...	...		...	...	...
Start pattern	Left row indicator		Datawords	Right row indicator	Stop pattern
Start pattern	Left row indicator	Datawords	Error correction	Right row indicator	Stop pattern

In Figure 25.2, “PDF417 Structure - Details of a real barcode” the distinct data column (which on each row holds one data word) are indicated at the bottom ( $w_1, w_2, w_3, w_4, w_5$ ). This particular barcode have 8 rows and 5 columns which means that the total number of data words + error correction words encoded are  $8 \times 5 = 40$ .

**Figure 25.2. PDF417 Structure - Details of a real barcode**

The data to be converted into a barcode has to go through a number of steps which are handled by the library:

1. The first step is a high level compression schema known as compaction. This schema translates the input string into a number of codewords. Each codeword has a numeric value between 0 and 928. To achieve the highest possible compaction and flexibility the PDF417 standard defined three different compaction schema:
  - numeric (encodes only digits 0-9, ASCII 30-39). This schema can compact up to 2.9 digits per codeword (and has the highest density)
  - byte (encodes ASCII 0-255). This schema can only compact up to 1.2 bytes per codeword (and has the lowest density)
  - text (which encodes ASCII 32-126). This schema can compact up to 1.8 characters per codeword.
2. The second step is the transformation of codeword into (4,17) symbols. The exact symbol used is dependent on which row the codeword to be encoded is on. Three different sets of codewords, known as clusters, are used. This ensures that two adjacent rows uses different clusters. This allows the barcode to be scanned without using specific divider symbols.
3. In the third step the error codewords specified are calculated and added to the end of the payload data. The error correction uses polynomial Reed-Solomon error correcting coding (same schema as used on CD:s) to achieve a good balance between error correcting efficiency and computational effort and space requirements.
4. Finally these codeword are positioned sequentially in each row starting at the top left corner down to the bottom right in between the left and right row indicators, and the start and stop patterns.

By necessity this is a fairly shallow description where we have omitted many technical details in the encodation process. We therefore refer to the official standard which gives much more technical details on the encodation process.

## 25.2. Creating barcodes

### 25.2.1. Introduction

The library allows the creation of PDF417 barcodes as either images (in either PNG or JPEG format) or as a Postscript files. In addition the barcode extension provides extensive data verification which makes it impossible to create an invalid barcode

## Verification

The JpGraph PDF417 barcode module have been extensively verified using an ARGOX 8213 2D CCD handheld scanner.

## Features

This is a summary of the features available in the JpGraph PDF417 barcode extension.

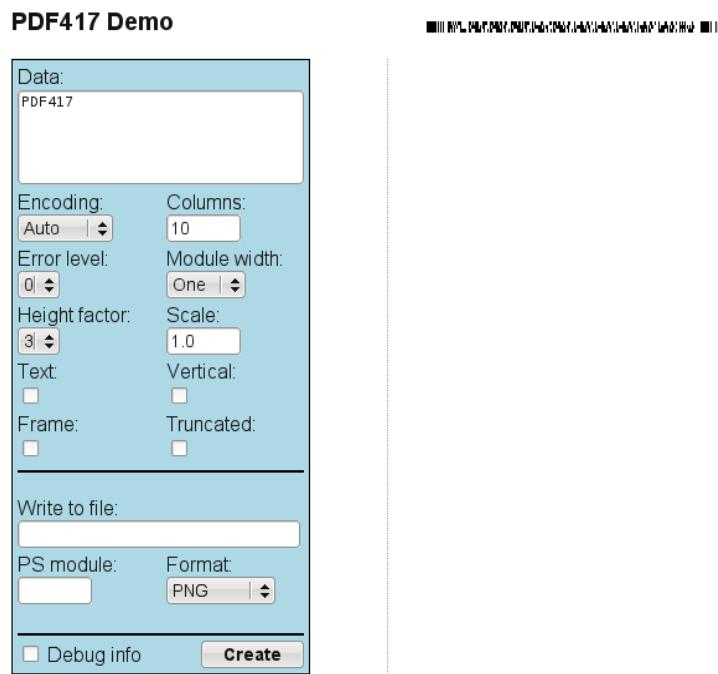
- Supports both standard and truncated PDF 417 codes
- Output format
  1. Image format, (any of the formats supported by the library)
  2. Postscript
  3. Encapsulated postscript
- Supports both auto and user selectable number of columns and rows
- Supports all 8 possible error correction levels
- Supports all defined compaction schemes, Alpha, Numeric and Byte. In addition the library has an optimization algorithm that will determine the best possible encoding given a specific input data.
- User specified module height
- User specified module width factor
- User specified overall image scaling of resulting barcode
- Horizontal or vertical barcode layout
- Optional human readable text at the bottom of the barcodes with selectable font and size
- Foreground and background color specification

## Sample application

Even though the library module is meant, and designed, to be used as a library within a larger system there is a small demo barcode creation application included in the distribution. This application can be used to easily create barcode through it's WEB interface as can be seen in Figure 25.3, "PDF417 WEB-based demo application"

This application is primarily included as a demo on the features available in the library and not as a finalized product.

The application is available at "`pdf417/demoapp/index.html`" in the distribution.

**Figure 25.3. PDF417 WEB-based demo application**

The fields in the demo application has the following meaning

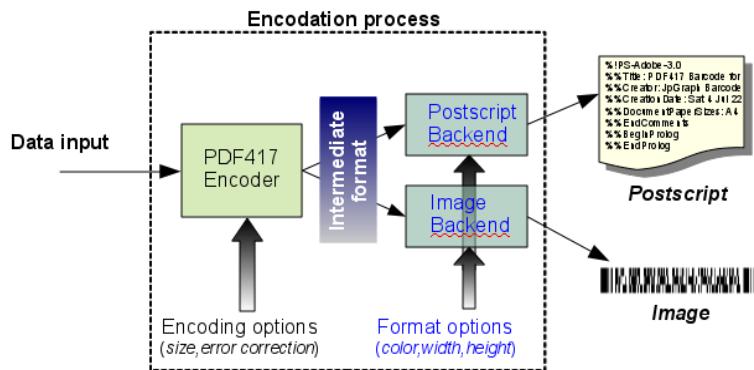
- **Data:** Data to be encoded. This should be entered as a standard string.
- **Encoding:** Determine the compaction schema to be used
- **Columns:** Number of data columns to be used
- **Error level:** What error level to be used
- **Module width:** Module width
- **Height factor:** Specify the height of the individual rows in the barcode as a multiplier of the width.
- **Scale:** Scale factor.
- **Text:** Should the human readable text be shown?
- **Vertical:** Should the barcode use vertical layout?
- **Frame:** Should a frame be added to the barcode?
- **Truncated:** Should the barcode be truncated?
- **File:** If specified the output will be written to this file
- **PS Module:** If specified will be used as the module width for PS
- **Format:** If specified will set what image encoding will be used output
- **Debug info:** Will open a new frame and display some internal debugging information from the library.

## 25.3. Creating barcodes

In order to access the PDF417 functionality the module "pdf417/jpgraph\_pdf417.php" must first be included.

Usage of PDF417 barcodes follows a similar schema as for the linear barcodes with the concepts of an encoder and backend. The principle of the overall encodation process is shown in Figure 25.4, "Overview of the interaction between encoder and backends"

**Figure 25.4. Overview of the interaction between encoder and backends**



In order to create a PDF417 barcode the following principle steps are needed

1. Create an instance of the PDF417 encoder (as an instance of class `PDF417Barcode`)
2. Create an instance of a backend to produce the wanted output (image, postscript or encapsulated postscript) by using the static factory method `PDF417BackendFactory::Create()`
3. Encode data and send it back to the browser or a file with a call to the backend `Stroke()` method.

The following script shows how to create the simplest possible barcode (in PNG format) representing the data string "PDF-417" encoded using default number of columns and the default error correction level

**Example 25.1. The most basic PDF417 script (`pdf417_ex0.php`)**

```

<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data = 'PDF-417';
// Create a new encoder and backend to generate PNG images
$backend = PDF417BackendFactory::Create(BACKEND_IMAGE,new PDF417Barcode());
$backend->Stroke($data);
?>

```

**Figure 25.5. The most basic PDF417 script (`pdf417_ex0.php`) [`example_src/pdf417_ex0.html`]**



As can be seen from the code above the basic interface to the library makes use of one abstract factory to create the appropriate output backend. This design makes the addition of new output formats transparent for the end user of the library.

The example above does not have any error handling. If there is some error in the process an exception will be thrown in the same way as in other places in the library. The default exception will display the standard library image error box. If we instead always wanted to do some additional processing and perhaps just display a text based re-formatted error message we could change the above code to catch this exception as the following example shows

### Example 25.2. Adding error handling (`pdf417_ex1.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data = 'PDF-417';
try {
 // Create a new encoder and backend to generate PNG images
 $backend = PDF417BackendFactory::Create(BACKEND_IMAGE,new PDF417Barcode());
 $backend->Stroke($data);
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
?>
```

**Figure 25.6. Adding error handling (`pdf417_ex1.php`)** [example\_src/  
`pdf417_ex1.html`]



when an exception is thrown it will be caught and the error string echoed. If instead we wanted to specifically encode the data using 8-columns and using error detection level 5 the code would have to be modified to

### Example 25.3. Adjusting the number of columns and error correction level (`pdf417_ex1b.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data = 'PDF-417';

// Setup some symbolic names for barcode specification

$cOLUMNS = 8; // Use 8 data (payload) columns
$ERRLEVEL = 2; // Use error level 2 (minimum recommended)

// Create a new encoder and backend to generate PNG images
try {
 $encoder = new PDF417Barcode($cOLUMNS,$ERRLEVEL);
 $backend = PDF417BackendFactory::Create(BACKEND_IMAGE,$encoder);
 $backend->Stroke($data);
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
?>
```

**Figure 25.7. Adjusting the number of columns and error correction level (pdf417\_ex1b.php) [example\_src/pdf417\_ex1b.html]**



Later on we will go through all the different options available both on the encoder and on the backends. But for now let's just show how easy it is to change the size of the barcode and add a human readable text at the bottom of the bar from the example above.

In the same way as for linear barcode the concept of "module width" is used. This is basically the width of a unit bar in the barcode. Remember that each codeword is made up of 17 modules where there are 4 black and 4 white areas of various width. The width of the module is specified with the backend method

- Backend::SetModuleWidth(\$aWidth)

For images this is specified in pixels and for a postscript backend this is interpreted as specifying the number of points (1 pt = 1/72 inch).

To add a human readable version of the data at the bottom of the barcode we use the method

- Backend::ShowText(\$aShow=true)

Adding these tow method calls to our previous example will give us the following code and resulting barcode.

#### **Example 25.4. Adjusting the module width and showing human readable text (pdf417\_ex2.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data = 'PDF-417';

// Setup some symbolic names for barcode specification

$columns = 8; // Use 8 data (payload) columns
$errlevel = 2; // Use error level 2 (minimum recommended)
$modwidth = 2; // Setup module width (in pixels)
$height = 3; // Height factor

// Create a new encoder and backend to generate PNG images
try {
 $encoder = new PDF417Barcode($columns,$errlevel);
 $backend = PDF417BackendFactory::Create(BACKEND_IMAGE,$encoder);
 $backend->SetModuleWidth($modwidth);
 $backend->SetHeight($height);
 $backend->Stroke($data);
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
?>
```

**Figure 25.8. Adjusting the module width and showing human readable text (pdf417\_ex2.php) [example\_src/pdf417\_ex2.html]**



In the remainder of this chapter we will explain in more detail what other formatting options are available.

## 25.4. Specifying the PDF417 parameters

### Note

The following section is not meant as a general introduction to the way PDF417 barcode is specified. It assumes that the reader has a basic understanding of the nature of PDF417 encoding. This section will focus on how to make use of the methods in the library to specify the various settings for the PDF417 barcodes.

### 25.4.1. Specifying encoding and input data

The absolute simplest way of encoding data is simply to create a simple string representing the data to be encoded and then pass that string as the first argument to the `Stroke()` method in the backend. The encoder will then analyze the input data and choose the most efficient space saving encoding schema for this data.

The PDF417 standard allows 3 different compaction schema that can be used to minimize the number of codewords used for a particular data string. This also means that a particular data string may have several different valid barcodes that visually looks different.

The available compaction modes are:

- **Alpha compaction mode** (also known as Text compaction mode). Efficient encoding of ASCII 32-126, (inclusively) i.e. the normal alphabet including numbers. Encodes 1.8 characters per codeword.
- **Numeric compaction mode.** Efficient encoding of numeric data. For long consecutive strings of digits this gives a better compaction than the alpha mode. Numeric compaction encodes about 2.9 digits per codeword.
- **Byte compaction mode.** The least efficient encoding. Used only when there is a need to encode byte values as is, i.e. values in the range 0-255. Please note that many barcode readers, especially those with a keyboard wedge, don't send back the proper encoding for ASCII values lower than 32 or higher than 126. Byte compaction mode encodes roughly 1.2 byte per codeword.

When the automatic encoding is chosen this will create an optimum encoding (from a size perspective) of the supplied data. This includes shifting encoding method in the middle of the data one or several time depending on the structure of the data.

It is also possible to manually control the exact encodation of the input data. This is done by supplying one or more data tuples where the first entry in the tuple is the compaction schema and the second the data. To encode the data manually the following structure must then be followed:

**Figure 25.9. Structure for manually specified encodation schema**

```
$data = array(array(<encoding_mode1> , <data1>),
 array(<encoding_mode2> , <data2>),
 ...
 array(<encoding_modeN> , <dataN>));
```

The encoding mode is specified as one of three symbolic constants

- USE\_TC, (short for **USE-TextCompaction**) Use Text compaction schema
- USE\_NC, (short for **USE-Numerical-Compaction**) Use Numeric compaction schema
- USE\_BC, (short for **USE-ByteCompaction**) Use Byte compaction schema

and the data is specified as a regular text string. Each section of data must therefore have the compaction mode specified.

### **Example 1:**

In the following example we will assume that we want to encode the data string

```
$data="1234567890"
```

To use automatic encoding then there is nothing more than specifying this data to the `Backend::Stroke()` method as

```
$backend->Stroke($data);
```

If instead we wanted to make sure that only alpha mode (text) compaction schema is used the input data would have to be changed to

```
<?php
$data="1234567890"
$newdata = array(array(USE_TC, $data));
...
$backend->Stroke($newdata);
?>
```

this will then force the input string to be encoded using only the text compaction schema.

If instead we wanted to enforce only numeric compaction the code would have to be changed to

```
<?php
$data="1234567890"
$newdata = array(array(USE_NC, $data));
...
$backend->Stroke($newdata);
?>
```

this will then force the input data string to be encoded using numeric compaction schema. #

In the above example we just used a single compaction schema to use multiple encodation schema we just need to split our data for each of the compaction mode we want to use and create an input array. An example will make this clear.

### **Example 2:**

We will assume that we want to encode the string "1234abc567" by using numeric compaction for the first 4 digits, then use text compaction for the three letters and finally go back to numeric compaction schema for the last three digits. For this to work we would have to create an input array as shown below.

```
<?php
$newdata = array(array(USE_NC, '1234'),
 array(USE_TC, 'abc'),
```

```

 array(USE_NC, '567'));
...
$backend->Stroke($newdata);
?>

#

```

## Note

Normally there are very few reasons to specify the encodation schema manually and it is therefore better to let the library determine the optimum encoding by itself.

## Using byte compaction mode

Using byte compaction mode is however slightly more complex. The reason is that we need, for technical reasons, specify if the size (length) of the data to be encoded is an even multiple of 6 or not.

Hence, there are actually two Byte code compaction schema

1. USE\_BC\_E6 (for even multiples of 6)
2. USE\_BC\_O6 (for odd data).

So to encode data using byte compaction mode the following template should be used to determine the proper byte compaction variant.

```

<?php
$even6 = (strlen($data) % 6 === 0);
$newdata = array(array($even6 ? USE_BC_E6 : USE_BC_O6, $data));
...
$backend->Stroke($newdata);
?>

```

## Caution

Remember that `strlen()` is not multi byte character encodation safe. If multi-byte characters should be encoded then the `mb_strlen()` should be used.

## Caution

Note that several keyboard wedge barcode scanners do not handle byte values < 32 or > 127 properly.

## 25.4.2. Encoder option: Adjusting the number of data columns

PDF417 barcode is made up of a number of rows and columns. The library allows the specification of the number of columns and it will then determine the necessary number of rows to hold all the given data + the error correction information.

Since each row has some overhead (start/stop and sync codewords) the overall area taken by the barcode will be minimized by trying to use as many columns as possible. The standards allow for up to 30 columns (and 90 rows). The most practical limit is how wide data the scanner is able to handle. Most hand hold scanner will usually not work very reliable with barcodes which are more than ~10cm wide.

## 25.4.3. Encoder option: Adjusting the error level

All PDF417 barcodes have a minimum of two error detection codewords. Above that the user is free to specify a higher level which will allow not only error detection but also (some) error correction.

The error level determines how much redundancy is added in the barcode label. A high level of redundancy ensures that a partially damaged barcode can still be correctly read by the barcode scanner. The downside is that the higher the error level the larger the barcode gets and since the total number of codewords in a PDF417 barcode has a maximum limit of 928 also less real data. Table 25.1, “Available error levels” shows the available error levels and how that will impact the maximum data payload. Table 25.1, “Available error levels” also shows the error correcting capacity. For example using error level 4 means that 15 of the codewords can have errors and still be corrected.

**Table 25.1. Available error levels**

Error level	Error correction codewords	Error correction capacity	Maximum payload
0	2	0	923
1	4	1	921
2	8	3	917
3	16	7	909
4	32	15	893
5	64	31	861
6	128	63	797
7	256	127	669
8	512	255	413

The recommended minimum error level is a dependent on the payload size and is given below.

**Table 25.2. Recommended error levels**

Data codewords	Recommended error level
1 to 40	2
41 to 160	3
161 to 320	4
321 to 863	5

Note that the number of codewords is not the same thing as, for example, the number of digits or letters in a string to be encoded. Depending on the chosen encoding the number of symbols per codeword is always > 1. For example in numeric compaction mode (encoding) each codewords encode, on average, 2.93 digits.

The error level is specified as an integer in the range [0-8] inclusively and can be specified when creating a particular encoder. For example the code below uses the default error correction (2).

```
<?php
// Use 10-columns for data
```

```
$columns = 10;

// Create a new encode using the default error correction
$encoder = new PDF417Barcode ($columns);
?>
```

While the following specifies an error correction level of 6

```
<?php
$columns = 10; // Use 10-columns for data
$errlevel = 6; // Error correction level 6

// Create a new encode using the default error correction
$encoder = new PDF417Barcode ($columns, $errlevel);
?>
```

In addition to specifying the number of data columns and error level in the creation of the encoder it is also possible to adjust them afterwards.

For example, it might be necessary to create the encoder in the beginning of a script and then use the same encoder with different settings controlled by, for example, entries in a DB.

The two encoder methods

- `Encoder::SetErrLevel($aErrLevel)`
- `Encoder::SetColumns($aColumns)`

are used for this purpose. The code snippet below does the exact same things as the code snippet above but using these two methods after the encoder has been instantiated instead.

```
<?php
$columns = 10; // Use 10-columns for data
$errlevel = 6; // Error correction level 6

// Create a new encode using the default error correction
$encoder = new PDF417Barcode();
$encoder->SetColumns($columns);
$encoder->SetErrLevel($errlevel);
?>
```

## 25.4.4. Truncated PDF417

### Warning

Not all PDF417 barcode readers can handle truncated PDF417

In situations where the physical size of the label is restricted one might use the truncated version of the PDF417 code.

This works by simply stripping of some redundant information on the right side of the barcode. This will also make the barcode more sensible for damage.

The two images below shows a normal version together with the truncated version (both barcodes encode the same information).

**Figure 25.10. Normal PDF417****Figure 25.11. Truncated PDF417**

To use the truncated version the method the following encoder method is used

- `Encoder::SetTruncated($aFlg=true)`

The following code snippet shows how this can be used

```
<?php
$colums = 10; // Use 10-columns for data
$errlevel = 4; // Error correction level 4
$truncated = true;

// Create a new encode using the default error correction
$encoder = new PDF417Barcode();
$encoder->SetTruncated($truncated);
$encoder->SetColumns($colums);
$encoder->SetErrLevel($errlevel);
?>
```

## 25.5. Adjusting the output

This section deals with the way you can adjust the visual quality and appearance of the output by adjusting a number of properties on the backend.

### 25.5.1. Output format

By choosing the appropriate backend you can choose to generate the barcode label as either an image (in either PNG or JPEG format) or as Postscript (both standalone and Encapsulated postscript). For images the PNG is strongly recommended and used by default. The reason is that PNG is a non-destructive format and will ensure the maximum quality of the barcodes while JPEG is not well suited for this type of applications since it is a destructive format.

This choice is being made by creating the appropriate backend. The backend is created by calling the factory method `PDF417BackendFactory::Create()` specifying what type of backend and what encoder to use. The backend factory will then return a suitable encoder.

The following code snippet shows how to create a backend that will generate an image.

```
<?php
// Create a new encode using the default error correction
// and columns
$encoder = new PDF417Barcode ();
$backend = PDF417BackendFactory::Create(BACKEND_IMAGE, $encoder);
?>
```

It is also possible to have one encoder and two different backends to allow the creation of both image and postscript output at the same time as the following example shows

```
<?php
```

```
$date='ABC123';
$file=' ...'; // Some filename to write the PS file to

// Create a new encode using the default error correction
// and columns
$encoder = new PDF417Barcode();

$eImg = PDF417BackendFactory::Create(BACKEND_IMAGE, $encoder);
$eEPS = PDF417BackendFactory::Create(BACKEND_EPS, $encoder);
$eEPS->Stroke($data,$file);
$eImg->Stroke($data);
?>
```

## 25.5.2. Summary of user settings for the backend

In the list below is a short description of the available possibilities to change the output. For a more detailed explanation of the parameters for each method please consult the method reference at the end of this chapter

- **Specifying the module width.** The module width for the barcode is user selectable, For images this specifies the number of pixels for the module and for Postscript output this specifies the number of points (1/72 inch).
- **Specifying the width/height factor** Specify the height of each individual row in the barcode as a multiple of the width. By default the height is 3x the width.
- **Specifying background and foreground color** The foreground color and the background color for the barcodes are user selectable. The colors can be specified as any of JpGraphs builtin colors.
- **Adding human readable text** This is an extension to the PDF417 standard. This adds a plaintext string of the data that has been encoded under the barcode. This is useful when debugging applications to make sure that the right values have been read, for example, from a DB.
- Rotating 90 degrees (Vertical output) This enables the barcode to be printed vertically.
- Scaling the image The resulting barcode images can be scaled by an arbitrary factor.

## 25.6. A template to create barcodes

In the example directory in the distribution ('pdf417/examples') you can find many more examples on how to create barcodes. As a good start the following (simple) template may be used as a base for further customizations.

```
<?php
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data = 'PDF-417';

// Specification for barcode

$columns = 8; // Use 8 data (payload) columns
$errlevel = 4; // Use error level 4
$modwidth = 2; // Setup module width (in pixels)
$height = 2; // Height factor (=2)
$truncated = false; // Don't truncate
```

```

$showtext = false; // Show human readable string

// Create a new encoder and backend to generate images
try {
 $encoder = new PDF417Barcode($columns,$errlevel);
 $encoder->SetTruncated ($truncated);
 $backend = PDF417BackendFactory::Create(BACKEND_IMAGE,$encoder);
 $backend->SetModuleWidth($modwidth);
 $backend->SetHeight($height);
 $backend->NoText(!$showtext);
 $backend->Stroke($data);
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
?>

```

## 25.7. Method reference

### 25.7.1. Encoder methods

All encoder methods are accessed on an instance of class `PDF417Barcode`

```

...
$encoder = new PDF417Barcode($columns,$errlevel);
...
```

#### **PDF417Barcode**

`PDF417Barcode($aNumCols=10, $aErrLevel=2)`

**Purpose:**

To create a new PDF417 encoder with a chosen number of columns and error level. Please note that these parameters can also be adjusted using the methods described below. The method will throw exceptions if the parameters are outside their valid range.

`$aNumCols`, an integer in the range [2,30]

`$aErrLevel`, an integer in the range [0,8]

**Returns:**

A new encoder which is used by the backend to generate the barcode label.

**Example:**

```

// Create a new encoder with 20 columns
$encoder = new PDF417Barcode(20);
```

#### **SetColumns**

`SetColumns($aCols)`

**Purpose:**

Specify number of data columns. This should be a value in the range [2-30] (inclusively) . Note that it is also possible to specify the number of columns in the instantiation of the encoder.

**Example:**

```
// Create a new encoder with 20 columns
$encoder = new PDF417Barcode();
$encoder->SetColumns(20);
```

**Note:**

The number of rows will be automatically adjusted to fit the data and the error correction codewords as specified by the error level setting. The number of rows is not directly user configurable.

## SetErrLevel

```
SetErrLevel($aErrLevel)
```

**Purpose:**

Specify the chosen level of error correction.

**Example:**

```
$encoder = new PDF417Barcode();
$encoder->SetErrLevel(5);
```

**Note:**

A high error level will limit the number of payload data since the total number of data in the label is fixed.

## SetTruncated

```
SetTruncated($aTrunc=true)
```

**Purpose:**

Specify that barcode should use the truncated PDF format. This will make the barcode slightly narrower and might be used where the physical space is at premium.

**Example:**

```
$encoder = new PDF417Barcode();
$encoder->SetTruncated();
```

**Note:**

Not all barcode scanners can handle truncated PDF417.

## 25.7.2. Common backend methods

The backend is responsible for the actual output of the barcode (see Figure 25.4, “Overview of the interaction between encoder and backends”).

It is possible to either create an image or a postscript backend. In the creation of the backend an instance of the encoder is given as the second argument. The actual output from the script (to create the barcode

label) is done with a call to the `Backend::Stroke()` method. As usual this call should be the last line in the script if it is used to send the image back to the browser.

The method call to `Stroke()` should always be guarded with a `try{ } catch { }` statement.

**Example:**

```
try {
 $encoder = new PDF417Barcode($columns, $errlevel);
 $backend = PDF417BackendFactory::Create(BACKEND_IMAGE, $encoder);

 // .. Output formatting
 $backend->Stroke();
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
```

In the example above we have chosen to create an image encoder (specified by constant `BACKEND_IMAGE`) by default the image format used will be PNG which is also the recommended format.

## Stroke

`Stroke($aData, $aFile='')`

**Purpose:**

Encode a specific data string entered as the first argument and send the barcode output, depending on backend, either directly back to the browser as an image or return the Postscript representation as a string.

**Example:**

```
<?php
// Data to be encoded
$data = ... ;

try {
 ...
 $backend->Stroke($data);
} catch(JpGraphException $e) {
 ...
}
?>
```

## SetVertical

`SetVertical($aFlg=true)`

**Purpose:**

Draw the barcode vertically instead of the default horizontal direction

**Example:**

```
$backend->SetVertical();
```

## SetScale

```
SetScale($aScale)
```

**Purpose:**

Will arbitrary scale the generated barcode image. Scale factor is specified as a floating point number .

**Example:**

```
$backend->SetScale(2.5);
```

## SetModuleWidth

```
SetModuleWidth($aWidth)
```

**Purpose:**

Specifies the module width to be used in the barcode. The module width is interpreted differently depending on whether it is applied on an image or a postscript backend. The module width will determine the physical width of the overall barcode. Please make sure that the printer has good enough resolution to handle the chosen module width.

For image backends is specifies the number of pixels to be used for the width of one module in the barcode. The module can be thought of as the thinnest line printed in the barcode. Each codeword is encoded 17 modules wide where there are 4 black and 4 white separated areas.

Even though most modern inkjet printers (2003 and later) can resolve modules down to 1 pixels it is recommended that for inkjet printers a module width of >= 2 is used.

For Postscript backend the module width specified the width in Postscript points (1 point = 1/72 inch).

**Example:**

```
$backend->SetModuleWidth(2);
```

### Tip

The size of the barcode image can be fine tuned by applying an overall scaling constant as specified with the method `SetScale()`.

## SetHeight

```
SetHeight($aHeight)
```

**Purpose:**

Specified the height to width ratio for the individual rows in the barcode. By default the ratio is 3, which means that the height of the rows in the barcode is 3 times the module width. Most scanners can handle a factor between 2-5. You should very rarely have to change this parameter.

The ratio is specified as an integer.

One reason to change this to a smaller value is to make a barcode label take up less vertical space.

**Example:**

```
$backend->SetHeight(2);
```

## Caution

Not all barcode scanners are good at handling very small ratios. This will also make the barcode more susceptible to damages.

## SetColor

```
SetColor($aFrgColor,$aBkgColor)
```

### Purpose:

Specify the foreground and background color for the barcode.

### Example:

```
$backend->SetColor('black', 'lightyellow');
```

## ShowFrame

```
ShowFrame($aFlg=true)
```

### Purpose:

Draw a 1 pixel frame around the barcode. This is not recommended in production since it might disturb some scanners reading. This is only useful if you need a visual feedback on the boundaries for the image.

### Example:

```
backend->SetFrame();
```

## ShowText

```
ShowText($aFlg=true)
```

## Caution

This is an extension specific to JpGraph. Human readable text is not part of the official PDF417 standard and hence this is disabled by default.. There are usually no problems of including this text but it is not recommended to do so in a production environment. This is mostly useful when debugging an application and to make it easier to see that the correct data is encoded as a barcode label.

### Purpose:

Show human readable text of the encoded data underneath the barcode label.

### Example:

## SetFont

```
SetFont($aFontFam,$aFontStyle,$aFontSize)
```

### Purpose:

Specify the font for the human readable text underneath the barcode label.

**Example:**

```
$backend->SetFont(FF_ARIAL, FS_NORMAL, 8);
```

### 25.7.3. Image backend methods

#### SetImgFormat

```
SetImgFormat($aFormat)
```

**Purpose:**

Specify image format. Possible values are

- 'auto'
- 'png'
- 'jpeg'
- 'gif'

**Example:**

```
$backend->SetImgFormat('jpeg');
```

### 25.7.4. Postscript backend methods

#### SetEPS(\$aFlg=true)

```
SetEPS($aFlg=true)
```

**Purpose:**

Format the output as Encapsulated Postscript. This adds a bounding box information to the output and makes this format suitable to be included in other postscript files.

**Example:**

```
$encoder = new PDF417Barcode($columns, $errlevel);
$backend = PDF417BackendFactory::Create(BACKEND_PS, $encoder);
$backend->SetEPS();
```

## 25.8. Example scripts

The following section shows some example scripts that can serve as starting points for further development.

## 25.8.1. Showing human readable text

### Example 25.5. Showing human readable text (`pdf417_ex3.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data = 'PDF-417';

// Setup some symbolic names for barcode specification

$columns = 8; // Use 8 data (payload) columns
$errlevel = 4; // Use error level 4
$modwidth = 2; // Setup module width (in pixels)
$height = 2; // Height factor (=2)
$showtext = false; // Show human readable string

// Create a new encoder and backend to generate PNG images
try {
 $encoder = new PDF417Barcode($columns,$errlevel);
 $backend = PDF417BackendFactory::Create(BACKEND_IMAGE,$encoder);
 $backend->SetModuleWidth($modwidth);
 $backend->SetHeight($height);
 $backend->NoText(!$showtext);
 $backend->Stroke($data);
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
?>
```

Figure 25.12. Showing human readable text (`pdf417_ex3.php`)  
[`example_src/pdf417_ex3.html`]



## 25.8.2. Altering colors

### Example 25.6. Changing colors (`pdf417_ex4.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data = 'PDF-417';

// Setup some symbolic names for barcode specification

$columns = 8; // Use 8 data (payload) columns
$errlevel = 4; // Use error level 4
$modwidth = 2; // Setup module width (in pixels)
$height = 2; // Height factor (=2)
$showtext = true; // Show human readable string

try {
 // Create a new encoder and backend to generate PNG images
 $encoder = new PDF417Barcode($columns,$errlevel);
 $backend = PDF417BackendFactory::Create(BACKEND_IMAGE,$encoder);

 $backend->SetModuleWidth($modwidth);
 $backend->SetHeight($height);
 $backend->NoText(!$showtext);
 $backend->SetColor('black','yellow');
 $backend->Stroke($data);
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
?>
```

Figure 25.13. Changing colors (`pdf417_ex4.php`) [example\_src/  
`pdf417_ex4.html`]



### 25.8.3. Creating postscript output

#### Example 25.7. (pdf417\_ex5.php)

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data = 'PDF-417';

// Setup some symbolic names for barcode specification

$columns = 8; // Use 8 data (payload) columns
$errlevel = 4; // Use error level 4
$modwidth = 0.8;// Setup module width (in PS points)
$height = 3; // Height factor (=2)
$showtext = true; // Show human readable string

try {
 // Create a new encoder and backend to generate PNG images
 $encoder = new PDF417Barcode($columns,$errlevel);
 $backend = PDF417BackendFactory::Create(BACKEND_PS,$encoder);

 $backend->SetModuleWidth($modwidth);
 $backend->SetHeight($height);
 $backend->NoText(!$showtext);
 $backend->SetColor('black','yellow');
 $output = $backend->Stroke($data);
 echo nl2br(htmlspecialchars($output));
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
?>
```

### 25.8.4. Manually selecting compaction schema

#### Purpose:

To show how to manually specify the compaction schema to be used for the input data. The example below shows how to manually specify when to use numeric and when to use text mode.

**Example 25.8. (pdf417\_ex6.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/pdf417/jpgraph_pdf417.php');

$data1 = '12345';
$data2 = 'Abcdef';
$data3 = '6789';

// Manually specify several encodation schema
$data = array(
 array(USE_NC,$data1),
 array(USE_TC,$data2),
 array(USE_NC,$data3));

// $data = "12345Abcdef6789";

// Setup some symbolic names for barcode specification

$columns = 8; // Use 8 data (payload) columns
$modwidth = 2; // Use 2 pixel module width
$errlevel = 2; // Use error level 2
$showtext = true; // Show human readable string

try {
 // Create a new encoder and backend to generate PNG images
 $encoder = new PDF417Barcode($columns,$errlevel);
 $backend = PDF417BackendFactory::Create(BACKEND_IMAGE,$encoder);

 $backend->SetModuleWidth($modwidth);
 $backend->NoText(!$showtext);
 $backend->Stroke($data);
}
catch(JpGraphException $e) {
 echo 'PDF417 Error: '.$e->GetMessage();
}
?>
```

**Figure 25.14. (pdf417\_ex6.php) [example\_src/pdf417\_ex6.html]**

# Chapter 26. Datamatrix (2D-Barcode)

## 26.1. Principle of Datamatrix Barcodes

Datamatrix (or Data Matrix) is a high density 2 dimensional barcode that can encode up to 3116 characters from the entire 256 byte ASCII character set. Compared with DF417 barcode symbology the datamatrix barcode belongs to newer family of 2 dimensional barcodes that makes better use of both dimensions and thus can achieve higher data capacity than the PDF417 symbology (~3kB vs ~2kB). The symbol is built on a square grid which have a finder pattern around the edges of the symbol to allow a scanner to identify the barcode. The finder pattern makes it possible to read the barcode regardless of the physical orientation of the code.

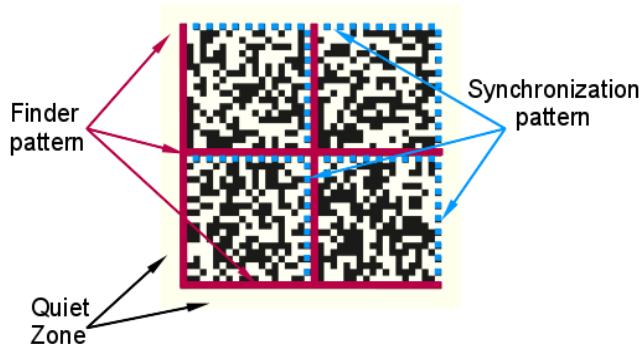
In the same way as with other 2 dimensional barcodes the datamatrix code includes error correction capability in order to be resilient towards physical damages of a code. Originally data matrix used an older convolutional error correction schema (ECC) but that has later been changed to use a Reed-Solomon type of error correction which is much more efficient. The older ECC version is known as ECC 000 to ECC 140 and should be considered obsolete and should not be used in new applications.

The newer error correction schema (with Reed-Solomon codes) is known as ECC 200 schema and is the current and recommended schema. By default the library will use the newer schema but support also exists for legacy applications to use the older ECC schema. (See ??)

Figure 26.1, “Datamatrix structure.” shows the principle of a Datamatrix barcode.

**Figure 26.1. Datamatrix structure.**

The image shows an annotated Datamatrix where the finder and synchronization patterns have been highlighted.



Even though it is primarily designed to handle the Western alphabet (ISO-8859/x code tables) it will support user prepared Unicode characters through the use of the "Extended Channel Interpretation" (ECI) mechanism. However description of the ECI standard is out of scope for this manual and the interested reader are referred to the official ECI standard document.

Datamatrix standard has been adopted by (among others) "The American National Standards Institute" (ANSI) as a standard symbology and a number of industry standard associations (e.g. EIA, SEMI, AIAG, ATA) where it has been recommended for use.

## 26.1.1. Summary of features offered in the library

The following list summarizes the features that the library offers for Datamatrix barcodes. Some of the terms used here assumes familiarity with Datamatrix barcodes. All terms are also described in the remainder of this chapter.

- Supports both the new ECC 200 variant and the older ECC 140
- Output formats
  1. Image
  2. Postscript
  3. ASCII
- Supports all recommended encodation formats
  1. ASCII
  2. C40
  3. BASE256
  4. Text
  5. X12
- Supports all specified symbol sizes
- Supports both auto and user selectable encodation
- Supports both auto and user selectable symbol size
- Supports user specified module size
- Supports custom color specification (foreground, background)
- Supports user specified quiet zone
- Supports easy handling of non-printable characters through the use of special escape sequences ("Tilde" - processing)
- Supports concatenated symbols
- Symbols can be written directly to a file or sent back as an image to the browser

## 26.1.2. Limitation of the JpGraph Datamatrix implementation

This version of the library does not support the EDIFACT compaction standard due to the very specialized and limited use of this encodation schema.

## 26.1.3. Datamatrix standard

Datamatrix as a standard is fully described in the ISO/IEC 16022E International Standard and is available for purchase from the ISO Standard Organization. [<http://www.iso.ch/iso/en/CombinedQueryResult.CombinedQueryResult?queryString=datamatrix>]

Additional information about Data Matrix code is available in the following United States patents: 4,939,354; 5,053,609; 5,124,536. See US patent Office [<http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=Search&Query=Patent+Number+4939354>] for full disclosures of these patents.

## 26.1.4. Structure of Data Matrix codes

Datamatrix is a two-dimensional symbology in the shape of a rectangle. The size and shape of the symbol is usually chosen either automatically or by the user. Usually it is chosen to be the smallest size that will have enough data capacity to encode the given data. The symbol rectangle is build up by square dots whose size "the module" is also user specified.

The Data Matrix symbol rectangle comes in two basic shapes.

1. It is either a square between the sizes of 10x10 up to 144x144 modules in even steps
2. It is a rectangle between the size of 8x16 up to 16x48

Examples of the two basic shapes are shown in Figure 26.2, “Datamatrix - Square symbol shape” and Figure 26.3, “Datamatrix - Rectangle symbol shape”

**Figure 26.2. Datamatrix - Square symbol shape** | **Figure 26.3. Datamatrix - Rectangle symbol shape**



The maximum capacity for Data Matrix codes is up to 3116 numeric characters or up to 2335 alphanumeric characters or up to 1555 bytes of binary information.

The exact number of characters that can fit in a Data Matrix symbol depends on the actual encoding (or compaction) schema used. In short this is used to more efficiently encode ASCII characters to fit more data into a fixed number of bytes. For example if only numeric data is to be encoded then instead of using one byte to hold each digit two digits is stored in a single byte hence doubling the amount of data that can be stored in a given number of bytes.

To encode data into a Datamatrix symbol the following (principal) steps are taken.

1. The input string (which can be any ASCII values between 0-255) is encoded using the selected encoding or encodings (it is possible to switch encoding mid-way through the string). The primary purpose of the encoding is to compress the data into a much shorter form.
2. If needed the data is padded to fill up to the capacity of the selected symbol size.
3. Once the string has been encoded (and possibly padded) a number of error correcting code words are added so that the data can be recovered even if part of the printed symbol have been destroyed (perhaps a corner has been torn off).
4. Finally the encoded data and the error correcting words are placed in the symbol according to an algorithm specified in the standard. This is done by placing each bit of every data byte in a specific position in the data matrix symbol.

The above explanation is by necessity simplified and for those interested into the specific details we refer to the official standard. It is also possible to review the code itself to understand the details.

## 26.1.5. Encodation efficiency

As explained in the previous section several compaction schema are used to encode the data to enable more data to fit in a given symbol. Depending on the actual data there are several compaction schema that can be used in order to achieve the greatest possible compression. The standard specifies six different schema. The compaction efficiency are given in Table 26.1, “Datamatrix encodation efficiency”.

Depending on the application the user of the library may chose to either select a fixed encodation mode but it is usually best to let the library automatically select a combination of encodation schema that will give the smallest possible symbol size.

**Table 26.1. Datamatrix encodation efficiency**

Encodation schema	Characters	Bits per character
ASCII	Double digit numerics	4
	ASCII 0-127	8
	Extended ASCII 128-255	16
C40	Primarily upper-case alphanumeric	5.33
Text	Primarily lower-case alphanumeric	5.33
X12	ANSI X12 EDI data set	5.33
EDIFACT	ASCII values 32-94	6
Base 256	All byte values 0-255	8

## 26.1.6. More on ECC Datamatrix subsets

As was mentioned in the introduction there are two main subsets of Datamatrix symbols. Those using convolutional codes for error correction which were used for most of the initial installations of Datamatrix systems, these earlier versions are referenced as ECC-000 to ECC-140 (the number specifies the level of convolutional error correcting code).

This first subset will be commonly referred to as ECC-140 in the remainder of this manual.

The second subset is referenced ECC-200 and uses Reed-Solomon error correction techniques. The two subsets have the following characteristic:

1. ECC-000 to ECC-140 symbols all have an odd number of modules along each square side.
2. ECC-200 symbols have an even number of modules on each side. ECC-200 can have non-square symbol sizes.

Hence the type of encoding used is auto-discriminative. The maximum data capacity of an ECC-200 symbol is 3116 numeric digits, or 2335 alpha numeric characters, in the largest 144 modules square symbol.

Even though the library supports the creation of both type of Datamatrix symbols it is recommended that all new applications uses the more modern ECC-200 subset. This is also the recommendation in the standard. ECC-140 should only be used in legacy system where old equipment is used which have not be upgraded to handle the modern ECC-200 subset.

## 26.1.7. Symbology Data capacity

As was mentioned in the previous section the actual data capacity depends on the symbol size. By default the library will select the smallest possible symbol size that will encode a given character string with the chosen encoding (possibly automatic). Table 2 below gives the maximum capacity for the three most common encoding schema for each symbol size as well as robustness in each symbol specified as the number of errors (destroyed data) that can be recovered.

**Table 26.2. Maximum data capacity for the different symbol sizes in ECC-200 Data Matrix subset.**

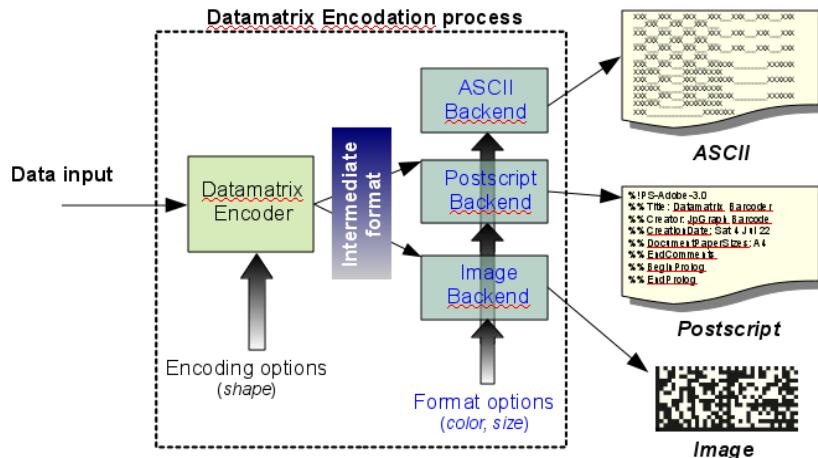
Size	Numeric capacity	Alphanumeric capacity	ca- Binary capacity	Max Error/Erasures	Correctable
10 x 10	6	3	1	2	
12 x 12	10	6	3	3	
14 x 14	16	10	6	5/7	
16 x 16	24	16	10	6/9	
18 x 18	36	25	16	7/11	
20 x 20	44	31	20	9/15	
22 x 22	60	43	28	10/17	
24 x 24	72	52	34	12/21	
26 x 26	88	64	42	14/25	
32 x 32	124	91	60	18/33	
36 x 36	172	127	84	21/39	
40 x 40	228	169	112	24/45	
44 x 44	288	214	142	28/53	
48 x 48	348	259	172	34/65	
52 x 52	408	304	202	42/78	
64 x 64	560	418	278	56/106	
72 x 72	736	550	366	72/132	
80 x 80	912	682	454	96/180	
88 x 88	1152	862	574	112/212	
96 x 96	1392	1042	694	136/260	
104 x 104	1632	1222	814	168/318	
120 x 120	2100	1573	1048	204/390	
132 x 132	2608	1954	1302	248/472	
144 x 144	3116	2335	1556	310/590	
8 x 18	10	6	3	3	
8 x 32	20	13	8	5	
12 x 26	32	22	14	7/11	
12 x 36	44	31	20	9/15	
16 x 36	64	46	30	12/21	
16 x 48	98	72	47	14/25	

## 26.2. Creating barcodes

In order to use datamatrix barcodes the module "datamatrix/datamatrix.inc.php" must first be included.

Usage of Datamatrix barcodes follows a similar schema as for the linear and PDF417 barcodes with concepts of an encoder and backend. The principle of the overall encodation process is shown in Figure 26.4, "Datamatrix encodation principle"

**Figure 26.4. Datamatrix encodation principle**



### 26.2.1. Getting started

Assuming that the library is installed where it can be found by PHP it is now very simple to create a basic Data Matrix symbol using just default values. We first show the complete first example and then discuss it. In the following scripts we will assume that the include path has the core JpGraph directory in its path.

All barcode creation follows the following basic three steps :

1. Create an instance of the encoder with the chosen datamatrix layout as an instance of class `Data-matrix`
2. Create an instance of a suitable backend for the chosen output format (image or postscript) by calling the `DatamatrixBackendFactory::Create()`
3. Encode data and send it back to the browser or to a file with a call to the backend `Backend::Stroke()` method.

The following script shows how to create the simplest possible barcode (in PNG format) representing the data string "The first datamatrix" encoded using all default values. The resulting barcode is shown in Figure 26.5, "The simplest possible datamatrix script (datamatrix\_ex00.php)"

**Example 26.1. The simplest possible datamatrix script (`datamatrix_ex00.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/datamatrix/datamatrix.inc.php');

$data = 'The first datamatrix';

$encoder = DatamatrixFactory::Create();
$backend = DatamatrixBackendFactory::Create($encoder);
$backend->Stroke($data);
?>
```

**Figure 26.5. The simplest possible datamatrix script (`datamatrix_ex00.php`) [[example\\_src/datamatrix\\_ex00.html](#)]**



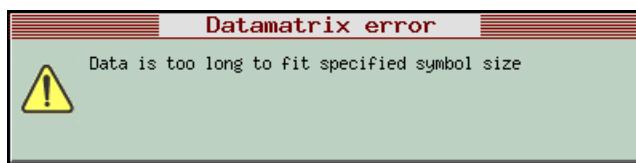
(As can be seen this follows the same principles as the creation of PDF417 symbols)

The principle is the same for all type of data matrix symbol creations. We first create an instance of the chosen encoder. In this case a standard ECC-200 encoder by creating an instance of class `Datamatrix`. We then create a suitable backend that handles the output of the barcode. By default the output will be an image encoded in the PNG image format.

The final step is to send back the generated image to the browser with a call to the method `Backend::Stroke()` with the data to be encoded as its first argument.

The example above does not have any error handling. If there is some error in the process an exception will be thrown in the same way as in other places in the library. The default exception will display a standard library image error box. An example of this is shown in Figure 26.6, “Datamatrix error image”.

**Figure 26.6. Datamatrix error image**



If some additional processing is necessary and just display a text based re-formatted error message we could change the above code to catch this exception as the following example shows.

**Example 26.2. Datamatrix with basic error handling (`datamatrix_ex0.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/datamatrix/datamatrix.inc.php');

$data = 'The first datamatrix';
$encoder = DatamatrixFactory::Create();
$encoder->SetEncoding(ENCODING_ASCII);
$backend = DatamatrixBackendFactory::Create($encoder);

// We increase the module width to 3 pixels
$backend->SetModuleWidth(3);

try {
 $backend->Stroke($data);
} catch (Exception $e) {
 echo 'Datamatrix error: '.$e->GetMessage()."\n";
 exit(1);
}
?>
```

**Figure 26.7. Datamatrix with basic error handling (`datamatrix_ex0.php`)**  
[\[example\\_src/datamatrix\\_ex0.html\]](#)



In contrast to the PDF417 encodation process there is no option to select an level of error correction. The error correction level is automatically specified in the standard depending on the size of the barcode. As a final initial example the next script uses the backend method `Backend::SetModuleWidth($aWidth)` to increase the width of one module.

**Example 26.3. Datamatrix with modified module width (`datamatrix_ex1.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/datamatrix/datamatrix.inc.php');

$data = '123456';

$encoder = DatamatrixFactory::Create();
$backend = DatamatrixBackendFactory::Create($encoder);
$backend->SetModuleWidth(3);

// Create the barcode from the given data string and write to output file
try {
 $backend->Stroke($data);
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 echo "Datamatrix error message: $errstr\n";
}

?>
```

**Figure 26.8. Datamatrix with modified module width (`datamatrix_ex1.php`)**  
`[example_src/datamatrix_ex1.html]`



## 26.2.2. Error handling

As in other parts of the library the Datamatrix module throws an exception when an error occurs. As shown in the previous section an error image is the default if no explicit `try {} catch {}` statement is added to the script.

The script below shows how to catch the error, do some possible clean up and then explicitly create and send back the error image.

```
<?php
...
try {
 $backend->Stroke($data);
} catch(Exception $e) {
 doCleanup();
 $errobj = new DMErrorObjectImg();
 $errobj->Raise($e->getMessage());
}
?>
```

another variant of this would be to re-throw the exception after the cleanup has been performed as the following example shows

```
<?php
...
try {
 $backend->Stroke($data);
} catch(Exception $e) {
 doCleanup();
 throw $e;
}
?>
```

It is also possible to get hold of the internal error code that corresponds to each error message by calling the PHP standard exception method `Exception::getCode()` as the following example shows

```
<?php
...
try {
 $backend->Stroke($data);
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 $errcode = $e->GetCode();
 echo "Datamatrix error ($errcode). Message: $errstr\n";
}
?>
```

Table 26.3, “Datamatrix error messages” lists the errors that can be thrown by the datamatrix module (the table deliberately excludes internal error messages).

**Table 26.3. Datamatrix error messages**

Error number	Error messages
-1	Data is too long to fit specified symbol size
-2	The BASE256 data is too long to fit available symbol size
-3	Data must have at least three characters for C40 encodation
-4	Data must have at least three characters for TEXT encodation
-8	The given data can not be encoded using X12 encodation.
-9	The "tilde" encoded data is not valid.
-10	Data must have at least three characters for X12 encodation
-11	Specified data can not be encoded with datamatrix 000 140
-12	Can not create image
-13	Invalid color specification
-15	This PHP installation does not support the chosen image encoding format
-20	The specification for shape of matrix is out of bounds (0,29)
-21	Cannot open the data file specifying bit placement for Datamatrix 200
-22	Datafile for bit placement is corrupt, crc checks fails.
-26	Cannot open the data file specifying bit placement for Datamatrix 140
-30	The symbol size specified for ECC140 type Datamatrix is not valid
-31	Data is too long to fit into any available matrix size for datamatrix 140
-34	Cannot open file %s for writing.
-35	Cannot write to file %s .
-99	EDIFACT encodation not implemented
-100	Datamatrix Error: HTTP headers have already been sent.  Caused by output from file %s at line %d  Explanation: HTTP headers have already been sent back to the browser indicating the data as text before the library got a chance to send it's image HTTP header to this browser. This makes it impossible for the Datamatrix library to send back image data to the

Error number	Error messages
	<p>browser (since that would be interpreted as text by the browser and show up as junk text).</p> <p>Most likely you have some text in your script before the call to DatamatrixBackend::Stroke().</p> <p>If this texts gets sent back to the browser the browser will assume that all data is plain text. Look for any text (even spaces and newlines) that might have been sent back to the browser.</p> <p>For example it is a common mistake to leave a blank line before the opening "&lt;?php"</p>

### 26.2.3. Encodation options

The primary encodation option is to manually specify the shape of the Datamatrix barcode. By default the symbol size will be chosen as the smallest possible. However some application require that usage of a fixed size symbol shape.

The available shapes are listed in Table 26.2, “Maximum data capacity for the different symbol sizes in ECC-200 Data Matrix subset.”. The wanted shape is specified when the instance of the encoder is created or by a call to `Datamatrix::SetSize()`. The shape is specified by a symbolic constant that corresponds to each available shape in Table 26.2, “Maximum data capacity for the different symbol sizes in ECC-200 Data Matrix subset.”. The symbol name is “DMAT\_<H>x<W>” where <H> is the height (in modules) and <W> is the width (in modules). For example the following line specifies a 24 module square symbol

```
$encoder->SetSize(DMAT_24x24);
```

In addition to the sizes specified in Table 26.2, “Maximum data capacity for the different symbol sizes in ECC-200 Data Matrix subset.” there is also an option to specify automatic sizing (which is the default). The following script sets the sizing to be automatic.

```
$encoder->SetSize(DMAT_AUTO);
```

As was discussed in Section 26.1.5, “Encodation efficiency” there are several ways by which data can be encoded. By default the encoder will use an optimal algorithm to create a combination of the available encodation schema to create the smallest possible symbol. In some application it might however be stated that a specific encodation method should be chosen.

The encoding method is a property of the encodation class and is set with a call to

- `Datamatrix::SetEncoding($aEncoding=ENCODING_ASCII)`

The encoding is specified as one of the symbolic constants shown in Table 26.4, “Datamatrix encodation schemas”

**Table 26.4. Datamatrix encodation schemas**

Symbolic constant	Encoding schema	Character set
ENCODING_AUTO	--	An optimal combination of all encodation schema
ENCODING_ASCII	ASCII	Most efficient for numeric data

Symbolic constant	Encoding schema	Character set
ENCODING_C40	C40	Primarily upper-case alphanumeric
ENCODING_TEXT	Text	Primarily lower-case alphanumeric
ENCODING_X12	X12	ANSI X12 EDI data set
ENCODING_EDIFACT	EDIFACT	ASCII values 32-94
ENCODING_BASE256	Base 256	All byte values 0-255

The following example sets the ASCII encodation schema

```
$encoder->SetEncoding(ENCODING_ASCII);
```

The default automatic encoding is the same as

```
$encoder->SetSize(ENCODING_AUTO);
```

## 26.2.4. Processing special input characters

In order to allow the specification of special input data in normal ASCII encodation for easy entering from a keyboard the library supports what is commonly known as "tilde" processing. When this is enabled (it is disabled by default) certain character sequences that begins with a tilde sign (i.e. "~") will be translated to a special ASCII value that cannot normally be entered from a keyboard.

When tilde mode is enabled with a call to the method

- `Datamatrix::SetTilde($aFlg=true)`

Table 26.5, “Tilde processing translating” lists the translations that will then be enabled on the input data. Some of the translations are technical and requires full understanding of Datamatrix standard to be used.

**Table 26.5. Tilde processing translating**

Input sequence	Translation
~X : where 'X' is a character in the range [@-Z]	Translates to ASCII value 0-25, i.e. ~@==0, ~A==1, ~B==2, ...
~1	Represents the character FNC1. Alternate Data Type Identifier. See Table 6: ISO 16022
~2nnmmffffggg	<p>Structured Append. This code is only allowed in the first position of the data. Up to 16 ECC 200 symbols may be appended in a structured format. The structured append is specified with symbol index, total number of symbols and a file identification.</p> <p>The first two values "nn" is the position of this particular symbol specified as two digits in the range of 01-16.</p> <p>The second two values "mm" is the total number of symbols in this structured append in the range 02-16 specified as two digits.</p>

Input sequence	Translation
	The last two 3 digit codewords are the file identification specified with digits and necessary leading 0:s to make it a full three digits. For example. Assume that we want to encode the 4:th symbol from a total of 9 symbols and with the file identification "0,1". This would then be encoded as "~20409000001". Note: Internally this is translated to three codewords with the reserved starting codeword of ASCII 233.
~5 and ~6	<p>The special 05 and 06 Macros. Can only be in the first position and is used to encode industry standard headers in certain structured formats.</p> <p>Macro 05 is translated by the barcode reader to : Symbol prefix: chr(30) chr(05) chr(29) Symbol postfix: chr(30) chr(04) Macro 06 is translated by the barcode reader to : Symbol prefix: chr(30) chr(06) chr(29) Symbol postfix: chr(30) chr(04)</p>
~7nnnnnn	<p>Extended Channel Interpretation (ECI) nnnnnn.</p> <p>The ECI protocol allows the output data stream to have interpretations different from that of the default character set. See "<i>Extended Channel Interpretation Assignments</i>" document (available from ISO) for a list of channels and there meaning.</p> <p>The ECI is identified by a 6-digit number which is encoded according to Table 10: in the ISO 16022:2000 specification. This is the encodation that is used for the "~7" tilde specification of the ECI protocol.</p>
~9	Send the special control code (ASCII=234) so that the reader will interpret the rest of the symbol as a bar code reader programming instruction.
~dnnn	Character value as 3 digits, i.e. ~d142 means ASCII value 142

## 26.2.5. Creating different backends

In order to create the actual output one or more backends must be created. The Datamatrix supports the following backends:

- BACKEND\_IMAGE, Create an image backend. This is the default backend if no explicit backend is specified.
- BACKEND\_PS, Create a postscript backend. The text string that represents the postscript for the barcode is returned directly from the Backend::Stroke()
- BACKEND\_EPS, Create an encapsulated postscript backend
- BACKEND\_ASCII, This is a special backend which will generate an ASCII rendering of the datamatrix barcode. This is mostly practical for technical investigations regarding the technical structure of a Datamatrix barcode.

The following code snippet shows two ways of creating a barcode image backend.

```
// Create an image backend
$backend = DatamatrixBackendFactory::Create($encoder);

// Create an image backend
$backend = DatamatrixBackendFactory::Create($encoder, BACKEND_IMAGE);
```

The following code shows a complete script to generate a postscript output

```
<?php
require_once('jpgraph/datamatrix/datamatrix.inc.php');
$data = '0123456789';

// Create and set parameters for the encoder
$encoder = DatamatrixFactory::Create();

// Create the image backend (default)
$backend = DatamatrixBackendFactory::Create($encoder, BACKEND_PS);
try {
 $ps_txt = $backend->Stroke($data);
 echo '<pre>' . $ps_txt . '</pre>';
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 echo "Datamatrix error message: $errstr\n";
}
?>
```

## 26.2.6. Generic backend methods

The following methods are available to adjust the final look and feel of the barcode

1. Backend::SetColor(\$aOne, \$aZero, \$aBackground='white')

the color of the 'black' and 'white' modules in the barcode and the quiet area around the barcode

2. Backend::SetQuietZone(\$aSize)

the size of the "quiet zone" in pixels for the image backend and in points (1 pt = 1/72 inch) for the postscript backend

3. Backend::SetModuleWidth(\$aWidth)

the module width specified in pixels for the image backend and in points (1 pt = 1/72 inch) for the postscript backend

4. Backend::Stroke(\$aData, \$aFileName= '')

create the barcode and send it back the client or store it to a file if the second parameter is set. For postscript backends the postscript string is returned directly from the method. (See example above)

## 26.2.7. Image backend methods

For the image backend it is possible to adjust the image encoding format with the following method

### 1. Backend::SetImgFormat (\$aFormat, \$aQuality=75)

Specify image format. Possible values are

- 'auto'
- 'png'
- 'jpeg'
- 'gif'

For 'jpeg' format the quality parameter is a number in range [1-100] and specifies how much compression / (Data loss) should be used. 100=no compression. Normal values are in the range [60-95]

The following example sets the image format to 'JPEG' with quality 80.

```
$backend->SetImgFormat('jpeg', 80);
```

## 26.2.8. Postscript backend format options

For postscript backend it is possible to select whether the postscript should be generated as encapsulated postscript. This is controlled by the method

- Backend::SetEPS(\$aFlg=true)

## 26.2.9. A template to create barcodes

In the example directory in the distribution ('datamatrix/examples') you can find many more examples on how to create barcodes. As a good start the following (simple) template may be used as a base for further customizations.

```
<?php
require_once('jpgraph/datamatrix/datamatrix.inc.php');

$data = '0123456789';

$shape = DMAT_AUTO;
$encoding = ENCODING_AUTO;
$modulewidth = 3;
$quietzone = 10;
$cicolor1 = 'black';
$cicolor0 = 'white';
$cicolorq = 'white';
$outputfile = '';

// Create and set parameters for the encoder
$encoder = DatamatrixFactory::Create($shape);
$encoder->SetEncoding($encoding);

// Create the image backend (default)
$backend = DatamatrixBackendFactory::Create($encoder);

// By default the module width is 2 pixel so we increase it a bit
$backend->SetModuleWidth($modulewidth);
```

```

// Set Quiet zone
$backend->SetQuietZone($quietzone);

// Set other than default colors (one, zero, quiet zone/background)
$backend->SetColor($color1, $color0, $colorq);

// Create the barcode from the given data string and write to output file
try {
 $backend->Stroke($data,$outputfile);
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 echo "Datamatrix error message: $errstr\n";
}
?>

```

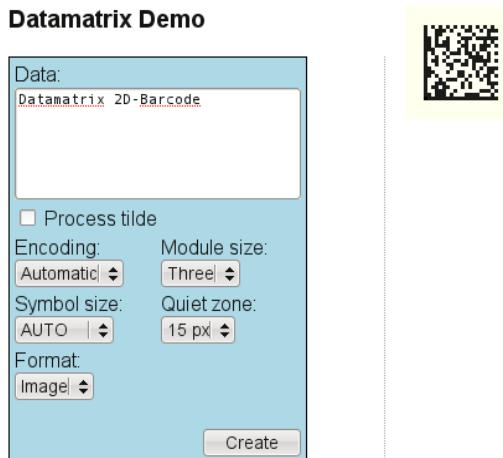
## 26.2.10. Sample application

As an example the library includes a WEB-based demo barcode creation application. This application can be used to easily create barcode through it's WEB interface. It is available at 'datamatrix/de-moapp/index.html'

This application is primarily included as a demo on the features available in the library and not as a finalized product.

Figure 26.9, “Datamatrix WEB-based demo application” shows a screen shot of the application interface.

**Figure 26.9. Datamatrix WEB-based demo application**



## 26.3. Example script

### 26.3.1. Example 1 - Setting the shape

The following example shows how to

- Set the shape to 64x64

- Use ASCII encoding
- Adjust the quiet zone

**Example 26.4. Datamatrix example, setting quiet zone and ASCII encoding  
(datamatrix\_ex4.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/datamatrix/datamatrix.inc.php');

$data = 'This is a 64x64 datamatrix symbol';

// Create and set parameters for the encoder
$encoder = DatamatrixFactory::Create(DMAT_64x64);
$encoder->SetEncoding(ENCODING_TEXT);

// Create the image backend (default)
$backend = DatamatrixBackendFactory::Create($encoder);
$backend->SetModuleWidth(3);

// Adjust the Quiet zone
$backend->SetQuietZone(10);

// Create the barcode from the given data string and write to output file
try {
 $backend->Stroke($data);
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 echo "Datamatrix error message: $errstr\n";
}

?>
```

**Figure 26.10. Datamatrix example, setting quiet zone and ASCII encoding  
(datamatrix\_ex4.php) [example\_src/datamatrix\_ex4.html]**



### 26.3.2. Example 2 - Writing to a file

The following example shows how to generate a barcode and write it to a file.

### Example 26.5. Datamatrix example, writing to a file (`datamatrix_ex6.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/datamatrix/datamatrix.inc.php');

$data = 'This is a datamatrix symbol';

$outputfile = 'dm_ex6.png';

// Create and set parameters for the encoder
$encoder = DatamatrixFactory::Create();
$encoder->SetEncoding(ENCODING_TEXT);

// Create the image backend (default)
$backend = DatamatrixBackendFactory::Create($encoder);
$backend->SetModuleWidth(5);
$backend->SetQuietZone(10);

// Set other than default colors (one, zero, background)
$backend->SetColor('navy','white');

// Create the barcode from the given data string and write to output file
$dir = dirname(__FILE__);
$file = '' . $dir . '/' . $outputfile . '';
try {
 $backend->Stroke($data,$outputfile);
 echo 'Barcode sucessfully written to file: ' . $file;
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 $errcode = $e->GetCode();
 echo "Failed writing file: " . $file . '
';
 echo "Datamatrix error ($errcode). Message: ' . $errstr . '\n";
}
?>
```

### 26.3.3. Example 3 - Creating postscript output

The following example shows how to generate a postscript output.

When

**Example 26.6. Datamatrix example, creating postscript output  
(datamatrix\_ex7.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/datamatrix/datamatrix.inc.php');

$data = 'A Datamatrix barcode';

// Create and set parameters for the encoder
$encoder = DatamatrixFactory::Create();
$encoder->SetEncoding(ENCODING_BASE256);

// Create the image backend (default)
$backend = DatamatrixBackendFactory::Create($encoder, BACKEND_PS);
$backend->SetModuleWidth(3);

try {
 $ps_txt = $backend->Stroke($data);
 echo '<pre>' . $ps_txt . '</pre>';
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 echo "Datamatrix error message: $errstr\n";
}

?>
```

The resulting output is shown in Figure 26.11, “Datamatrix postscript output” below

**Figure 26.11. Datamatrix postscript output**

```
%!PS-Adobe-3.0
%%Title: Datamatrix Barcode
%%Creator: JpGraph Barcode http://www.aditus.nu/jpgraph/
%%CreationDate: Sun 5 Jul 23:06:27 2009
%%DocumentPaperSizes: A4
%%EndComments
%%BeginProlog
%%EndProlog
%%Page: 1 1
%Module width: 3 pt

%Data for bars. Only black bars are defined.
%The figures are for each row and in format: [xpos]
%Data: A Datamatrix barcode
3.05 setlinewidth
[0] [6] [12] [18] [24] [30] [36] [42] [48] [54] {{}} forall 60 moveto 0 -3.05 rlineto stroke} forall
[0] [6] [12] [18] [21] [33] [36] [39] [42] [51] [54] [57] {{}} forall 57 moveto 0 -3.05 rlineto stroke} forall
[0] [18] [21] [24] [36] [42] [51] {{}} forall 54 moveto 0 -3.05 rlineto stroke} forall
[0] [12] [15] [24] [30] [36] [39] [42] [45] [48] [51] [54] [57] {{}} forall 51 moveto 0 -3.05 rlineto stroke} forall
[0] [6] [9] [21] [24] [33] [36] [51] [54] {{}} forall 48 moveto 0 -3.05 rlineto stroke} forall
[0] [3] [6] [15] [21] [24] [33] [48] [51] [57] {{}} forall 45 moveto 0 -3.05 rlineto stroke} forall
[0] [9] [12] [15] [24] [27] [39] [51] [54] {{}} forall 42 moveto 0 -3.05 rlineto stroke} forall
[0] [6] [18] [24] [27] [33] [39] [42] [48] [54] [57] {{}} forall 39 moveto 0 -3.05 rlineto stroke} forall
[0] [9] [12] [15] [21] [24] [27] [30] [33] [39] [45] [54] {{}} forall 36 moveto 0 -3.05 rlineto stroke} forall
[0] [3] [15] [18] [21] [24] [30] [33] [36] [39] [48] [57] {{}} forall 33 moveto 0 -3.05 rlineto stroke} forall
[0] [3] [6] [18] [21] [27] [39] [45] {{}} forall 30 moveto 0 -3.05 rlineto stroke} forall
[0] [15] [18] [21] [27] [30] [33] [36] [42] [51] [54] [57] {{}} forall 27 moveto 0 -3.05 rlineto stroke} forall
[0] [3] [6] [12] [15] [18] [21] [30] [33] [36] [39] [42] [48] {{}} forall 24 moveto 0 -3.05 rlineto stroke} forall
[0] [6] [12] [27] [30] [42] [57] {{}} forall 21 moveto 0 -3.05 rlineto stroke} forall
[0] [3] [9] [15] [18] [24] [33] [39] [42] [45] [48] {{}} forall 18 moveto 0 -3.05 rlineto stroke} forall
[0] [9] [12] [21] [27] [30] [36] [39] [42] [45] [57] {{}} forall 15 moveto 0 -3.05 rlineto stroke} forall
[0] [9] [21] [27] [36] [39] [42] [45] [51] [54] {{}} forall 12 moveto 0 -3.05 rlineto stroke} forall
[0] [9] [21] [27] [36] [39] [42] [45] [51] [57] {{}} forall 9 moveto 0 -3.05 rlineto stroke} forall
[0] [15] [21] [24] [27] [36] [39] [42] [45] [54] {{}} forall 6 moveto 0 -3.05 rlineto stroke} forall
[0] [3] [6] [9] [12] [15] [18] [21] [24] [27] [30] [33] [36] [39] [42] [45] [48] [51] [54] [57] {{}} forall 3 moveto 0 -3.05 rlineto stroke} forall

%End of Datamatrix Barcode
showpage
%%Trailer
```

### 26.3.4. Example 4 - Changing background color

The following example shows how to modify the colors in the barcode.

**Example 26.7. Datamatrix example, changing colors (`datamatrix_ex5.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/datamatrix/datamatrix.inc.php');

$data = 'This is a 20x20 symbol';

// Create and set parameters for the encoder
$encoder = DatamatrixFactory::Create(DMAT_20x20);
$encoder->SetEncoding(ENCODING_TEXT);

// Create the image backend (default)
$backend = DatamatrixBackendFactory::Create($encoder);

// By default the module width is 2 pixel so we increase it a bit
$backend->SetModuleWidth(4);

// Set Quiet zone
$backend->SetQuietZone(10);

// Set other than default colors (one, zero, quiet zone/background)
$backend->SetColor('navy','white','lightgray');

// Create the barcode from the given data string and write to output file
try {
 $backend->Stroke($data);
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 echo "Datamatrix error message: $errstr\n";
}

?>
```

**Figure 26.12. Datamatrix example, changing colors (`datamatrix_ex5.php`)**  
[`example_src/datamatrix_ex5.html`]



---

# Chapter 27. QR (2D-Barcode)

## 27.1. Principle of QR Barcodes

### Note

This module is only available in the pro-version of the library.

QR is currently the highest capacity general two-dimensional matrix symbology available (up to ~7Kb numeric data can be encoded) and it is designed to encode the full 256 byte ASCII character set as well as the Kanji (Shift-JIS character set). QR code belongs to the modern 2 dimensional codes that is designed for both high capacity as well as to be efficient for scanner equipment to process and this is also the reason for its name - Quick Response code.

An advantage with QR code is also there relatively small size for a given amount of information

The QR code is available in 40 different square sizes each with a user selectable error correction level in four steps (referred to as error correction level L,M,Q and H). With the highest level of error correction used up to ~30% of the codewords can be damaged and still be restored.

QR code is extensively used in some Asian countries and is finding more and more usage to transfer medium sized information onto mobile phones where the QR codes are interpreted by first taking a photo of the barcode with the mobile and then running a QR decoding program on the cell phone. This has currently become the most expansive usage of QR codes and some mobile manufacturers are providing software to aid in interpreting QR codes for free.

### Note

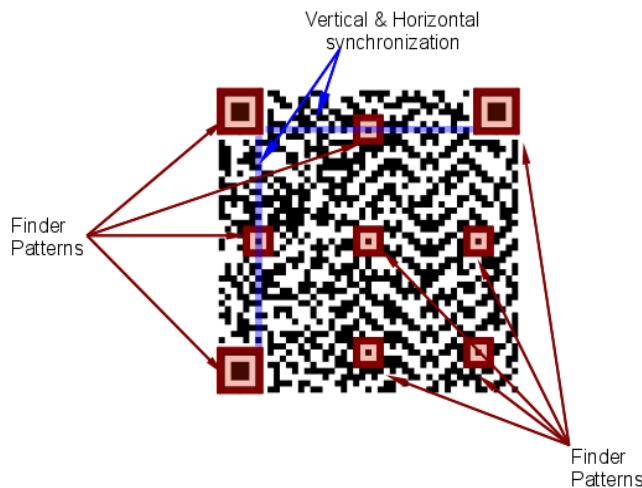
QR codes are also finding their way into public information sharing. For example in Texas in US some cities are using QR codes to display public information.

### Note

QR Codes can be freely used without royalties. QR Codes are copyrighted by:

DENSO WAVE INCORPORATED [<http://www.denso-wave.com>].

In Figure 27.1, “QR Code high level structure” we have amended a large version of a basic QR code. In this figure the special finding patterns and timing patterns have been highlighted in red and blue.

**Figure 27.1. QR Code high level structure**

The images below shows some real-life creative use of QR barcodes.

**Figure 27.2. Creative usage of QR Barcodes**

### 27.1.1. Summary of library features

- Output format
  1. Image format
  2. Postscript and encapsulated postscript
  3. ASCII
- Encodation formats

1. Numeric
  2. Alphanumeric
  3. Binary
- Supports all specified symbol sizes
  - Supports both auto and user selectable encodation
  - Supports both auto and user selectable symbol size
  - Supports user specified module size
  - Supports custom color specification (foreground, background)
  - Symbols can be written directly to a file or sent back as an image to the browser

## 27.1.2. Limitation in the JpGraph QR implementation

This implementation is a faithful implementation of the ISO/IEC 18004E International Standard with one important exception. Version 1.0 does not support the encoding of 2 byte Shift-JIS character set. This means that two byte Kanji can only be encoded using binary data format which is less efficient than the native Kanji encoding.

### Caution

The computational complexity of the encodation process makes QR code generation slightly slower than both PDF417 and Datamatrix. The technical reason for most of the time is the final necessary evaluation of a bitmap mask which is chosen to give as close to 50% mix of black and white areas in the code as possible. Unfortunately this evaluation take time proportional to the square of the size of the matrix. This time can be noticeable when using "large" symbol sizes. There are existing QR code libraries that cheat on this final step and just randomly selects a bit mask to avoid this computational intensive evaluation. This is also the cause for the visual difference between this libraries final result and some other.

In this case we claim that this library implements the proper standard much more faithfully than some other libraries that exists.

## 27.1.3. QR Standard

The QR code standard is fully described in the ISO/IEC 18004E International Standard and is available for purchase from the ISO Standard Organization [<http://www.iso.ch/iso/en/CombinedQueryResult.CombinedQueryResult?queryString=qr>]. for a nominal fee.

## 27.1.4. Structure and capacity of QR barcodes

All QR barcode have square layout made up of equal spaced square modules. Depending on the size of the QR code a certain number of finder patterns are included in the code to aid in scanner decoding. The standard specifies 40 versions (sizes) of the QR code from the smallest 21x21 up to 177x177 modules in size.

Examples of two basic QR codes are shown in Figure 27.3, “A small sized QR Code (version=2) ” and Figure 27.4, “A medium sized QR code (version=13)”

**Figure 27.3. A small sized QR Code (version=2)**



**Figure 27.4. A medium sized QR code (version=13)**



Depending on the actual data there are several compaction schema that can be used in order to achieve the greatest possible compression. The standard specifies four different principal schema, Numeric, Alphanumeric, Binary and Kanji.

Depending on the application the user of the library may chose to either select a fixed encodation mode or let the library automatically chose the most efficient encodation method. It is usually best to let the library automatically select a combination of encodation schema that will give the smallest possible symbol size.

The maximum capacity for QR codes dependent on the encodation schema (using the lowest possible error correction overhead) are given in Table 27.1, “QR Data capacity”

**Table 27.1. QR Data capacity**

Encoding mode	Maximum capacity
Numeric	7089 digits (Datamatrix=3116)
Alphanumeric	4296 characters (Datamatrix=2335)
Binary	2953 (Datamatrix=1555)
Kanji	1817 (Datamatrix NA)

The exact number of characters that can fit in a QR symbol depends on the actual encoding (or compaction) schema used. In short this is used to more efficiently encode ASCII characters to fit more data into a fixed number of bytes. For example if only numeric data is to be encoded then instead of using one byte to hold each digit three digits is stored in 10 bits. Which gives the equivalent capacity that 12 digits takes only 5 bytes to encode.

To encode data into a QR symbol the following principal steps are taken.

1. The input string (which can be any ASCII values between 0-255) is encoded using the selected encoding or encodings (it is possible to switch encoding mid-way through the string). The primary purpose of the encoding is to compress the data into a much shorter form.
2. If needed the data is padded to fill up to the capacity of the selected symbol size.
3. Once the string has been encoded (and possibly padded) a number of error correcting code words are added so that the data can be recovered even if part of the printed symbol have been destroyed (perhaps a corner has been torn off).

4. Finally the encoded data and the error correcting words are placed in the symbol according to an algorithm specified in the standard. This is done by placing each bit of every data byte in a specific position in the qr matrix symbol.

The above explanation is by necessity simplified and for those interested into the specific details we refer to the official standard. It is also possible to review the code itself to understand the details.

## 27.1.5. QR versions and symbol size

As mentioned in the previous section the QR standard specifies 40 different sizes of the QR code and the maximum data capacity will also vary depending on the size. Table 27.2, “Maximum data capacity for the different symbol sizes in the QR-code.” shows the defined sizes and for each size specifies the possible Error correction levels and the maximum data capacity depending on the compaction schema used.

By default the symbol size will be chosen as the smallest possible. However some application require that usage of a fixed size symbol. The symbol size is a parameter of the encodation schema and is adjusted in the creation of the `QREncoder`.

**Table 27.2. Maximum data capacity for the different symbol sizes in the QR-code.**

Version	Modules	ECC Level	Data bits	Numeric	Alphanumeric
1	21x21	L	152	41	25
M	128	34	20	14	8
Q	104	27	16	11	7
H	72	17	10	7	4
2	25x25	L	272	77	47
M	224	63	38	26	16
Q	176	48	29	20	12
H	128	34	20	14	8
3	29x29	L	440	127	77
M	352	101	61	42	26
Q	272	77	47	32	20
H	208	58	35	24	15
4	33x33	L	640	187	114
M	512	149	90	62	38
Q	384	111	67	46	28
H	288	82	50	34	21
5	37x37	L	864	255	154
M	688	202	122	84	52
Q	496	144	87	60	37
H	368	106	64	44	27
6	41x41	L	1,088	322	195
M	864	255	154	106	65
Q	608	178	108	74	45
H	480	139	84	58	36
7	45x45	L	1,248	370	224

<b>Version</b>	<b>Modules</b>	<b>ECC Level</b>	<b>Data bits</b>	<b>Numeric</b>	<b>Alphanumeric</b>
M	992	293	178	122	75
Q	704	207	125	86	53
H	528	154	93	64	39
8	49x49	L	1,552	461	279
M	1,232	365	221	152	93
Q	880	259	157	108	66
H	688	202	122	84	52
9	53x53	L	1,856	552	335
M	1,456	432	262	180	111
Q	1,056	312	189	130	80
H	800	235	143	98	60
10	57x57	L	2,192	652	395
M	1,728	513	311	213	131
Q	1,232	364	221	151	93
H	976	288	174	119	74
11	61x61	L	2,592	772	468
M	2,032	604	366	251	155
Q	1,440	427	259	177	109
H	1,120	331	200	137	85
12	65x65	L	2,960	883	535
M	2,320	691	419	287	177
Q	1,648	489	296	203	125
H	1,264	374	227	155	96
13	69x69	L	3,424	1,022	619
M	2,672	796	483	331	204
Q	1,952	580	352	241	149
H	1,440	427	259	177	109
14	73x73	L	3,688	1,101	667
M	2,920	871	528	362	223
Q	2,088	621	376	258	159
H	1,576	468	283	194	120
15	77x77	L	4,184	1,250	758
M	3,320	991	600	412	254
Q	2,360	703	426	292	180
H	1,784	530	321	220	136
16	81x81	L	4,712	1,408	854
M	3,624	1,082	656	450	277
Q	2,600	775	470	322	198
H	2,024	602	365	250	154

<b>Version</b>	<b>Modules</b>	<b>ECC Level</b>	<b>Data bits</b>	<b>Numeric</b>	<b>Alphanumeric</b>
17	85x85	L	5,176	1,548	938
M	4,056	1,212	734	504	310
Q	2,936	876	531	364	224
H	2,264	674	408	280	173
18	89x89	L	5,768	1,725	1,046
M	4,504	1,346	816	560	345
Q	3,176	948	574	394	243
H	2,504	746	452	310	191
19	93x93	L	6,360	1,903	1,153
M	5,016	1,500	909	624	384
Q	3,560	1,063	644	442	272
H	2,728	813	493	338	208
20	97x97	L	6,888	2,061	1,249
M	5,352	1,600	970	666	410
Q	3,880	1,159	702	482	297
H	3,080	919	557	382	235
21	101x101	L	7,456	2,232	1,352
M	5,712	1,708	1,035	711	438
Q	4,096	1,224	742	509	314
H	3,248	969	587	403	248
22	105x105	L	8,048	2,409	1,460
M	6,256	1,872	1,134	779	480
Q	4,544	1,358	823	565	348
H	3,536	1,056	640	439	270
23	109x109	L	8,752	2,620	1,588
M	6,880	2,059	1,248	857	528
Q	4,912	1,468	890	611	376
H	3,712	1,108	672	461	284
24	113x113	L	9,392	2,812	1,704
M	7,312	2,188	1,326	911	561
Q	5,312	1,588	963	661	407
H	4,112	1,228	744	511	315
25	117x117	L	10,208	3,057	1,853
M	8,000	2,395	1,451	997	614
Q	5,744	1,718	1,041	715	440
H	4,304	1,286	779	535	330
26	121x121	L	10,960	3,283	1,990
M	8,496	2,544	1,542	1,059	652
Q	6,032	1,804	1,094	751	462

<b>Version</b>	<b>Modules</b>	<b>ECC Level</b>	<b>Data bits</b>	<b>Numeric</b>	<b>Alphanumeric</b>
H	4,768	1,425	864	593	365
27	125x125	L	11,744	3,514	2,132
M	9,024	2,701	1,637	1,125	692
Q	6,464	1,933	1,172	805	496
H	5,024	1,501	910	625	385
28	129x129	L	12,248	3,669	2,223
M	9,544	2,857	1,732	1,190	732
Q	6,968	2,085	1,263	868	534
H	5,288	1,581	958	658	405
29	133x133	L	13,048	3,909	2,369
M	10,136	3,035	1,839	1,264	778
Q	7,288	2,181	1,322	908	559
H	5,608	1,677	1,016	698	430
30	137x137	L	13,880	4,158	2,520
M	10,984	3,289	1,994	1,370	843
Q	7,880	2,358	1,429	982	604
H	5,960	1,782	1,080	742	457
31	141x141	L	14,744	4,417	2,677
M	11,640	3,486	2,113	1,452	894
Q	8,264	2,473	1,499	1,030	634
H	6,344	1,897	1,150	790	486
32	145x145	L	15,640	4,686	2,840
M	12,328	3,693	2,238	1,538	947
Q	8,920	2,670	1,618	1,112	684
H	6,760	2,022	1,226	842	518
33	149x149	L	16,568	4,965	3,009
M	13,048	3,909	2,369	1,628	1,002
Q	9,368	2,805	1,700	1,168	719
H	7,208	2,157	1,307	898	553
34	153x153	L	17,528	5,253	3,183
M	13,800	4,134	2,506	1,722	1,060
Q	9,848	2,949	1,787	1,228	756
H	7,688	2,301	1,394	958	590
35	157x157	L	18,448	5,529	3,351
M	14,496	4,343	2,632	1,809	1,113
Q	10,288	3,081	1,867	1,283	790
H	7,888	2,361	1,431	983	605
36	161x161	L	19,472	5,836	3,537
M	15,312	4,588	2,780	1,911	1,176

Version	Modules	ECC Level	Data bits	Numeric	Alphanumeric
Q	10,832	3,244	1,966	1,351	832
H	8,432	2,524	1,530	1,051	647
37	165x165	L	20,528	6,153	3,729
M	15,936	4,775	2,894	1,989	1,224
Q	11,408	3,417	2,071	1,423	876
H	8,768	2,625	1,591	1,093	673
38	169x169	L	21,616	6,479	3,927
M	16,816	5,039	3,054	2,099	1,292
Q	12,016	3,599	2,181	1,499	923
H	9,136	2,735	1,658	1,139	701
39	173x173	L	22,496	6,743	4,087
M	17,728	5,313	3,220	2,213	1,362
Q	12,656	3,791	2,298	1,579	972
H	9,776	2,927	1,774	1,219	750
40	177x177	L	23,648	7,089	4,296
M	18,672	5,596	3,391	2,331	1,435
Q	13,328	3,993	2,420	1,663	1,024
H	10,208	3,057	1,852	1,273	784

## 27.1.6. Error correction level

As shown in Table 27.2, “Maximum data capacity for the different symbol sizes in the QR-code.” the QR standard specifies four different error correction levels. In the library the error correction can either be set to be chosen automatically or specified manually. The properties of the available error correction levels are given in Table 27.3, “QR Error correction levels”. The “*Error correction capacity*” column specifies how large percentage of the codewords that can be destroyed and the code still being decoded.

**Table 27.3. QR Error correction levels**

Error level	Symbolic constant	Error correction capacity
L	QRCapacity::ErrL	5 %
M	QRCapacity::ErrM	15 %
Q	QRCapacity::ErrQ	25 %
H	QRCapacity::ErrH	30 %

## 27.1.7. Comparing the visual output with other QR-generating software

It is possible that the JpGraph library gives a visually different result than some other available QR encoders. As a matter of fact many QR encoders gives a different visual result from the same input. This does not mean that one QR encoder is more correct than any other. This is a consequence of interpretation of the standard in a way that (without going into technical details) does not in any way affect the decoding of the barcode. It only affects the visual appearance.

## Note

Part of the reason is that the original ISO/IEC 18004E standard is not consistent between its theoretical description and the examples that are given in the standard. For those interested in a more technically explanation of these visual issues we refer to

- "A note on minor errors in the International QR Barcode standard" [<http://www.aditus.nu/jpgraph/qr-comment.pdf>] , 2008, J. Persson

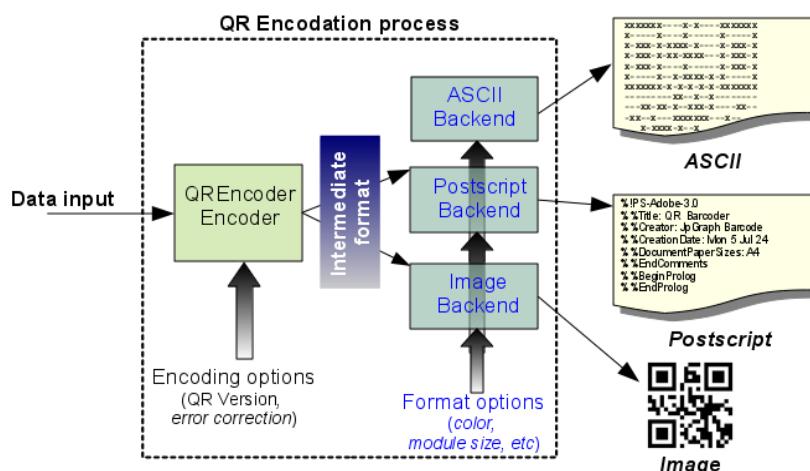
which gives a detailed analysis and explanation of these issues.

## 27.2. Creating barcodes

In order to use QR codes the module "QR/qrencoder.inc.php" must first be included.

Usage of QR codes follows a similar schema as for the linear, PDF417 and Datamatrix barcodes with the concepts of an encoder and backend. The principle of the overall encodation process is shown in Figure 26.4, "Datamatrix encodation principle"

**Figure 27.5. QR Encodation Process**



### 27.2.1. Getting started

In order to use the QR barcodes the library module "QR/qrencoder.inc.php" must first be included in the script.

1. Create an instance of the encoder and optionally specify size and error correction level. The encoder is created as an instance of class `QREncoder`
2. Create an instance of a suitable backend for the chosen output format (image or postscript) by calling the `DatamatrixBackendFactory::Create()`
3. Encode data and send it back to the browser or to a file with a call to the backend `Backend::Stroke()` method.

The following script shows how to create the simplest possible QR code (in PNG format) representing the data string "The first QR code" encoded using all default values. The resulting QR Code is shown in Figure 27.6, "The first very basic QR code (qrexample00.php)"

**Example 27.1. The first very basic QR code (`qrexample00.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/QR/qrencoder.inc.php');

// Data to be encoded
$data = '01234567';

// Create a new instance of the encoder and let the library
// decide a suitable QR version and error level
$e=new QREncoder();

// Use the image backend (this is also the default)
$b=QRCodeBackendFactory::Create($e);

// .. send the barcode back to the browser for the data
$b->Stroke($data);
?>
```

**Figure 27.6. The first very basic QR code (`qrexample00.php`)**  
`[example_src/qrexample00.html]`



(As can be seen this follows the same principles as the creation of Datamatrix symbols)

The principle is the same for all type of data QR code creations. First an instance of the chosen encoder is instantiated. In this case using all default parameters by creating an instance of class `QREncoder`. We then create a suitable backend that handles the output of the barcode. By default the output will be an image encoded in the PNG image format.

The final step is to send back the generated image to the browser with a call to the method `Backend::Stroke()` with the data to be encoded as its first argument.

The example above does not have any error handling. If there is some error in the process an exception will be thrown in the same way as in other places in the library. The default exception will display a standard library image error box. An example of this is shown in Figure 27.7, “QR Error message”

**Figure 27.7. QR Error message**



If some additional processing is necessary and to just display a text based re-formatted error message we could change the above code to catch this exception as the following example shows.

**Example 27.2. Adding basic exception handling (`qrexample0.php`)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/QR/qrencoder.inc.php');

// Data to be encoded
$data = '01234567';

// Create a new instance of the encoder and let the library
// decide a suitable QR version and error level
$encoder = new QREncoder(1);

// Use the image backend (this is also the default)
$backend = QRCodeBackendFactory::Create($encoder);

try {
 // . send the QR Code back to the browser
 $backend->Stroke($data);
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 echo 'QR Code error: '.$e->GetMessage()."\\n";
 exit(1);
}

?>
```

**Figure 27.8. Adding basic exception handling (`qrexample0.php`)**  
[`example_src/qrexample0.html`]



As a final initial example the next script uses the backend method `Backend::SetModuleWidth($aWidth)` to increase the size of one module. Since a module in QR code refers to the smallest square used adjusting the module width will also adjust the height of the overall QR code

**Example 27.3. Adjusting the module width for the QR code (qrexample01.php)**

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/QR/qrencoder.inc.php');

// Data to be encoded
$data = '01234567';

// Create a new instance of the encoder and let the library
// decide a suitable QR version and error level
$encoder = new QREncoder();

// Use the image backend
$backend = QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

// Set the module size (quite big)
$backend->SetModuleWidth(5);

// .. send the barcode back to the browser for the data
$backend->Stroke($data);
?>
```

**Figure 27.9. Adjusting the module width for the QR code (qrexample01.php)  
[example\_src/qrexample01.html]**



## 27.2.2. Error handling

As in other parts of the library the QR module throws an exception when an error occurs. As shown in the previous section an error image is the default if no explicit `try {} catch {}` statement is added to the script.

### Note

The fact that an image is shown as the default exception is caused by the library module "qrexception.inc.php" installing its own default error handler which will be called when an exception happens. However, this handler is intelligent enough to know if the exception is caused by the library or not. If the exception is not caused by the library the exception is re-thrown and the original error handler restored. This means that JpGraph will not interfere with previously installed error handlers for other parts of an application.

The script below shows how to catch the error, do some possible clean up and then explicitly create and send back the error image.

```
<?php
...
try {
```

```

 $backend->Stroke($data);
 } catch(Exception $e) {
 doCleanup();
 $errobj = new QRErrObjectImg();
 $errobj->Raise($e->getMessage());
 }
?>

```

another variant of this would be to re-throw the exception after the cleanup has been performed as the following example shows

```

<?php
...
try {
 $backend->Stroke($data);
} catch(Exception $e) {
 doCleanup();
 throw $e;
}
?>

```

It is also possible to get hold of the internal error code that corresponds to each error message by calling the PHP standard exception method `Exception::getCode()` as the following example shows

```

<?php
...
try {
 $backend->Stroke($data);
} catch (Exception $e) {
 $errstr = $e->GetMessage();
 $errcode = $e->GetCode();
 echo "QR error ($errcode). Message: $errstr\n";
}
?>

```

Table 27.4, “QR Error messages” lists the public errors that can be thrown by the QR code module (the table deliberately excludes internal error messages)

**Table 27.4. QR Error messages**

Error number	Error string
1000	Tilde processing is not yet supported for QR Bar-codes.
1001	Inverting the bit pattern is not supported for QR Bar-codes.
1002	Cannot read data from file %s
1003	Cannot open file %s
1004	Cannot write QR barcode to file %s
1005	Unsupported image format selected. Check your GD installation
1006	Cannot set the selected barcode colors. Check your GD installation and spelling of color name
1007	QR Error: HTTP headers have already been sent.

Error number	Error string
	<p>Caused by output from file %s at line %d</p> <p>Explanation: HTTP headers have already been sent back to the browser indicating the data as text before the library got a chance to send its image HTTP header to this browser. This makes it impossible for the QR library to send back image data to the browser (since that would be interpreted as text by the browser and show up as junk text).</p> <p>Most likely you have some text in your script before the call to QRBackend::Stroke().</p> <p>If this text gets sent back to the browser the browser will assume that all data is plain text. Look for any text (even spaces and newlines) that might have been sent back to the browser.</p> <p>For example it is a common mistake to leave a blank line before the opening "&lt;?php"</p>
1008	Could not create the barcode image with image format=%s. Check your GD/PHP installation.
1009	Cannot open log file %s for writing.
1010	Cannot write log info to log file %s.
1011	Could not write the QR Code to file. Check the file system permissions.
1400	QR Version must be specified as a value in the range [1,40] got %d
1401	Input data to barcode can not be empty.
1402	Automatic encodation mode was specified but input data looks like specification for manual encodation.
1403	Was expecting an array of arrays as input data for manual encoding.
1404	Each input data array element must consist of two entries. Element \$i has of \$nn entries
1405	Each input data array element must consist of two entries with first entry being the encodation constant and the second element the data string. Element %d is incorrect in this respect.
1406	Was expecting either a string or an array as input data
1407	Manual encodation mode was specified but input data looks like specification for automatic encodation.
1408	Input data too large to fit into one QR Symbol
1409	The selected symbol version %d is too small to fit the specified data and selected error correction level.

Error number	Error string
1410	Trying to read past the last available codeword in block split.
1415	Manually specified encodation schema MODE_NUMERIC has no data that can be encoded using this schema.
1416	Manually specified encodation schema MODE_ALPHANUM has no data that can be encoded using this schema.
1417	Manually specified encodation schema MODE_BYTEx has no data that can be encoded using this schema.
1418	Unsupported encodation schema specified (%d)
1419	Found character in data stream that cannot be encoded with the selected manual encodation mode.
1420	Encodation using KANJI mode not yet supported.
1422	Found unknown characters in the data stream that can't be encoded with any available encodation mode.
1423	Kanji character set not yet supported.
1427	Expected either DIGIT, ALNUM or BYTE but found ASCII code=%d

### 27.2.3. Creating an encoder

A QR encoder is created as an instance of class `QREncoder`. The constructor have the following signature

- `QREncoder::__construct($aVersion = -1, $aErrLevel = QRCapacity::ErrM)`
- `$aVersion`, Specifies the QR version and is specified as an integer in the range [1,40], see Table 27.2, “Maximum data capacity for the different symbol sizes in the QR-code.”
- `$aErrLevel`, Specifies the error correction level to be used, see Table 27.3, “QR Error correction levels”

The following examples creates a basic encoder which will automatically select the smallest possible size to fit the data given

```
$encoder = new QREncoder();
```

The following example will create an instance of an encoder with size 22 and error correction level 'Q'

```
$encoder = new QREncoder(22, QRCapacity::ErrQ);
```

### 27.2.4. Encodation of input data options

The absolute simplest way of encoding data is simply to create a simple string representing the data to be encoded and then pass that string as the first argument to the `Stroke()` method in the backend. The encoder will then analyze the input data and choose the most efficient space saving encoding schema for this data.

## Note

Unless there are some really good reasons to use manual encodation it should be left to the library to determine this in an optimal way!

The QR standard allows 3 different compaction schema that can be used to minimize the number of code-words used for a particular data string. This also means that a particular data string may have several different valid barcodes that visually looks different.

The supported compaction modes in the library are:

- **Alpha compaction mode.** Efficient encoding of digits 0 - 9; upper case letters A -Z and nine other characters: space, \$ % \* + - . / : .
- **Numeric compaction mode.** Efficient encoding of numeric data. For long consecutive strings of digits this gives a better compaction than the alpha mode.
- **Byte compaction mode.** Used only when there is a need to encode byte values as is, i.e. values in the range 0-255. Please note that some barcode readers, especially those with a keyboard wedge, don't send back the proper encoding for ASCII values lower than 32 or higher than 126.

When the automatic encoding is chosen this will create an optimum encoding (from a size perspective) of the supplied data. This includes shifting encoding method in the middle of the data one or several time depending on the structure of the data.

It is also possible to manually control the exact encodation of the input data. This is done by supplying one or more data tuples where the first entry in the tuple is the compaction schema and the second the data. To encode the data manually the following structure must then be followed:

**Figure 27.10. Structure for manually specify QR encodation schema**

```
$data = array(array(<encoding_mode1> , <data1>),
 array(<encoding_mode2> , <data2>),
 ...
 array(<encoding_modeN> , <dataN>));
```

The encoding mode is specified as one of three symbolic constants

- `QREncoder::MODE_NUMERIC`
- `QREncoder::MODE_ALPHANUM`
- `QREncoder::MODE_BYTE`

and the data is specified as a regular text string. Each section of data must therefore have the compaction mode specified.

In order to use a specific encodation schema one has to create an array as input instead of the normal string input. For example, to specify an input where the encodation should be alphanumeric encoding of the digits '0123456789' one would have to create the array

```
<?php
$data = array(array(QREncoder::MODE_ALPHANUM, '0123456789'));
...
$backend->Stroke($data)
?>
```

It is also possible to use different encodings for different parts of the data in a similar way as the following example shows

```
<?php
$data = array(array(QREncoder::MODE_ALPHANUM, '0123456789') ,
 array(QREncoder::MODE_BYTE, 'ABCDEFGH')
);
...
...
$backend->Stroke($data)
?>
```

## 27.2.5. Reading input data from a file

Normally the method Backend::Stroke(\$aData) is used to create a QR code from the datastring given. This string can of course come from a database, user input or from a file. The library provides a utility method

- Backend::StrokeFromFile(\$aFromFile, \$aToFile= '')  
\$aFromFile, The file to read the data from  
\$aToFile, Optional filename to write the generated QR code to

which makes it easy to get data directly from a file. Otherwise it behaves exactly the same as the normal Backend::Stroke().

## 27.2.6. Creating different backends

In order to create the actual output one or more backends must be created. The QR Code supports the following backends:

- BACKEND\_IMAGE, Create an image backend. This is the default backend if no explicit backend is specified.
- BACKEND\_PS, Create a postscript backend. The text string that represents the postscript for the barcode is returned directly from the Backend::Stroke()
- BACKEND\_EPS, Create an encapsulated postscript backend
- BACKEND\_ASCII, This is a special backend which will generate an ASCII rendering of the datamatrix barcode. This is mostly practical for technical investigations regarding the technical structure of a Datamatrix barcode.

The following code snippet shows two ways of creating a barcode image backend.

```
<?php
$encoder = new QREncoder();
$backend = QRCodeBackendFactory::Create($encoder);
...
?>

<?php
$encoder = new QREncoder();
$backend = QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);
...
```

```
?>
```

The following code shows a complete script to generate a postscript output

### Example 27.4. Generating Postscript output (`qrexample11.php`)

```
<?php // content="text/plain; charset=utf-8"
// Include the library
require_once('jpgraph/QR/qrencoder.inc.php');

// Example 11 : Generate postscript output

$data = 'ABCDEFGH01234567'; // Data to be encoded
$version = -1; // -1 = Let the library decide version (same as default)
$corrlevel = QRCapacity::ErrH; // Error correction level H (Highest possible)
$modulewidth = 3;

// Create a new instance of the encoder using the specified
// QR version and error correction
$encoder = new QREncoder($version,$corrlevel);

// Use the image backend
$backend = QRCodeBackendFactory::Create($encoder, BACKEND_PS);

// Set the module size
$backend->SetModuleWidth($modulewidth);

// Store the barcode in the specified file
$ps_str = $backend->Stroke($data);

echo '<pre>' . $ps_str . '</pre>';
?>
```

**Figure 27.11. QR Code - Example Postscript Output**

```
%!PS-Adobe-3.0
%Title: QR Barcode 2-H, mask=6
%Creator: JpGraph Barcode http://www.aditus.nu/jpgraph/
%CreationDate: Mon 6 Jul 16:08:12 2009
%DocumentPaperSizes: A4
%EndComments
%BeginProlog
%EndProlog
%Page: 1 1

%Data: ABCDEFGH01234567
%Each line represents one row and the x-position for black modules: [xpos]

3.05 setlinewidth
[[12][15][18][21][24][27][20][45][51][66][69][72][75][78][81][84]] (0) forall 87 moveto 0 -3.05 rlineto stroke) forall
[[12][30][45][66][84]] (0) forall 84 moveto 0 -3.05 rlineto stroke) forall
[[12][18][21][24][30][36][39][42][45][51][66][72][75][78][84]] (0) forall 81 moveto 0 -3.05 rlineto stroke) forall
[[12][18][21][24][30][36][39][45][51][54][66][72][75][78][84]] (0) forall 78 moveto 0 -3.05 rlineto stroke) forall
[[12][18][21][24][30][45][66][72][75][78][84]] (0) forall 75 moveto 0 -3.05 rlineto stroke) forall
[[12][30][39][45][66][84]] (0) forall 72 moveto 0 -3.05 rlineto stroke) forall
[[12][15][18][21][24][27][30][42][48][54][60][66][69][72][75][78][81][84]] (0) forall 69 moveto 0 -3.05 rlineto stroke) forall
[[39][42][51][66][84]] (0) forall 66 moveto 0 -3.05 rlineto stroke) forall
[[12][18][21][24][30][33][42][45][51][54][57][60][66][72][75][78][84]] (0) forall 63 moveto 0 -3.05 rlineto stroke) forall
[[12][18][21][24][30][33][42][51][54][57][60][66][72][75][78][84]] (0) forall 60 moveto 0 -3.05 rlineto stroke) forall
[[12][21][27][30][33][36][42][51][54][57][60][66][72][75][78][84]] (0) forall 57 moveto 0 -3.05 rlineto stroke) forall
[[18][21][27][39][45][51][54][57][60][66][69][72][75][78][84]] (0) forall 54 moveto 0 -3.05 rlineto stroke) forall
[[12][15][18][21][24][27][30][33][39][51][57][63][69][72][75][78][84]] (0) forall 51 moveto 0 -3.05 rlineto stroke) forall
[[12][15][21][48][66][81][84]] (0) forall 48 moveto 0 -3.05 rlineto stroke) forall
[[12][15][18][21][24][30][36][51][54][57][72][75][81]] (0) forall 45 moveto 0 -3.05 rlineto stroke) forall
[[12][18][24][27][39][45][51][60][75][78][84]] (0) forall 42 moveto 0 -3.05 rlineto stroke) forall
[[12][24][27][39][43][42][45][51][57][60][66][72][75][78][84]] (0) forall 39 moveto 0 -3.05 rlineto stroke) forall
[[36][42][51][66][69][72][75][78][84]] (0) forall 36 moveto 0 -3.05 rlineto stroke) forall
[[12][15][18][21][24][31][51][54][57][60][66][69][72][75][78][84]] (0) forall 33 moveto 0 -3.05 rlineto stroke) forall
[[12][30][39][42][45][51][54][57][60][66][69][72][75][81][84]] (0) forall 30 moveto 0 -3.05 rlineto stroke) forall
[[12][18][21][24][30][36][42][45][51][54][60][66][69][72][75][78][84]] (0) forall 27 moveto 0 -3.05 rlineto stroke) forall
[[12][18][21][24][30][39][45][51][60][66][69][72][75][81][84]] (0) forall 24 moveto 0 -3.05 rlineto stroke) forall
[[12][18][21][24][30][39][45][51][60][66][69][72][75][81][84]] (0) forall 21 moveto 0 -3.05 rlineto stroke) forall
[[12][30][39][42][45][51][60][63][66][69][72][75][78][81][84]] (0) forall 18 moveto 0 -3.05 rlineto stroke) forall
[[12][15][18][21][24][27][30][42][48][54][57][60][66][69][72][75][78][81][84]] (0) forall 15 moveto 0 -3.05 rlineto stroke) forall

%End of QR Barcode
showpage
%Trailer
```

## 27.2.7. Generic backend methods

The following methods are available to adjust the final look and feel of the barcode

1. `Backend::SetColor($aOne, $aZero, $aBackground='white')`

the color of the 'black' and 'white' modules in the barcode and the quiet area around the barcode

2. `Backend::SetQuietZone($aSize)`

the size of the "quiet zone" in pixels for the image backend and in points (1 pt = 1/72 inch) for the postscript backend

3. `Backend::SetModuleWidth($aWidth)`

the module width specified in pixels for the image backend and in points (1 pt = 1/72 inch) for the postscript backend

4. `Backend::Stroke($aData, $aFileName= '')`

create the barcode and send it back the client or store it to a file if the second parameter is set. For postscript backends the postscript string is returned directly from the method. (See example above)

5. `Backend::StrokeFromFile($aFromFileName, $aFileName= '')`

This allows the creation of barcode directly from data in a file.

## 27.2.8. Image backend methods

For the image backend it is possible to adjust the image encoding format with the following method

1. `Backend::SetImgFormat($aFormat, $aQuality=75)`

Specify image format. Possible values are

- 'auto'
- 'png'
- 'jpeg'
- 'gif'

For 'jpeg' format the quality parameter is a number in range [1-100] and specified how much compression / (Data loss) should be used. 100=no compression. Normal values are in the range [60-95]

The following example sets the image format to 'JPEG' with quality 80.

```
$backend->SetImgFormat('jpeg', 80);
```

## 27.2.9. Postscript backend methods

For postscript backend it is possible to select whether the postscript should be generated as encapsulated postscript. This is controlled by the method

- `Backend::SetEPS($aFlg=true)`

## 27.2.10. A template to create barcodes

The following (complete) script can be used as a starting point for creation of more advanced barcode scripts

### Example 27.5. A QR code template (`qr_template.php`)

```
<?php // content="text/plain; charset=utf-8"
require_once('jpgraph/QR/qrencoder.inc.php');

// Data to be encoded
$data = 'ABCDEFGH01234567';

// QR Code specification
$version = -1; // -1 = Let the library decide version (same as default)
$corrlevel = QRCapacity::ErrM; // Medium error correction
$modulewidth = 2; // Module width
$back = BACKEND_IMAGE; // Default backend
$quiet = 4; // Same as default value

// Create encoder and backend
$encoder = new QREncoder($version, $corrlevel);
$backend = QRCodeBackendFactory::Create($encoder, $back);

// Set the module size
$backend->SetModuleWidth($modulewidth);

// Set Quiet zone (this should rarely need changing from the default)
$backend->SetQuietZone($quiet);

if($back == BACKEND_IMAGE) {

 $backend->Stroke($data);
}
else {
 $str = $backend->Stroke($data);
 echo '<pre>' . $str . '</pre>';
}
?>
```

**Figure 27.12. A QR code template (`qr_template.php`) [example\_src/`qr_template.html`]**



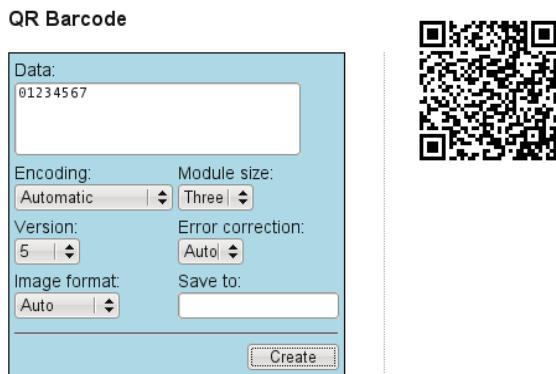
## 27.2.11. Sample application

As an example the library includes a WEB-based demo barcode creation application. This application can be used to easily create barcode through its WEB interface. It is available at '`qr/demoapp/index.html`'

This application is primarily included as a demo on the features available in the library and not as a finalized product.

Figure 27.13, “QR Code WEB-based demo application” shows a screen shot of the application interface.

**Figure 27.13. QR Code WEB-based demo application**



## 27.3. Example scripts

The following sections shows a number of example script that can be used as a base for further refinement.

### 27.3.1. Example 1 - Store QR code to file

QR Barcode with specified version and stored to a file

**Example 27.6. Storing image to a file (qrexample03.php)**

```
<?php // content="text/plain; charset=utf-8"
 // Example 3 : QR Barcode with specified version and stored to a file

 // Include the library
 require_once('jpgraph/QR/qrencoder.inc.php');

 // Data to be encoded
 $data = '01234567';
 $version = 3; // Use QR version 3
 $fileName = 'qrexample03.png';

 // Create a new instance of the encoder and let the library
 // decide a suitable error level
 $encoder = new QRCodeEncoder($version);

 // Use the image backend
 $backend = QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

 // Set the module size (quite big)
 $backend->SetModuleWidth(5);

 // Store the barcode in the specified file
 $backend->Stroke($data,$fileName);
 list($version,$errorcorrection) = $backend->GetQRInfo();

 echo "QR Barcode, (Version: $version-$errorcorrection), image stored in
?>
```

### 27.3.2. Example 2 - Specified error correction level

QR Barcode with specified version and error correction level

**Example 27.7. Specified error correction level (qrexample04.php)**

```
<?php // content="text/plain; charset=utf-8"
 // Example 4 : QR Barcode with specified version and error correction level

 // Include the library
 require_once('jpgraph/QR/qrencoder.inc.php');

 // Data to be encoded
 $data = '01234567';
 $version = 12; // Use QR version 4
 $corrlevel = QRCapacity::ErrH; // Level H error correction (the highest possi

 // Create a new instance of the encoder using the specified
 // QR version and error correction
 $encoder = new QREncoder($version,$corrlevel);

 // Use the image backend
 $backend = QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

 // Set the module size
 $backend->SetModuleWidth(3);

 // Store the barcode in the specified file
 $backend->Stroke($data);
?>
```

**Figure 27.14. Specified error correction level (qrexample04.php)  
[example\_src/qrexample04.html]**



### 27.3.3. Example 3 - Manual specified encoding

QR Barcode with manually specified encodation

**Example 27.8. QR Barcode with manually specified encodation  
(qrexample05.php)**

```
<?php // content="text/plain; charset=utf-8"
 // Example 5 : QR Barcode with manually specified encodation

 // Include the library
 require_once('jpgraph/QR/qrencoder.inc.php');

 // Data to be encoded
 // We want the data to be encoded using alphanumeric encoding even though
 // it is only numbers
 $data = array(
 array(QREncoder::MODE_ALPHANUM, '01234567')
);

 $version = 3; // Use QR version 3
 $corplevel = QRCapacity::ErrH; // Level H error correction (the highest possi

 // Create a new instance of the encoder using the specified
 // QR version and error correction
 $encoder = new QREncoder($version,$corplevel);

 // Use the image backend
 $backend = QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

 // Set the module size
 $backend->SetModuleWidth(4);

 // Store the barcode in the specified file
 $backend->Stroke($data);
?>
```

**Figure 27.15. QR Barcode with manually specified encodation  
(qrexample05.php) [example\_src/qrexample05.html]**



#### 27.3.4. Example 4 - JPEG image format

QR Barcode with image in JPG format

**Example 27.9. QR Barcode with image in JPG format (`qrexample06.php`)**

```
<?php // content="text/plain; charset=utf-8"
 // Example 6 : QR Barcode with image in JPG format

 // Include the library
 require_once('jpgraph/QR/qrencoder.inc.php');

 $data = 'ABCDEFGH01234567'; // Data to be encoded
 $version = -1; // -1 = Let the library decide version (same as default)
 $corrlevel = -1; // -1 = Let the library decide error correction level (same as default)

 // Create a new instance of the encoder using the specified
 // QR version and error correction
 $encoder = new QREncoder($version,$corrlevel);

 // Use the image backend
 $backend=QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

 // Use JPEG format with 80% quality level
 $backend->SetImgFormat('jpeg',80);

 // Set the module size
 $backend->SetModuleWidth(4);

 // Store the barcode in the specified file
 $backend->Stroke($data);
?>
```

**Figure 27.16. QR Barcode with image in JPG format (`qrexample06.php`)**  
[`example_src/qrexample06.html`]



### 27.3.5. Example 5 - Multiple manual encoding

QR Barcode with multiple manually specified encodation schema.

#### Note

Normally there would be no need to ever use manually specified encodation)

**Example 27.10. multiple manually specified encodation schema.  
(qrexample07.php)**

```
<?php // content="text/plain; charset=utf-8"
 // Example 7 : QR Barcode with multiple manually specified encodation schemas
 // (Note: Normally there would be no need to ever use manually specified encod

 // Include the library
 require_once('jpgraph/QR/qrencoder.inc.php');

 // Data to be encoded
 $data = array(
 array(QREncoder::MODE_ALPHANUM, '01234567'),
 array(QREncoder::MODE_NUMERIC, '89012345')
);

 // Create a new instance of the encoder (automatically determined QR version a
 // error correction level)
 $encoder = new QREncoder();

 // Use the image backend
 $backend = QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

 // Set the module size
 $backend->SetModuleWidth(4);

 // Store the barcode in the specified file
 $backend->Stroke($data);
?>
```

**Figure 27.17. multiple manually specified encodation schema.  
(qrexample07.php) [example\_src/qrexample07.html]****27.3.6. Example 6 - Reading data from file**

QR Barcode with data read from file

**Example 27.11. data read from file (qrexample08.php)**

```
<?php // content="text/plain; charset=utf-8"
 // Example 8 : QR Barcode with data read from file

 // Include the library
 require_once('jpggraph/QR/qrencoder.inc.php');

 $readFromFilename = 'qr-input.txt';

 // Create a new instance of the encoder and let the library
 // decide a suitable QR version and error level
 $encoder=new QREncoder();

 // Use the image backend
 $backend=QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

 // Set the module size (quite big)
 $backend->SetModuleWidth(5);

 // .. send the barcode back to the browser for the data in the file
 $backend->StrokeFromFile($readFromFilename);
?>
```

**Figure 27.18. data read from file (qrexample08.php) [example\_src/qrexample08.html]**



### 27.3.7. Example 7 - Adjusting colors

QR Barcode with data read from file and different colors

**Example 27.12. (qrexample09.php)**

```
<?php // content="text/plain; charset=utf-8"
 // Example 9 : QR Barcode with data read from file and different colors

 // Include the library
 require_once('jpgraph/QR/qrencoder.inc.php');

 $readFromFilename = 'qr-input.txt';

 // Create a new instance of the encoder and let the library
 // decide a suitable QR version and error level
 $encoder = new QREncoder();

 // Use the image backend
 $backend = QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

 // Set the module size (quite big)
 $backend->SetModuleWidth(5);

 // Use blue and white colors instead
 $backend->SetColor('navy','white');

 // .. send the barcode back to the browser for the data in the file
 $backend->StrokeFromFile($readFromFilename);
?>
```

**Figure 27.19. (qrexample09.php) [example\_src/qrexample09.html]**



### 27.3.8. Example 8 - Setting error correction level

Specified error correction level

**Example 27.13. Specified error correction level (`qrexample10.php`)**

```
<?php // content="text/plain; charset=utf-8"
 // Example 10 : Specified error correction level

 // Include the library
 require_once('jpgraph/QR/qrencoder.inc.php');

 $data = 'ABCDEFGH01234567'; // Data to be encoded
 $version = -1; // -1 = Let the library decide version (same as default)
 $corrlevel = QRCapacity::ErrH; // Error correction level H (Highest possible)

 // Create a new instance of the encoder using the specified
 // QR version and error correction
 $encoder = new QREncoder($version,$corrlevel);

 // Use the image backend
 $backend = QRCodeBackendFactory::Create($encoder, BACKEND_IMAGE);

 // Set the module size
 $backend->SetModuleWidth(3);

 // Set color
 $backend->SetColor('brown','white');

 // Store the barcode in the specified file
 $backend->Stroke($data);
?>
```

**Figure 27.20. Specified error correction level (`qrexample10.php`)**  
[`example_src/qrexample10.html`]



### 27.3.9. Example 9 - Generating ASCII output

Using the ASCII backend

**Example 27.14. Using the ASCII backend (qrexample12.php)**

```
<?php // content="text/plain; charset=utf-8"
// Include the library
require_once('jpgraph/QR/qrencoder.inc.php');

// Example 11 : Generate postscript output

$data = 'ABCDEFGH01234567'; // Data to be encoded
$version = -1; // -1 = Let the library decide version (same as default)
$corrlevel = QRCapacity::ErrH; // Error correction level H (Highest possible)
$modulewidth = 1;

// Create a new instance of the encoder using the specified
// QR version and error correction
$encoder = new QREncoder($version,$corrlevel);

// Use the image backend
$backend = QRCodeBackendFactory::Create($encoder, BACKEND_ASCII);

// Set the module size
$backend->SetModuleWidth($modulewidth);

// Store the barcode in the specified file
$ps_str = $backend->Stroke($data);

echo '<pre>' . $ps_str . '</pre>';
?>
```

**Figure 27.21. QR With ASCII rendering**

```
XXXXXXXX-----X-X----XXXXXXXX
X-----X-----X-----X
X-XXX-X-XXXX-X---X-XXX-X
X-XXX-X-XX-XXXXX---X-XXX-X
X-XXX-X---X-----X-XXX-X
X-----X-X-XXXX-X-X-----X
XXXXXXXX-X-X-X-X-X-XXXXXX
-----X-X-X-X-----
--XX-XX-X-XXX-X-----XX-
-XX-X---XXXXXX---X-XXX
X--X-XXXX-X---XXX-X-XX-X
--XX-X---X-X-X-XXXX---XX
XXX-XXXX-X---X-X-X-X-X
XX-X-----X-----XX---XX
XXXX-X-X---XXX---X-XX-
X-X-XX---X-X-X---XX-X
X---XXXX---X-X-XXXXXX---
-----X-XX---X---XXX-X
XXXXXXXX-XX---XXXXX-X-X-XX
X---X-XXX-XXXX---X---X
X-XXX-X-X-XX-X-XXXXXX-
X-XXX-X-XXX---X---X-X
X-XXX-X-X-XX-X---XXX-X
X---X-XXXX---XXXX-X-XX-
XXXXXXXX---X-X-XXXXXXXXXX
```

---

## **Part VII. Case studies**

---

## Table of Contents

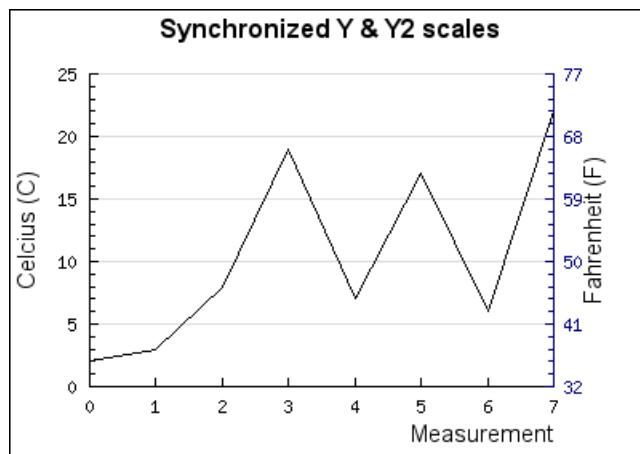
28. Synchronized Y-axis .....	599
28.1. Creating two scales .....	599
29. USPS Confirmation Barcodes .....	602
29.1. Creating the confirmation codes .....	602
30. Showing SPAM statistics .....	606
30.1. Introduction and purpose .....	606
30.2. Step 1: Parsing the log file .....	608
30.3. Step 2: Creating the graph .....	612
30.4. Step 3: Uploading the image file to a server .....	613
30.5. Step 0: The full driver script .....	614
31. Creating Critical chain buffer penetration charts .....	621
31.1. Introduction and purpose .....	621
31.2. Creating a utility class to construct CC BP charts .....	626
31.3. The Init() method .....	628
31.4. Suggested improvements .....	630
31.5. The implementation of class CCBPGraph .....	630
31.6. References .....	632

# Chapter 28. Synchronized Y-axis

The purpose of this chapter is to show how one can have one set of data displayed with two different scales that are still synchronized meaning that the tick marks are at identical positions.

One classic example of when this can be useful is to show temperature in both Celsius and Fahrenheit. Figure 28.1, “Synchronized y and y2 scales (y2synch.php) ” below illustrates what we would like to accomplish.

**Figure 28.1. Synchronized y and y2 scales (y2synch.php) [example\_src/y2synch.html]**



## 28.1. Creating two scales

Normally the auto scaling puts tick marks at "nice" positions that for example are multiples of 5,2,10 and so on. This would mean that it seems like the graph in Figure 28.1, “Synchronized y and y2 scales (y2synch.php) ” with tick positions at 59 F and 68 F can not be auto scaled.

The simplest method of creating two scales as in the example above is to add a secondary Y-axis, add the same plot to that axis. This would then mean that the two axes would be identical. The magic now happens in the last step when we add a format callback method to the second Y-axis. This format callback will be applied to each label. All we have to do now is create a callback function that does the proper scale conversion.

This way also means that we have to chose one of the scales as the "master" which will be auto scaled with tick positions at "nice" intervals.

### Caution

For the second scale one can add the same plot or one could create a new plot from the same data set. If the data set to be plotted is very large it can be advantageous to create a new plot and set the line weight to 0 (zero). This way the line will not actually be plotted and that plot time is eliminated.

The script to create the graph in Figure 28.1, “Synchronized y and y2 scales (y2synch.php) ” is given below.

```

<?php // content="text/plain; charset=utf-8"
require_once ("jpgraph/jpgraph.php");
require_once ("jpgraph/jpgraph_line.php");
require_once ("jpgraph/jpgraph_bar.php");

function toFahrenheit($aVal) {
 return round(($aVal*9/5)+32,2);
}

function toCelcius($aVal) {
 return round(($aVal-32)*5/9,2);
}

$datay =array(2,3,8,19,7,17,6,22);

// Create the graph.
$graph = new Graph(400,280);

// Slightly bigger margins than default to make room for titles
$graph->SetMargin(50,60,40,45);
$graph->SetMarginColor('white');

// Setup the scales for X,Y and Y2 axis
$graph->SetScale("intlin"); // X and Y axis
$graph->SetY2Scale("lin"); // Y2 axis

// Overall graph title
$graph->title->Set('Synchronized Y & Y2 scales');
$graph->title->SetFont(FF_ARIAL,FS_BOLD,12);

// Title for X-axis
$graph->xaxis->title->Set('Measurement');
$graph->xaxis->title->SetMargin(5);
$graph->xaxis->title->SetFont(FF_ARIAL,FS_NORMAL,11);

// Create Y data set
$lplot = new LinePlot($datay);
$graph->yaxis->title->Set('Celcius (C)');
$graph->yaxis->title->SetMargin(5);
$graph->yaxis->title->SetFont(FF_ARIAL,FS_NORMAL,11);
// ... and add the plot to the Y-axis
$graph->Add($lplot);

// Create Y2 scale data set
$l2plot = new LinePlot($datay);
$l2plot->SetWeight(0);
$graph->y2axis->title->Set('Fahrenheit (F)');
$graph->y2axis->title->SetMargin(5); // Some extra margin to clear labels
$graph->y2axis->title->SetFont(FF_ARIAL,FS_NORMAL,11);
$graph->y2axis->SetLabelFormatCallback('toFahrenheit');
$graph->y2axis->SetColor('navy');

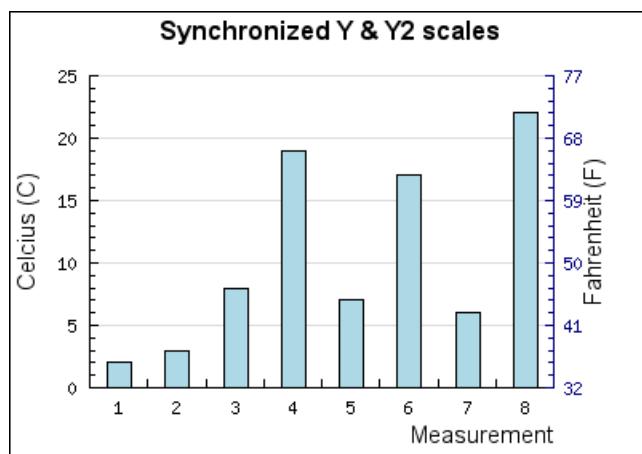
// ... and add the plot to the Y2-axis
$graph->AddY2($l2plot);

$graph->Stroke();
?>

```

The same principle will also work with other plot variants, for example bar plots. By making the first plot an instance of class BarPlot the graph in Figure 28.2, “Using a barplot with two different scales (y2synch2.php)” can be created.

**Figure 28.2. Using a barplot with two different scales (y2synch2.php) [example\_src/y2synch2.html]**



# Chapter 29. USPS Confirmation Barcodes

US Postal Service (USPS) uses EAN-128 barcodes to tag confirmation codes for the sender of parcels and letters in the case the sender is a registered business entity. The confirmation barcode gives feedback to the sender on the date, ZIP Code and the time the article was delivered.

**Figure 29.1. usps\_exhibit44.png**



The full standard describing this is available in "Confirmation Services Technical Guide [<http://www.usps.com/cpim/ftp/pubs/pub91/welcome.htm>]"

In order to create labels adhering to this strict standard it is possible to use JpGraph as a basic building block in order to create the barcode part (which uses UCC/EAN128 as mandatory coding from 2004 and onwards). The standard describes two basic forms of labels :

1. Inclusion of postal service routing information (destination ZIP code)
2. Exclusion of the destination ZIP code. This will then only include the Dun & Bradstreet Number (DUNS). The DUNS uniquely identifies businesses at specific geographical locations. For more information regarding this and how to obtain such a number please refer to USPS technical documentation.

## 29.1. Creating the confirmation codes

In order to create the final Package Identification Code (PIC) the following three steps must be taken

1. Determine the basic element of the code. This consists of identifying the Zip code (of the addressee), USPS service type, the DUNS and Sequence number of parcel (as determined by the business). These numbers are input and the process of how to get those numbers is not described further in this short note. It is assumed that a potential client will know how these numbers are obtained.
2. Determine what type of confirmation code should be used (with or without Zip code)
3. Calculate the additional checkdigit for the code and append that code to the digit sequence.

The final code that one arrives at in the final stage is then the barcode that should be created. However creating this code requires knowledge of the EAN-128 barcode format and the rules for creating barcodes that follow the highly standardized format for a EAN-128 barcode.

EAN-128 barcodes are the same as CODE-128 barcodes from a technical barcode point of view but the data to be encoded must follow a rigorous standard. The JpGraph library has built in validation to ensure that any data that is encoded using the EAN-128 symbology follows this standard.

This standard requires that special control character is inserted at specific points in the data stream. In the discussion below we will simply state what those control characters are and not discuss the general format of the EAN-128 barcodes in too much details.

In order to do this we will create a utility class with methods that does the following three things

1. Create the additional USPS Modulo-10 check digit.
2. Create a confirmation code without the ZIP number
3. Create a confirmation code with ZIP number

### The USPS\_Confirmation Utility class

For both types of confirmation code USPS uses its own checkdigit (a variant of a Modulo-10 checkdigit) at the end of the complete Package Identification Code (PIC), (the exact process for calculating this number is also described in the technical documentation released by USPS). The utility class will therefore consist of three methods, one to calculate the checkdigit and one method each to create PIC with and without ZIP code. The three methods are listed below

#### 1. function \_USPS\_chkd(\$aData)

```
<?php
// Calculate the single digit check digit from sequence of numbers
// in a string
function _USPS_chkd($aData) {
 $n = strlen($aData);

 // Add all numbers at position 0,2,4,... from the end
 $et = 0 ;
 for($i=1; $i <= $n; $i+=2) {
 $d = intval(substr($aData,-$i,1));
 $et += $d;
 }

 // Add all numbers at position 1,3,5,... from the end
 $ot = 0 ;
 for($i=2; $i <= $n; $i+=2) {
 $d = intval(substr($aData,-$i,1));
 $ot += $d;
 }

 // Calculate the checkdigit
 $tot = 3*$et + $ot;
 return (10 - ($tot % 10))%10;
}
?>
```

#### 2. function GetPIC(\$aServiceType,\$aDUNS,\$aSeqNbr)

```
<?php
```

```
// Get type 2 of confirmation code (without ZIP)
function GetPIC($aServiceType,$aDUNS,$aSeqNbr) {
 // Convert to USPS format with AI=91
 $data = '91' . $aServiceType . $aDUNS . $aSeqNbr;
 $cd = $this->_USPS_chkd($data);
 return $data . $cd;
}
?>
```

**3. function GetPICwithZIP(\$aZIP,\$aServiceType,\$aDUNS,\$aSeqNbr)**

```
<?php
// Get type 1 of confirmation code (with ZIP)
function GetPICwithZIP($aZIP,$aServiceType,$aDUNS,$aSeqNbr) {
 // Convert to USPS format with AI=420 and extension starting
 // with AI=91
 $data = '420'. $aZIP . '91' . $aServiceType .
 $aDUNS . $aSeqNbr;
 // Only calculate the checkdigit from the AI=91 and forward
 // and do not include the ~1 (FUNC1) in the calculation
 $cd = $this->_USPS_chkd(substr($data,8));
 $data = '420'. $aZIP . '~191' . $aServiceType .
 $aDUNS . $aSeqNbr;
 return $data . $cd;
}
?>
```

All that now remains is to tie this together with the EAN-128 standard barcode generation in order to create the confirmation code. The following script shows how this can be done

```
<?php
$zip = '92663'; // Zip code
$service = '21'; // Service 21 = Priority Mail
$DUNS = '805213907'; // DUNS
$seqnr = '04508735'; // Seqnr

$usps = new USPS_Confirmation();
$data = $usps->GetPICwithZIP($zip,$service,$DUNS,$seqnr);

$encoder = BarcodeFactory::Create(ENCODING_EAN128);
$e = BackendFactory::Create(BACKEND_IMAGE,$encoder);
$e->SetModuleWidth(2);
$e->SetFont(FF_ARIAL,FS_NORMAL,14);
$e->Stroke($data);
?>
```

The script above will then generate the following barcode

**Figure 29.2. USPS example 1**



420926639121805213907045087356

## Caution

Even though only numbers are input to the PIC, integers should not be used since initial "0":s will be lost. Use strings as the example above shows.

### Additional example

The following script shows how to generate PIC that does not use Zip codes.

```
<?php
$service = '01'; // Priority mail
$DUNS = '123456789'; // DUNS
$seqnr = '00000001'; // Seqnr

$usps = new USPS_Confirmation();
$data = $usps->GetPIC($service,$DUNS,$seqnr);

$encoder = BarcodeFactory::Create(ENCODING_EAN128);
$e = BackendFactory::Create(BACKEND_IMAGE,$encoder);
$e->SetModuleWidth(2);
$e->SetFont(FF_ARIAL,FS_NORMAL,14);
$e->Stroke($data);
?>
```

The script above will generate the following confirmation barcode.

**Figure 29.3. USPS example 2**



A complete script to implement this is available in the distribution as "barcode/examples/barcode\_usps\_example.php"

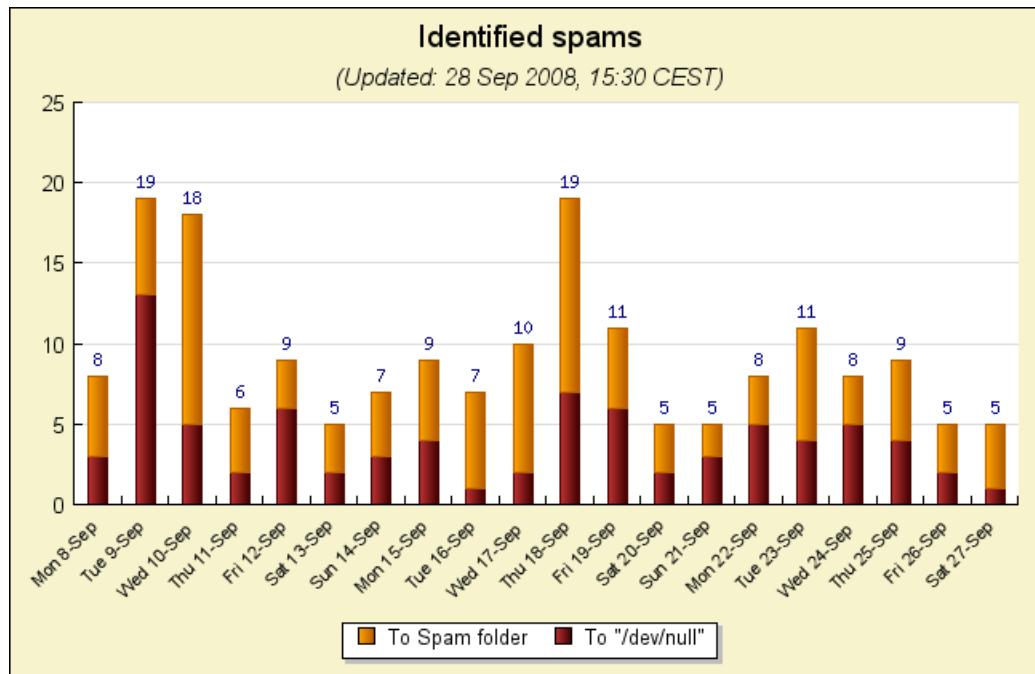
# Chapter 30. Showing SPAM statistics

## 30.1. Introduction and purpose

This worked example will show how to setup a graph that is periodically generated offline to show Spam statistics for a mail server. The graph will show daily summaries for the three things; **a)** The total number of identified spam mails **b)** The number of immediately deleted spams and **c)** The number of suspected spams that was stored in the "spam" folder instead of being immediately deleted.

An example of the graph we will generate is shown in Figure 30.1, "Spam statistics"

**Figure 30.1. Spam statistics**



The graph above makes two assumptions;

1. The spam setup has two levels of identifications and how spams are handled. Suspected emails are either deleted immediately (by sending them to **dev/null**) or stored in the user's spam folder.
2. The log files from the mail server are available for analysis.

In the following we will construct a complete PHP command line script that will be run periodically and analyse the email logs and produce a graph similar to what is shown above. The script assumes that the log file uses **procmail** log format so that the folder where each mail is stored are logged.

### Warning

For very high load email servers doing log file analysis in PHP is probably not a very good idea due to performance reasons in regards to both time and memory constraints. We do not make any claim that the scripts below are enough optimized to be used on high volume mail servers.

The script will consist of three parts

1. A parser to scan the log file and create the data
2. A suitable graph script to create an accumulated bar graph
3. Uploading of the created image file with the graph to a server where it will be displayed

We will therefore use three classes that corresponds to each step above.

To define the different files and ftp credentials we will use the following symbolic constants which will need to be defined depending on the system setup. Constants that must be adjusted is marked as "....".

```
<?php
/ FTP Server credentials
DEFINE('FTP_SERVER','....');
DEFINE('FTP_UID','....');
DEFINE('FTP_PWD','....');

// Directory on FTP server where the image should be stored
DEFINE('FTP_DIR','....');

// Which procmail logfile to read
DEFINE('PROCMAIL_LOGFILE','....');

// 2 Weeks windows to display
DEFINE('WINDOWSIZE',14);

// Where to store the temporary image file
DEFINE('IMGFILE','/tmp/spamstat.png');
?>
```

The whole process is then driven by the following relative small main script:

```
<?php
// Use the text based error handling and log potential errors to the
// system default system logger
JpGraphError::SetImageFlag(false);
JpGraphError::SetLogFile('syslog');
JpGraphError::SetTitle('Spamstat Message: ');

// Step 1) Get the statistics. We return a window of WINDOWSIZE days

$parser = new ParseProcmailLogFile(PROCMAIL_LOGFILE);
list($xdata, $ydata, $y2data) = $parser->GetStat(WINDOWSIZE);

// Step 2) Create the graph and store it in the file IMGFILE

$width = 650; $height = 420;
$graph = new SpamGraph($width,$height);
$graph->Create(IMGFILE,$xdata,$ydata,$y2data);
```

```

// Step 3) Upload the file to the FTP_SERVER server and store the
// local file IMGFILE with the same base name in directory FTP_DIR

$ftp = new FTPUploader(FTP_SERVER,FTP_UID,FTP_PWD);
$ftp->Upload(IMGFILE,FTP_DIR);
?>
```

For brevity we have excluded the lines that defines the symbolic constants above and also the inclusion of the necessary library files.

In the following sections we will shortly discuss each of the three support classes.

### Note

An actual usage of this can be found on JpGraph home page where the result of this script is run by a daily cronjob. The graph is available at <http://www.aditus.nu/jp-graph/spamstat.php>

## 30.2. Step 1: Parsing the log file

The log file we will be parsing is the standard **procmail** log. An authentic example of the log file from our main mail server (that receives around ~10,000 spam mails per month) is shown below (slightly anonymized to remove the real name of the mailbox)

```
...
From sterne@gvt.net.br Thu Jul 16 00:03:32 2009
Subject: *** SPAM (5.3) *** =?koi8-r?B?79DUyc3J2sHDydEg08HK1ME=?=
Folder: /dev/null 14278
From yhrtjgommsg@boomboomroom.com Thu Jul 16 00:13:33 2009
Subject: *** SPAM (6.7) *** Women will be begging you to sleep with you.
Folder: /srv/mail/john/.Spam/new/1247696013.18366_0.lambda 2070
From ErikaFrazier12@aol.com Thu Jul 16 00:13:33 2009
Subject: *** SPAM (41.5) *** Obama Allows Meds Sold Online
Folder: /dev/null
...
```

Each entry for a received mail in the log are three lines, sender, subject and the folder the mail will be stored in. For mails that are identified with (according to this setup) 100% confidence as spams the destination is set to `/dev/null`, i.e they are immediately deleted. For mails that are determined to be spams but where there might be a chance that they are legitimate they are stored in the users Spam folder. From the above log we can see that two mails are deleted immediately and one mail is stored in the users Spam folder.

To analyze this we will create the class `ParseProcmailLogFile` the constructor will take a file name of the log file as the only argument and to get hold of the statistics we use the method

- `ParseProcmailLogFile::GetStat($aWindowSize)`

`$aWindowSize` is the number of days back the stats should be based on. The returned statistics will be an array with three array elements with the following layout:

`($dateArray, nbrDevNull, nbrSpamFolder)`

These returned values are both arrays indexed by date and value the number of spams for that date (key)

## Caution

Remember that the PHP process or user running the script must have read privileges for the log file.

We will not walk through the parsing class in any more details than what is given in the phpdoc comments in the source below

```
<?php
/**
 * Class ParseProcmailLogFile
 */
class ParseProcmailLogFile {
 private $iFileName='';
 /**
 * Constructor for the parse class
 *
 * @param mixed $aFileName Log file to read
 * @return ParseProcmailLogFile
 */
 function __construct($aFileName) {
 $this->iFileName = $aFileName ;
 }
 /**
 * Get line without trailing "\n"
 *
 * @param mixed $fp Filepointer
 * @return string Read line without trailing "\n"
 */
 function GetLine($fp) {
 $s = fgets($fp);
 return substr($s,0,strlen($s)-1);
 }
 /**
 * Get statistics from the parsed log file
 *
 * @param $aWindow Window size. How many days to include in the returned stats
 * @return array with (date, number of killed spam, number of non killed spams)
 */
 function GetStat($aWindow) {

 $fp = fopen($this->iFileName,'r');
 if($fp === false) {
 JpGraphError::Raise('Cannot read log file');
 }

 // Now loop through the file. We always keep the last 3 lines in
 // the buffer to be able to get the context of a line since the
 // folder is stored on one line and the date of the main on the
 // previous line
 $buf[1] = $this->GetLine($fp);
 $buf[2] = $this->GetLine($fp);

 $idx = 0;
```

```
$idx2 = 0;
$found = array(); // All /dev/null spam headers
$found2 = array(); // All Spam folder headers

// Loop through all lines in the file and store the found emails
// in the two $found arrays
while(! feof($fp)) {

 //Shift buffer one step
 $buf[0] = $buf[1];
 $buf[1] = $buf[2];
 $buf[2] = $this->GetLine($fp);

 // Find /dev/null entries
 if(strpos($buf[2],'Folder: /dev/null') !== false) {
 if(strpos($buf[0],'From ') !== false) {
 $datepos = 0 ;
 }
 elseif(strpos($buf[1],'From ') !== false) {
 $datepos = 1 ;
 }
 else {
 continue;
 }
 // Aggregate all the data per day
 $date = strtotime(date('D j M Y'),strtotime(substr($buf[$datepos],-
$found[$idx++] = array(str_replace(' Subject: ','',$buf[1]),$date)
}

// Find spam folder entries
if(strpos($buf[2],'.Spam') !== false) {
 if(strpos($buf[0],'From ') !== false) {
 $datepos = 0 ;
 }
 elseif(strpos($buf[1],'From ') !== false) {
 $datepos = 1 ;
 }
 else {
 continue;
 }
 // Aggregate all the data per day
 $date = strtotime(date('D j M Y'),strtotime(substr($buf[$datepos],-
$found2[$idx2++] = array(str_replace(' Subject: ','',$buf[1]),$dat
}

fclose($fp);

// Find out how many at each day of dev null
$date = $found[0][1];
$daystat[$date] = 0;

for($i=0; $i < $idx; ++$i) {
```

```
 if($date == $found[$i][1]) {
 ++$daystat[$date];
 }
 else {
 $date = $found[$i][1];
 $daystat[$date] = 1;
 }
 }

 // Find out how many at each day of spam
 $daystat2 = array();
 if(count($found2) > 0) {
 $date = $found2[0][1];
 $daystat2[$date] = 0;

 for($i=0; $i < $idx2; ++$i) {
 if($date == $found2[$i][1]) {
 ++$daystat2[$date];
 }
 else {
 $date = $found2[$i][1];
 $daystat2[$date] = 1;
 }
 }
 }

 // Now make sure that both arrays have the same
 // number of entries.
 foreach($daystat as $key => $val) {
 if(!isset($daystat2[$key])) {
 $daystat2[$key] = 0;
 }
 }

 foreach($daystat2 as $key => $val) {
 if(!isset($daystat[$key])) {
 $daystat[$key] = 0;
 }
 }

 // Window and return the data
 $n = count($daystat);
 $start = $n > $aWindow ? $n - $aWindow : 0;
 $date_keys = array_keys($daystat);
 $idx=0;

 $datax = array(); $datay1 = array(); $datay2 = array();
 for($i=$start; $i < $n; ++$i) {
 $datax[$idx] = date('D j M', $date_keys[$i]);
 $datay1[$idx] = $daystat[$date_keys[$i]];
 $datay2[$idx++] = $daystat2[$date_keys[$i]];
 }

 return array($datax,$datay1,$datay2);
```

```
 }
}
```

### 30.3. Step 2: Creating the graph

We will use a basic accumulated bar graphs with a text scale and integer y-scale. The remainder of the script should be fairly self explanatory since this is a straightforward graph with no "tricks".

```
<?php
/**
 * Class SpamGraph
 */
class SpamGraph {
 private $iWidth, $iHeight;
 /**
 * Constructor
 *
 * @param $aWidth Width of generated graph
 * @param $aHeight Height of generated graph
 * @return SpamGraph
 */
 public function __construct($aWidth,$aHeight) {
 $this->iWidth = $aWidth;
 $this->iHeight = $aHeight;
 }
 /**
 * Create an accumulated bar graph
 *
 * @param string $aFile File to write graph to
 * @param array $aXData Date x-labels
 * @param array $aYData Spam data 1 (/dev/null)
 * @param array $aY2Data Spam data 2 (Spam folder)
 */
 public function Create($aFile,$aXData,$aYData,$aY2Data) {

 $graph = new Graph($this->iWidth,$this->iHeight);
 $graph->SetMargin(40,20,50,110);
 $graph->SetScale('textint');
 $graph->SetMarginColor('khaki2@0.6');

 $graph->legend->SetPos(0.5,0.97,'center','bottom');
 $graph->legend->SetLayout(10);
 $graph->legend->SetFillColor('white');
 $graph->legend->SetFont(FF_ARIAL,FS_NORMAL,10);

 $graph->title->Set('Identified spams');
 $graph->title->SetMargin(10);
 $graph->title->SetFont(FF_ARIAL,FS_NORMAL,14);

 $graph->subtitle->Set('Updated: '.date('j M Y, H:i'. ' T')));
 $graph->subtitle->SetMargin(5);
 $graph->subtitle->SetFont(FF_ARIAL,FS_ITALIC,11);
```

```

$graph->xaxis->SetTickLabels($aXData);
$graph->xaxis->SetFont(FF_ARIAL,FS_NORMAL,8);
$graph->xaxis->SetLabelAngle(45);

$graph->yaxis->SetFont(FF_ARIAL,FS_NORMAL,10);
$graph->yscale->SetGrace(10);

$bar1 = new BarPlot($aYData);
$bar1->SetFillGradient('darkred:1.2','darkred:0.7',GRAD_VERT);
$bar1->SetLegend('To "/dev/null"');

$bar2 = new BarPlot($aY2Data);
$bar2->SetFillGradient('orange','orange:0.7',GRAD_VERT);
$bar2->SetLegend('To Spam folder');

$abar = new AccBarPlot(array($bar1,$bar2));
$abar->value->Show();
$abar->value->SetFormat('%d');
$graph->Add($abar);
$graph->Stroke($aFile);

syslog(LOG_INFO,"Created spam image in $aFile.");
}
}

?>

```

## 30.4. Step 3: Uploading the image file to a server

This class assumes that PHP has been built with support for FTP. Since the script is straightforward we do not discuss it any more in details other than noting that we use the default system logger to store information that we have uploaded the file successfully. This is done via the PHP method **syslog()**.

```

<?php
/**
 * Class FTPUploader
 */
class FTPUploader {
 private $iserver='', $iuid='', $ipwd='';
 /**
 * Create new instance of the FTP uploader class
 *
 * @param $aServer The URI for the server
 * @param $aUID The ftp user id
 * @param $aPWD The ftp user password
 * @return FTPUploader
 */
 function __construct($aServer,$aUID,$aPWD) {
 $this->iserver = $aServer;
 $this->iuid = $aUID;
 }
}

```

```

 $this->ipwd = $aPWD;
 }
/**/
 * Upload the specified file to the given directory on the server
 *
 * @param $aFile Name and path of file to uploads
 * @param $aUploadDir The directory on the server where the file should be stored
 */
function Upload($aFile,$aUploadDir) {
 $conn_id = @ftp_connect($this->iserver);

 if (!$conn_id) {
 JpGraphError::Raise("FTP connection failed.\nAttempted to connect to {$aUploadDir}");
 }

 $login_result = ftp_login($conn_id, $this->iuid, $this->ipwd);
 if (((!$conn_id) || (!$login_result)) {
 JpGraphError::Raise("FTP login has failed.\nAttempted to connect to {$aUploadDir}");
 }

 syslog(LOG_INFO,JpGraphError::GetTitle()."Connected to {$this->iserver}");

 // Delete potential old file
 $ftp_file = $aUploadDir.basename($aFile);
 $res = @ftp_delete($conn_id,$ftp_file);

 // Upload new image
 $upload = ftp_put($conn_id, $ftp_file, $aFile, FTP_BINARY);
 if (!$upload) {
 JpGraphError::Raise("FTP upload of image failed.");
 }

 syslog(LOG_INFO,JpGraphError::GetTitle()."Successfully uploaded $aFile to {$aUploadDir}");

 @ftp_close($conn_id);
}
?>

```

## 30.5. Step 0: The full driver script

For completeness we also include the exact main script we use

```

#!/usr/bin/php
<?php
require_once 'jpgraph/jpgraph.php';
require_once 'jpgraph/jpgraph_bar.php';
require_once 'jpgraph/jpgraph_line.php';

DEFINE('FTP_SERVER','aditus.nu');
DEFINE('FTP_UID','aditus');
DEFINE('FTP_PWD','Psion3a');

```

```
DEFINE('PROCMAIL_LOGFILE' , '/tmp/procmail-ljp-200907.log');
DEFINE('WINDOWSIZE',14); // nbr days window
DEFINE('IMGFILE','/tmp/spamstat.png');

DEFINE('FTP_DIR' , 'www/jpgraph/img/');

// Don't use image error handler, use syslog
JpGraphError::SetImageFlag(false);
JpGraphError::SetLogFile('syslog');

/**
 * Class ParseProcmailLogFile
 */
class ParseProcmailLogFile {
 private $iFileName='';
 /**
 * Constructor for the parse class
 *
 * @param mixed $aFileName Log file to read
 * @return ParseProcmailLogFile
 */
 function __construct($aFileName) {
 $this->iFileName = $aFileName ;
 }
 /**
 * Get line without trailing "\n"
 *
 * @param mixed $fp Filepointer
 * @return string Read line without trailing "\n"
 */
 function GetLine($fp) {
 $s = fgets($fp);
 return substr($s,0,strlen($s)-1);
 }
 /**
 * Get statistics from the parsed log file
 *
 * @param $aWindow Window size. How many days to include in the returned stats
 * @return array with (date, number of killed spam, number of non killed spams)
 */
 function GetStat($aWindow) {

 $fp = fopen($this->iFileName,'r');
 if($fp === false) {
 JpGraphError::Raise('Cannot read log file');
 }

 // Now loop through the file. We always keep the last 3 lines in
 // the buffer to be able to get the context of a line since the
 // folder is stored on one line and the date of the main on the
 // previous line
 $buf[1] = $this->GetLine($fp);
 $buf[2] = $this->GetLine($fp);
```

```
$idx = 0;
$idx2 = 0;
$found = array(); // All /dev/null spam headers
$found2 = array(); // All Spam folder headers

// Loop through all lines in the file and store the found emails
// in the two $found arrays
while(! feof($fp)) {

 //Shift buffer one step
 $buf[0] = $buf[1];
 $buf[1] = $buf[2];
 $buf[2] = $this->GetLine($fp);

 // Find /dev/null entries
 if(strpos($buf[2],'Folder: /dev/null') !== false) {
 if(strpos($buf[0],'From ') !== false) {
 $datepos = 0 ;
 }
 elseif(strpos($buf[1],'From ') !== false) {
 $datepos = 1 ;
 }
 else {
 continue;
 }
 $date = strtotime(date('D j M Y'),strtotime(substr($buf[$datepos],-
$found[$idx++] = array(str_replace(' Subject: ','',$buf[1]),$date)
}

// Find spam folder entries
if(strpos($buf[2],'.Spam') !== false) {
 if(strpos($buf[0],'From ') !== false) {
 $datepos = 0 ;
 }
 elseif(strpos($buf[1],'From ') !== false) {
 $datepos = 1 ;
 }
 else {
 continue;
 }
 $date = strtotime(date('D j M Y'),strtotime(substr($buf[$datepos],-
$found2[$idx2++] = array(str_replace(' Subject: ','',$buf[1]),$dat
}

fclose($fp);

// Find out how many at each day of dev null
$date = $found[0][1];
$daystat[$date] = 0;

for($i=0; $i < $idx; ++$i) {
 if($date == $found[$i][1]) {
```

```
 ++$daystat[$date];
 }
 else {
 $date = $found[$i][1];
 $daystat[$date] = 1;
 }
}

// Find out how many at each day of spam
$daystat2 = array();
if(count($found2) > 0) {
 $date = $found2[0][1];
 $daystat2[$date] = 0;

 for($i=0; $i < $idx2; ++$i) {
 if($date == $found2[$i][1]) {
 ++$daystat2[$date];
 }
 else {
 $date = $found2[$i][1];
 $daystat2[$date] = 1;
 }
 }
}

// Now make sure that both arrays have the same
// number of entries.
foreach($daystat as $key => $val) {
 if(!isset($daystat2[$key])) {
 $daystat2[$key] = 0;
 }
}

foreach($daystat2 as $key => $val) {
 if(!isset($daystat[$key])) {
 $daystat[$key] = 0;
 }
}

// Window and return the data
$n = count($daystat);
$start = $n > $aWindow ? $n - $aWindow : 0;
$date_keys = array_keys($daystat);
$idx=0;

$datax = array(); $datay1 = array(); $datay2 = array();
for($i=$start; $i < $n; ++$i) {
 $datax[$idx] = date('D j M', $date_keys[$i]);
 $datay1[$idx] = $daystat[$date_keys[$i]];
 $datay2[$idx++] = $daystat2[$date_keys[$i]];
}

return array($datax,$datay1,$datay2);
}
```

```

}

/**
 * Class SpamGraph
 */
class SpamGraph {
 /**
 * Create an accumulated bar graph
 *
 * @param mixed $aFile File to write graph to
 * @param mixed $aXData Date x-labels
 * @param mixed $aYData Spam data 1 (/dev/null)
 * @param mixed $aY2Data Spam data 2 (Spam folder)
 */
 static function Create($aFile,$aXData,$aYData,$aY2Data) {

 $graph = new Graph(650,420);
 $graph->SetMargin(40,20,50,110);
 $graph->SetScale('textint');
 $graph->SetMarginColor('khaki2@0.6');

 $graph->legend->SetPos(0.5,0.97,'center','bottom');
 $graph->legend->SetLayout(10);
 $graph->legend->SetFillColor('white');
 $graph->legend->SetFont(FF_ARIAL,FS_NORMAL,10);

 $graph->title->Set('Identified spams');
 $graph->title->SetMargin(10);
 $graph->title->SetFont(FF_ARIAL,FS_NORMAL,14);

 $graph->subtitle->Set('Updated: '.date('j M Y, H:i'. ' T')));
 $graph->subtitle->SetMargin(5);
 $graph->subtitle->SetFont(FF_ARIAL,FS_ITALIC,11);

 $graph->xaxis->SetTickLabels($aXData);
 $graph->xaxis->SetFont(FF_ARIAL,FS_NORMAL,8);
 $graph->xaxis->SetLabelAngle(45);

 $graph->yaxis->SetFont(FF_ARIAL,FS_NORMAL,10);
 $graph->yscale->SetGrace(10);

 $bar1 = new BarPlot($aYData);
 $bar1->SetFillGradient('darkred:1.2','darkred:0.7',GRAD_VERT);
 $bar1->SetLegend('To "/dev/null"');

 $bar2 = new BarPlot($aY2Data);
 $bar2->SetFillGradient('orange','orange:0.7',GRAD_VERT);
 $bar2->SetLegend('To Spam folder');

 $abar = new AccBarPlot(array($bar1,$bar2));
 $abar->value->Show();
 $abar->value->SetFormat('%d');
 $graph->Add($abar);
 $graph->Stroke($aFile);
 syslog(LOG_INFO,"Created spam image in $aFile.");
 }
}

```

```

 }
 }
/***
 * Class FTP
 */
class FTP {
 private $iserver='', $iuid='', $ipwd='';
 function __construct($aServer,$aUID,$aPWD) {
 $this->iserver = $aServer;
 $this->iuid = $aUID;
 $this->ipwd = $aPWD;
 }
 function Upload($aFile,$aUploadDir) {
 $conn_id = @ftp_connect($this->iserver);

 if (!$conn_id) {
 JpGraphError::Raise("FTP connection failed.\nAttempted to connect to {$this->iserver}");
 } else {
 syslog(LOG_INFO,"Connected to {$this->iserver}");
 }
 $login_result = ftp_login($conn_id, $this->iuid, $this->ipwd);
 if ((!$conn_id) || (!$login_result)) {
 JpGraphError::Raise("FTP login has failed.\nAttempted to connect to {$this->iserver}");
 } else {
 syslog(LOG_INFO,"Logged in to {$this->iserver}");
 }

 // Delete potential old file
 $ftp_file = $aUploadDir.basename($aFile);
 $res = @ftp_delete($conn_id,$ftp_file);
 syslog(LOG_INFO,"Uploading image as $aFile");

 // Upload new image
 $upload = ftp_put($conn_id, $ftp_file, $aFile, FTP_BINARY);
 if (!$upload) {
 JpGraphError::Raise("FTP upload of image failed.");
 } else {
 syslog(LOG_INFO,"Successfully uploaded $aFile to {$this->iserver}.");
 }

 @ftp_close($conn_id);
 }
}

// Step 1) Get the statistics
$parser = new ParseProcmailLogFile(PROCMAIL_LOGFILE);
list($xdata, $ydata, $y2data) = $parser->GetStat(WINDOWSIZE);

// Step 2) Create the graph
SpamGraph::Create(IMGFILE,$xdata,$ydata,$y2data);

// Step 3) Upload the file if defined
if(IMGFILE !== '') {

```

```
$ftp = new FTP(FTP_SERVER,FTP_UID,FTP_PWD);
$ftp->Upload(IMGFILE,FTP_DIR);
}
?>
```

---

# **Chapter 31. Creating Critical chain buffer penetration charts**

## **31.1. Introduction and purpose**

Critical chain is one (of many) suggested principles for project management that actually makes some sense in the meaning that it tries to take into account both the human aspect of making time estimates (it is difficult) as well as the different constraints put upon (for example) a SW project. Such constraints are typically that there are only two persons with the knowledge to do task X or that task Y has never been done before and therefore is almost impossible to estimate correctly.

Since this manual has no intention to serve as an introduction to full critical chain project management (CCPM) we will not dwell on the finer details instead we will take out one particular part, or rather a tool, that is one of the fundamental ways of keeping track of a projects using CCPM. The buffer penetration chart.

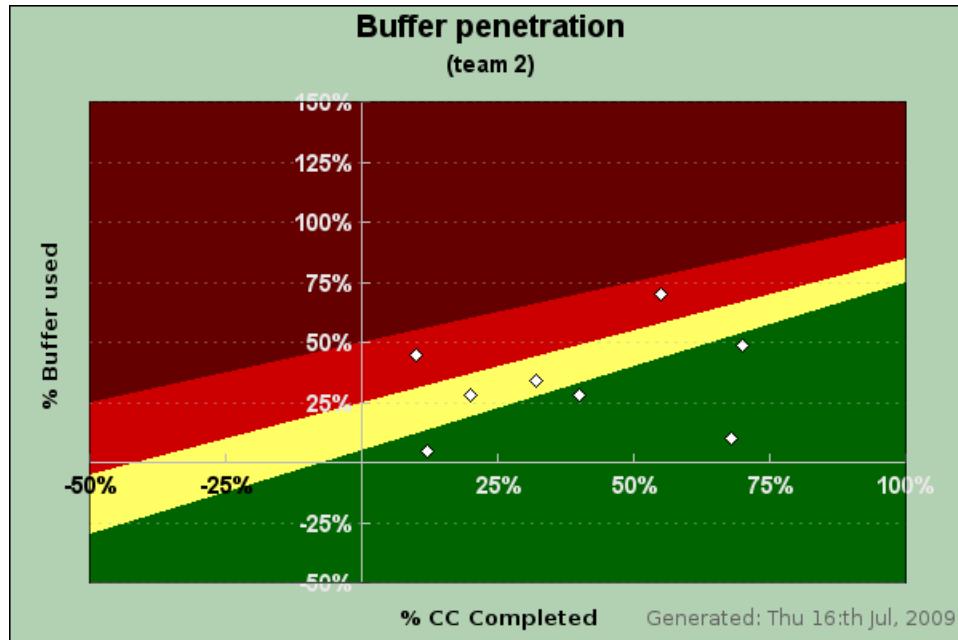
Even though CCPM in general might not be of interest to the reader the particular graph we will create is a good way to show how to think "out-of-the-box" in creating some graphs with the this library. So it is fully possible to read this case without understanding the underlying methodology.

The graph we will create makes use of ( among other things)

- Filled area graphs combined with scatter plots
- Showing how to modify the default fill behavior (from the x-axis)
- Showing how to disable the 0-labels
- Adjusting the display depth for grid lines (and changing the style of grid lines)
- Using different colors on individual scale labels
- How to create a custom graph class that can be reused

Figure 31.1, “Critical chain buffer penetration. Each white scatter dot represents the state of one task.” shows a typical example of what we will accomplish by the end of this case study

**Figure 31.1. Critical chain buffer penetration. Each white scatter dot represents the state of one task.**



### 31.1.1. The principle behind buffer penetration charts

While not strictly necessary in order to create the graph the following crash course on buffer penetration might help see the usefulness of these types of charts in large projects.

#### Note

Those wanting to know more about Critical Chain planning are referred to the book and online references at the end of this chapter, see [1] Critical chain Project Management, 2ed, Lawrence P. Leach and [2] A Guide to Implementing the Theory of Constraints, K. J. Youngman, .

To understand the principles of this graph we need to shortly discuss how task (time) estimation is done using the CC methodology. Time estimation for a task is divided in two parts; 1) the optimum time and 2) the contingency buffer. Together they will make up the allocated time for a task. On average each task is expected to make use of some of the contingency buffer (typically as much as 50-75%).

This might seem very strange for people with experience with other methodologies where use of contingency time is an indication of some kind of failure. However, that is not the case in CCPM. The usage of (some) of the allocated buffers are expected. This comes from the way the buffer and the "optimum" time are estimated. The optimum time is a 50% estimate, meaning that the task is only expected to be able to go that fast in 50% of all the times the task is performed. Hence a very optimistic estimate. With the contingency buffer added the estimate should correspond to a 90% estimate, i.e. in 90% of the times the task is performed it will finish within this time.

The graph in Figure 31.1, “Critical chain buffer penetration. Each white scatter dot represents the state of one task.” is a way to visualise the status of an entire project in terms of buffer penetration. The x-axis shows how much of a particular task is completed and the y-axis shows how much buffer up to now have been used. Another way of viewing this is to show how far off the optimum 50% time estimate for the task we are in practice.

The different colored background is a "health" monitor for the tasks. As long as a task is in the green area that task is not in the focus for corrective actions. However as the task moves from green through yellow and into the red this is a signal that immediate actions are needed to secure the end delivery time. If a task has gone into the dark red (or brown) area this is an indication that corrective actions are probably not going to help and a re-planning of the task (and potentially the project is needed). Hence we do not want any tasks in the brown area!

The exact limits for what is considered "green", "yellow" and "red" varies depending on the context and to some degree the flexibility and size of the team but the default values shown above represents a fairly average limits that have been shown to be useful indicators across several industries.

The final key to why this works fairly well in practice is that each team member only have to report one figure in order to track the progress he or she is making and that is how much time more he or she will need to complete the task. Since we now at what day the team member made the estimate we have all the information we need in order to update the plans to see how well we are tracking the original plan.

With the CC methodology we do not bother looking in the mirror and asking the team member to estimate how much of the task he or she has done because that is really irrelevant. The only key figure we need to complete the project on time is to how much more effort/time is needed from the team member. How much of the task has been completed can easily be calculated by knowing how much time is left and compare that with the original estimate. For example, if the original estimate was 12 days and the team member at a particular day estimates that he/she has 10 days left we can say that we have completed  $(12-10)/12 \sim 17\%$  of the task.

This also explains how it can come that we could get negative completion. This is just an indication that the original estimate was too low. For example if the task was originally estimated to 12 days and at a particular day the team member estimates that he/she will need at least 16 more days to complete the task the completion % would be  $(12-16)/12 \sim -33\%$

An example will illustrate how this works.

### Example 31.1. Buffer penetration example

Assume we have one task that has a 50% estimate of 6 days and a buffer of 5 days (indicating the volatility in the 50% estimate). The following table shows how much time left the team member estimates he has at the end of each working day. This single number allows us to compute (in relation to the original estimate) how much of the task is completed and how much buffer the member has used.

Day	Est. time left (days)	Completion	Buffer penetration (days)	Buffer penetration (%)
		%		
1	5	0	0	0
2	5	(6-5)/6 ~ 17%	1	1/5 = 20%
3	5	(6-5)/6 ~ 17%	2	2/5 = 40%
4	5	(6-5)/6 ~ 17%	3	3/5 = 60%
5	4	(6-4)/6 ~ 33%	3	3/5 = 60%
6	3	(6-3)/6 ~ 50%	3	3/5 = 60%
7	2	(6-2)/6 ~ 67%	3	3/5 = 60%
8	1	(6-1)/6 ~ 83 %	3	3/5 = 60%
9	0	(6-0)/6 = 100%	3	3/5 = 60%

The following three (small) buffer penetration diagram shows how the task at end of days 2,5 and 8

**Figure 31.2. Buffer penetration chart for example**

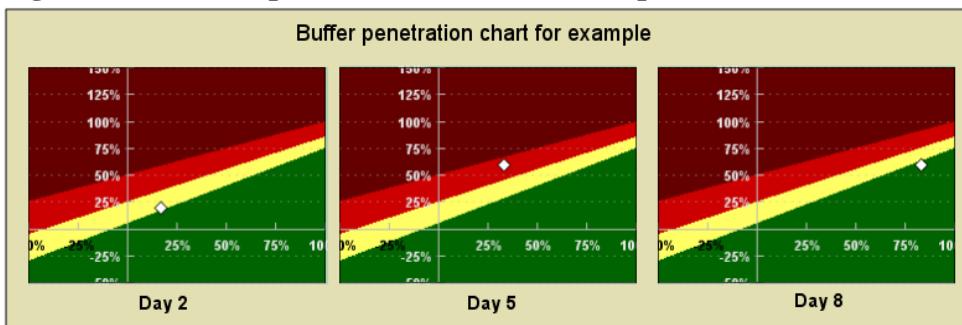
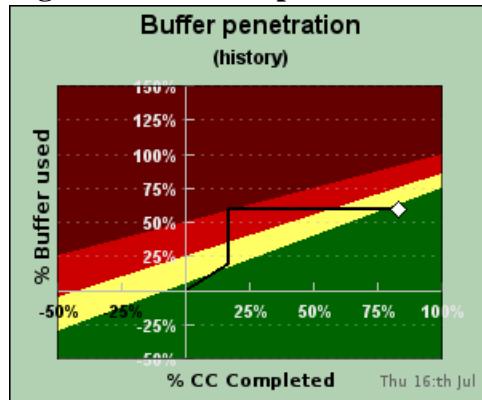


Figure 31.2, “Buffer penetration chart for example” shows a fairly typical (albeit not ideal) progress for a task. It gets a rocky start, has some problems halfway through and then manages to recover towards the end of the execution.

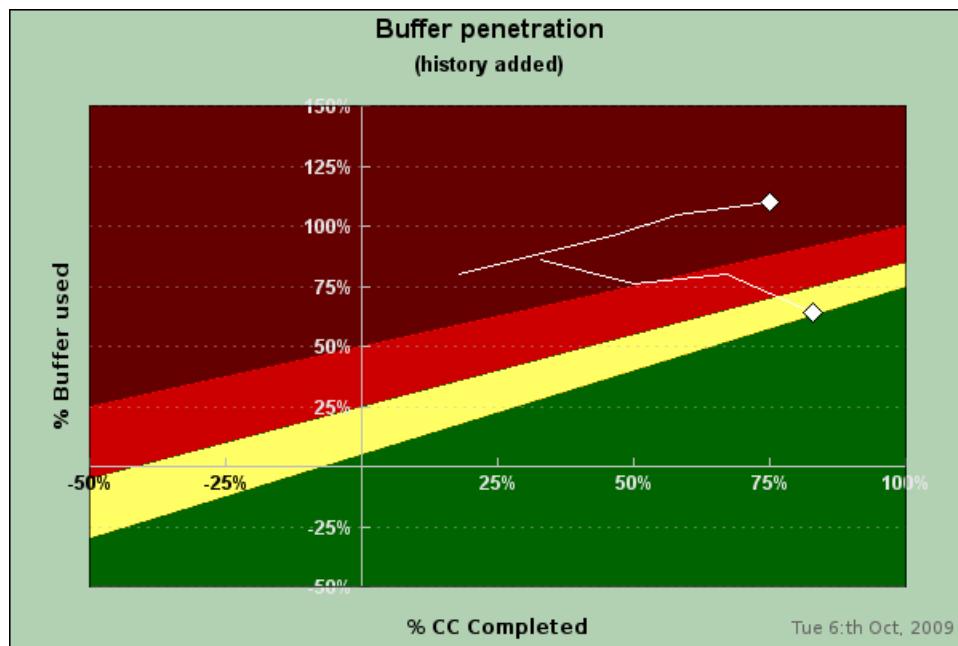
A common extension to the graph above is to also add a "historic" tail to the scatter point to show how it historically has moved. Adding a historic "tail" at day 8 would give the following penetration chart

**Figure 31.3. Buffer penetration chart with "historic" tail**



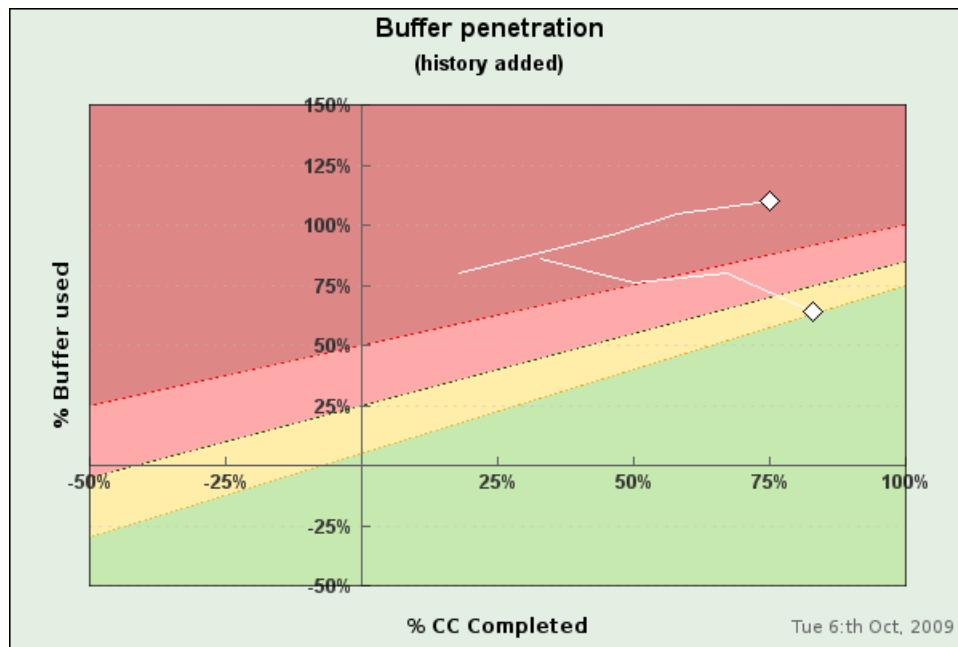
For an entire project these kind of "falling star" traces can be quite effective ways to see how tasks are progressing. We end this chapter with a final complete example (which uses the code we will develop in the following sections) to create a buffer chart for two tasks.

**Figure 31.4. Complete buffer penetration example with history trace (ccbp\_ex1.php) [example\_src/ccbp\_ex1.html]**



We can also use the alternative color map for this and get the result shown in Figure 31.5, “Using the alternative color map (ccbp\_ex2.php)”

**Figure 31.5. Using the alternative color map (ccbp\_ex2.php) [example\_src/ccbp\_ex2.html]**



The next section will explain in some details how to create a class that can mae these types of charts with an easy to use interface.

## 31.2. Creating a utility class to construct CC BP charts

We will start from a high level and think about how we would like to be able to use such a utility class. Ideally it would be similar to other types of graphs classes. To keep things simple we want the following parameter to be adjustable

- The min and max x- and y-axis scale values to be displayed in the graph
- The limits for the color indications, i.e. exactly what is the green, yellow, red and brown areas should be customizable. In oder to keep the interface simple we assume that the areas are bounded by straight lines and that we want to be able to specify the areas by stating the value at x=0 and the value at x=maximum x-scale

So for example the standard green area is limited by the specification (5,75) meaning that the green line crosses y=5 at x=0 and y=75 at x=100

- We would also like to be ale to adjust the colors used as indicators
- We want to be able to add both scatter marks and lines (to show history)
- We also need to be able to specify the title and the sub title
- The rest of the graph options and setting we would like the utility class to handle for us. As a bonus it would be nice if the class adjusted the size of the fonts to the overall size we have specified for the graph automatically to make it look visually proportional.

When doing designs like this it is always good to first write a small test driver without having to think about how to actually implement it. This keeps an end user perspective on things. Later on we can do changes if it turns out that a particular simplification for the user is simply too complex to implement.

So, lets think about how we (ideally) would like to be able to create the chart shown in Figure 31.3, “Buffer penetration chart with “historic” tail”

```
<?php
$graph = new CCBPGraph(300,250);
$graph->SetTitle('Buffer penetration','(history)');

$datax=array(83); $datay=array(60);
$datax_history = array(0,17,17,17,33,50,67,83);
$datay_history = array(0,20,40,60,60,60,60,60);

$sp = new ScatterPlot($datay,$datax);
$sp->mark->SetType(MARK_DIAMOND);
$sp->mark->SetFillColor('white');
$sp->mark->SetSize(10);

$sp_hist = new LinePlot($datay2,$datax2);
$sp_hist->SetWeight(2);
$sp_hist->SetColor('black');
```

```
$graph->Add($sp_hist);
$graph->Add($sp);

$graph->Stroke();
?>
```

We cannot hope to have the class make it any simpler than this. This is as close to a "normal" line graph we can get (apart from the color indications in the background). So now its just a matter on designing this class.

Since the utility class will be a rather restricted graph class it with a very specific usage it doesn't make sense to implement it as an extension to the normal graph. Instead we will create contain class that contains a graph that we can modify and setup. (In UML language a *has-a* relation.)

We will name our utility class CCBPGraph (Short for *Critical chain Buffer Penetration Graph*). The class should have the following signature

```
<?php
class CCBPGraph{
 private $graph = null ; // The real graph instance

 // The "normal" constructor
 public function __construct($aWidth,$aHeight) {}

 // Methods that will control the appearance
 public function SetTitle($aTitle,$aSubTitle) {}
 public function SetXMinMax($aMin,$aMax) {}
 public function SetYMinMax($aMin,$aMax) {}
 public function SetColorIndication($aSpec,$aColors=null) {}
 public function SetColorMap($aMap) {}

 // Internal function to create and setup the graph
 private function Init() {}

 // Mimix the standard graph functions Add() and Stroke()
 public function Add($aPlots) {}
 public function Stroke($aFile='') {}
}
```

In addition the class must have a number of instance variables that are used to store the settings until we need them when we create the graph. However to keep things simple here we do not list them above. Lets now walk through the four groups of methods.

## 1. The constructor

```
public function __construct($aWidth,$aHeight) {}
```

The constructor doesn't need to do very much. It just needs to store the user specified width and height so we get them back when we create the actual graph. We could also make use of the constructor to create an instance of the real graph class but we prefer to do as late as possible instantiation.

## 2. Controlling the appearance of the graph.

Again these methods will be very basic "setter" methods that will just store the user specified options until we need them when we construct the graph

```
public function SetTitle($aTitle,$aSubTitle) {}
public function SetXMinMax($aMin,$aMax) {}
public function SetYMinMax($aMin,$aMax) {}
public function SetColorIndication($aSpec,$aColors=null) {}
public function SetColorMap($aMap) {}
```

For the color setter we should also add some basic error checking so that the parameter makes sense.

### 3. The real worker method

```
private function Init() {}
```

Since this is the method is where we will do the actual work and will be fairly large we will spend the entire next section on this method.

### 4. The "fake" standard methods. Add() and Stroke(). Since these are fairly short we will show them here in there entirety

```
<?php
function Add($aPlots) {
 if(is_array($aPlots)) {
 $this->iPlots = array_merge($this->iPlots,$aPlots);
 }
 else {
 $this->iPlots[] = $aPlots;
 }
}

function Stroke($aFile='') {
 $this->Init();
 $this->graph->Add($this->iPlots);
 $this->graph->Stroke($aFile);
}
?>
```

When adding plots (with the `Add()` method) we have to take into account that the parameter can be either a single object or an array of plot objects (line and scatter plots) and we must handle that accordingly as is done above.

For the `Stroke()` method we use this to call the initialization method (to actually create the graph) and then add the user specified plots and finally call the real `Graph::Stroke()` of the graph.

These methods are very simple but from the outside it will look like a "real" graph class with a clean and simple interface.

## 31.3. The `Init()` method

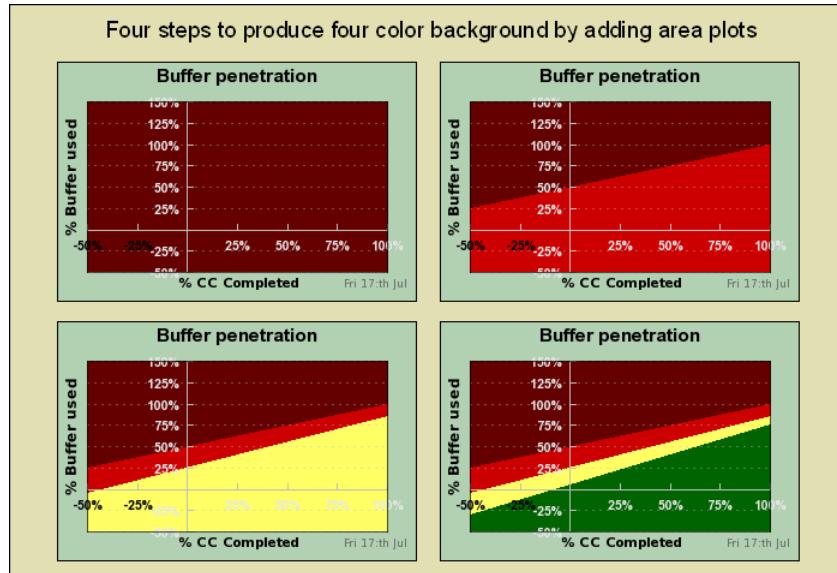
This is the real method that actually constructs the graph. While we will not go through every single line in details we will highlight the complications and some formatting options we have used.

### 31.3.1. Creating the colored backgrounds

In order to create the colored background we create filled area plots and add them to the graph. Starting with the "brown" and successively adding the rest to create the colored band effect we want. Figure 31.6,

“Steps to create the background” shows in “slow-motion” how this is done by adding four area plots, one at a time.

**Figure 31.6. Steps to create the background**



The exact position for the lines are calculated with the positions given for each color band. The position for each color band is specified by giving the y-coordinate at  $x=0$  and the y-coordinate at the maximum x-scale value.

When adding the area plots there is one thing we have to modify. By default the fill is done between the line and the  $y=0$  line. In our case we need the fill to go all the way down to the min y-value. To change this behavior we need to call the method

- `LinePlot::SetFillFromYMin()`

for each of the areas.

### 31.3.2. Getting the grid lines right

Since we want some discrete horizontal grid lines we might think that it is enough to do the normal

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

However, doing that will not show any grid lines. The reason is that by default the grid lines are drawn at the bottom of the plot. Since we have filled area plots covering 100% of the plot area no grid lines would show. In order to change this we need to move the grid line to the front with a call to the method

- `Graph::SetGridDepth($aDepth)`

using the argument `DEPTH_FRONT`. The rest of the grid line formatting is just basic style and color modification to make the grid visible but just barely.

### 31.3.3. Getting the scale and labels right

For this type of graph we have manually set the distance between each tick label to 25 units. This would put labels as 0,25,50, and so on. The maximum value (the user specifies) will be adjusted so that it is always an even multiple of 25 to allow the last tick mark to be at the end of the axis.

As can be seen from the previous images we are using one feature that hasn't been previously exemplified and that is the possibility to have unique colors on each label on the scale. We use this for the x-scale by having the negative labels in black and the positive labels in white. The reason is purely functional to allow the scale labels to be more easy to read against the colored background.

The color of the labels are specified as the second argument to

- `Axis::SetColor($aAxisColor,$aLabelColor)`

In addition we have also hidden the zero labels since they would just be disturbing in the middle and doesn't really add any information we don't already have.

Finally the labels are formatted to show a percentage sign after each label. This is done by a format string

### 31.3.4. Adjusting the margin and text sizes

In the beginning of the `Init()` method the margins for the graph is adjusted depending on the actual size the user specified. The same goes with establishing the basic font size used for the scale labels as well as the titles (both graph and axis). The size is just based on heuristics on what (in our view) gives a well balanced graph.

## 31.4. Suggested improvements

As means for further individual exploration the `CCBPGraph` class could be extended by adding (for example) the following features. (There is always a balance to be had between adding a lot of feature and still keeping the interface simple enough to be manageable)

1. Add customization for the titles of the two axis
2. Make it possible to adjust the margin color
3. Make the number of background color bands user adjustable
4. Add data manipulation so that the input data is specified as a number of historic values for each task. The latest value should be displayed as a mark and the last  $n$  values (user selectable window size) shown as a trailing path behind the mark to see how it has moved as shown in for example Figure 31.4, "Complete buffer penetration example with history trace (`ccbp_ex1.php`)"
5. Make use of the `MGraph` class to make it possible to create multiple CCBP charts in the same image. In order to identify which graph is affected by commands to it would be possible to just add a new argument giving the index to all the existing methods. However, a cleaner implementation would probably use the concept of a "current" graph which receives all the method calls. Implementing it that way would allow us to keep the existing signatures for all existing methods. In order to position the graphs it is probably a good idea to make use of some easy way for the user just to select how many rows should be used and many charts should be displayed on each row (to avoid the end user from having to position the graphs pixel by pixel)

## 31.5. The implementation of class `CCBPGraph`

The entire implementation is included below and is also available en the "Example/" directory of the distribution as "`ccbpgraph.class.php`". There are also two example scripts using this utility class "`ccbp_ex1.php`" and "`ccbp_ex2.php`"

```

$df = 'D j:S M, Y';
if($width < 400) {
 $df = 'D j:S M';
}

$time = new Text(date($df),$width-10,$height-10);
$time->SetAlign('right','bottom');
$time->SetFont(FF_VERA,FS_NORMAL,$labelsize-1);
$time->SetColor('darkgray');
$graph->Add($time);

// Use an accumulated fille line graph to create the colored bands

$n = 3;
for($i=0; $i < $n; ++$i) {
 $b = $this->iColorInd[$i][0];
 $k = ($this->iColorInd[$i][1] - $this->iColorInd[$i][0])/$this->iXMax;
 $colarea[$i] = array(array($lowx,$lowx*$k+$b), array($highx,$highx*$k)
}
$colarea[3] = array(array($lowx,$highy), array($highx,$highy));

$cb = array();
for($i=0; $i < 4; ++$i) {
 $cb[$i] = new LinePlot(array($colarea[$i][0][1],$colarea[$i][1][1]),
 array($colarea[$i][0][0],$colarea[$i][1][0]));
 $cb[$i]->SetFillColor($this->iColorSpec[$this->iColorMap][$i]);
 $cb[$i]->SetFillFromYMin();
}

$graph->Add(array_slice(array_reverse($cb),0,4));
$this->graph = $graph;
}

/**
 * Add a line or scatter plot to the graph
 *
 * @param mixed $aPlots
 */
public function Add($aPlots) {
 if(is_array($aPlots)) {
 $this->iPlots = array_merge($this->iPlots,$aPlots);
 }
 else {
 $this->iPlots[] = $aPlots;
 }
}
/**
 * Stroke the graph back to the client or to a file
 *
 * @param mixed $aFile
 */
public function Stroke($aFile='') {
$this->Init();
 if(count($this->iPlots) > 0) {
 $this->graph->Add($this->iPlots);
 }
 $this->graph->Stroke($aFile);
}
}
?>

```

## 31.6. References

- [1] Critical chain Project Management, 2ed, Lawrence P. Leach Artech House, ISBN 1-58053-903-3
- [2] A Guide to Implementing the Theory of Constraints, K. J. Youngman, [http://www.dbrmfg.co.nz/Projects\\_Buffers.htm](http://www.dbrmfg.co.nz/Projects_Buffers.htm) [http://www.dbrmfg.co.nz/Projects%20Project%20Buffers.htm]

---

## **Part VIII. Appendices**

---

# Table of Contents

A. How this manual was produced .....	635
B. JpGraph Professional License .....	638
B.1. Single License .....	638
B.2. Bulk (Re-seller license) .....	639
C. FAQ .....	641
D. Named color list .....	661
E. Available plot marks .....	673
E.1. Built in basic plot marks .....	673
E.2. Built in image plot marks .....	674
F. List of all country flags .....	677
G. List of files included in the library .....	696
H. Error messages .....	699
H.1. Core error messages .....	699
H.2. QR 2D Barcode error messages .....	710
H.3. Datamatrix 2D barcode error messages .....	713
I. Compiling PHP .....	716
I.1. Compiling PHP4 .....	716
I.2. Compiling PHP5 .....	718
J. Setting up PHP5 in parallel with PHP4 in SuSE 10.1 .....	721
J.1. Configuration files and directories for Apache2 in SuSE 10.1 .....	721
J.2. Making sure you have the correct Apache2 setup .....	722
J.3. Approaches to running multiple PHP versions .....	722
J.4. Outline of the remainder of the chapter .....	723
J.5. Part I - Installing PHP4 .....	724
J.6. Part II - Creating a virtual host .....	726
J.7. Part III - Installing PHP5 .....	728
J.8. Part IV - Verifying the setup .....	728
K. Why it is not possible to add a SVG backend to JpGraph .....	730
K.1. Background .....	730
K.2. Summary of findings .....	730
K.3. Detailing the issue .....	730
L. The JpGraph configuration file .....	733

---

# Appendix A. How this manual was produced

Unfortunately we couldn't locate any off-the-shelf system for producing this fairly large manual with some special requirements like automatic inclusion of PHP source that should be highlighted and in addition rendered by running the scripts and automatically include the resulting images in the resulting manual. To solve this we have based our solution around a DocBook5 setup with some custom steps that are described below.

## DocBook5

The source for the manual is written as a number of split DocBook5 [<http://www.docbook.org/>] XML compliant documents using `XInclude` [<http://www.w3.org/TR/xinclude/>] to bring them together into one master document.

The transformation of the XML source files was done by the means of a DocBook XSL stylesheet using the `xsltproc` XSL processor. (see `libxslt` [<http://xmlsoft.org/XSLT/xsltproc2.html>]) The DocBook5 style sheets can directly produce either single file HTML or chunked (many files) HTML (or XHTML).

In addition there is a style sheet to produce FO (Formatted Objects) output which can be further refined to PDF with the help of the `fop` processor (see Apache FOP [<http://xmlgraphics.apache.org/fop/>]). Unfortunately some formatting instructions in the source are lost in the transformation to PDF output. This means that some aspects of the manual doesn't come out perfect in the PDF output. For this reason the PDF version of the documentation should only be seen as a complementary documentation. The master output format is the chunked HTML.

## Note

In formatting the chunked output we have prioritized to keep down the number of files to avoid many pages with only a small amount of text on them. Our view is that documentation which breaks the pages down to very low levels are extremely tiresome to read.

## Phing based build system

The overall build process is driven by a **Phing** XML build script. **Phing** (See <http://phing.info/trac/>) is most easily described as a PHP version of the Java build system **Ant**. It has several advantages compared with a more traditional **make** setup, the build files are all written in clear XML which makes them easy to read and maintain. In addition there are a number of built-in commands that make deploying and handling of files extremely easy compared with a traditional make system which must rely on external tools to do everything.

## Syntax highlighting of example code

The syntax highlighting and handling of the numerous example images initially posed a small problem since there are no off-the-shelf good support for handling this. As a basic requirement we needed all PHP scripts to be runnable and kept in the normal example directories and then automatically included when the DocBook source was processed.

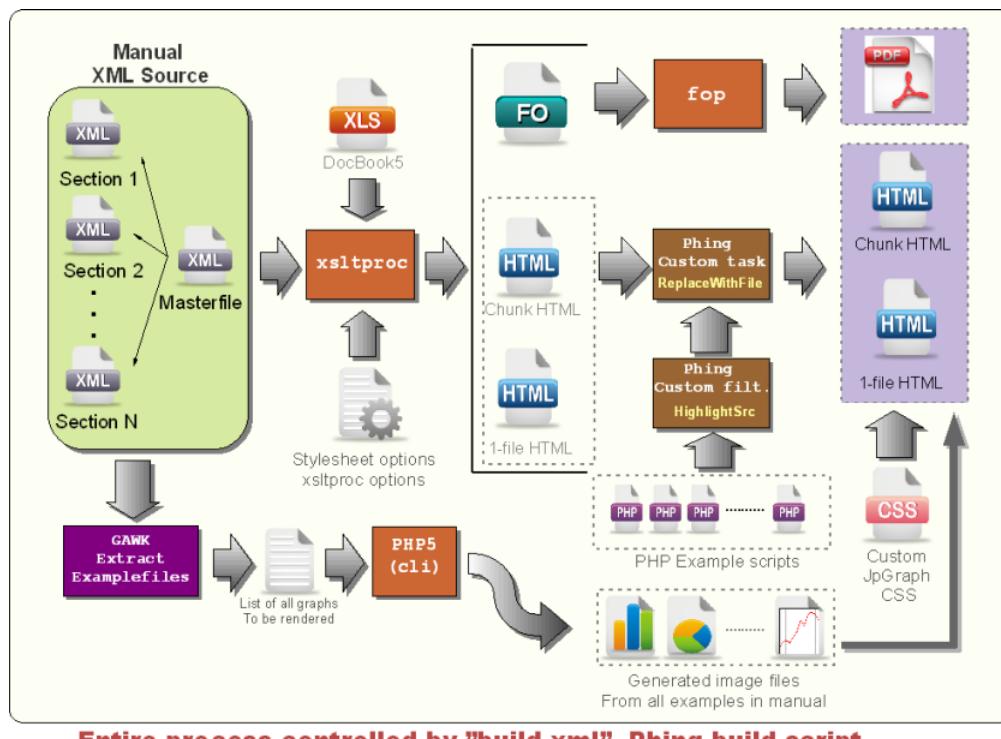
What was needed was some easy way by which we could just mark in the DocBook source (and still maintain valid DocBook XML) that we wanted a particular named example included and either show just the image, just the source or both. In addition we required the source to be syntax highlighted.

To handle this we had to write some custom tasks to extend Phing. In principal our build system works as follows.

1. When a new example have been added or an old one removed a special target in our build file are run which extracts all scripts from the example sections in the \*.XML files. The name of each found example script in the XML source corresponds to an existing PHP script in the Example directory in the JpGraph distribution. A batch file is then automatically created which is subsequently run and all generated images stored on disk.
2. When the normal DocBook XSL processing is done all the special example markups in the XML source is replaced with XML tags to include the image and make sure that the referred script is a proper PHP file name.
3. After the XSL process has been run all programlisting tags will have a special token, for example "\#\#example0.php\#\#" this then instructs a custom **Phing** task to replace the name in the double "#" tags with the corresponding source (in the resulting HTML code). At the same time this source is included it is also passed through the custom syntax highlight filter so that it comes out as proper marked up source which is inserted directly in the resulting HTML file.

The overall build system is illustrated in Figure A.1, “The documentation build process”

**Figure A.1. The documentation build process**



The way the special markup works is that whenever we want a full example (source and image) we create a <programlisting> tag with the file name and title within (single) "#" characters. For example to include the very first example in this manual we have the following tags in the docbook XML source

```
<programlisting>#example0|The very first example#</programlisting>
```

The first part (before the "|") is the file name without extension that we want to include and the second part (after the "|") is the title we want to use. This markup will include both the source as well as the generated graph/image directly in the resulting HTML.

When new examples have been added the examples target in our build file is run and that extracts all the example script used in the book (in the above example "example0.php") and creates a batch file which is then run to create all the images used in the examples.

The syntax highlighting is handled by a custom written filter extensions to **Phing** which internally uses the PEAR package `Pear::Text_Highlight`.

**Notes:**

1. In the current setup a regular expression in the Phing build script is responsible for replacing the markup in the programlisting with a `<figure>` tag and a new `<programlisting>`. This should be done with a custom XSL layer instead and we will update this for the next major revision.
2. Since the syntax highlighting makes use of HTML markup code for the colors the PDF output does not support syntax highlighting
3. For the reference manual we still use our old DB based documentation system which stores all the methods and classes in a DB augmented with source documentation. (We actually prefer this in front of adding a lot of end user documentation with PHPDoc comment sin the source which have a tendency of cluttering up the code as well as making it prone to error since the source files have to be modified in order to update a simple typo in the documentations. Our next step is therefor to update that old system to be able to produce DocBook5 compliant XML for further formatting and processing.

---

# **Appendix B. JpGraph Professional License**

## **B.1. Single License**

1. The JpGraph Professional License will be referred to as "The license" in the remainder of this text.
2. This irrevocable, perpetual license covers versions 1.x & 2.x of JpGraph
3. This license gives the holder right to install JpGraph libraries on One Server which can run one or several virtual HTTP servers.
4. The license holder is allowed to make modifications to the JpGraph sources but in no event will the original copyright holders of this library be held responsible for action or actions resulting from any modifications of the source.
5. The license holder is not required to publicize or otherwise make available any software used in conjunction with JpGraph.
6. The license holder may not re-distribute the library on it's own or versions thereof to third party without prior written permission of the copyright holder.
7. JpGraph License does not allow the library to be redistributed as part of another product.
8. In no event shall the copyright notice in any of the source files supplied in JpGraph be removed or modified.
9. The names "JpGraph" or "Aditus" must not be used to endorse or promote products derived from this software without prior written permission.
10. The license may be transferred to another server by removing all installed files from the old server.
11. The wording of this license may change without notice for future versions of JpGraph.
12. By acquiring a license the licensee agrees to all terms and conditions in this license text.

### **13. Limitations of Liability**

In no event, except for intellectual property claim, shall the initial developers or copyright holders be liable for any damages whatsoever, including - but not restricted to - lost revenue or profits or other direct, indirect, special, incidental or consequential damages, even if they have been advised of the possibility of such damages, except to the extent invariable law, if any, provides otherwise.

In addition, in no event does this license authorize you to use JpGraph in applications or systems where JpGraph's failure to perform can reasonably be expected to result in a physical injury, loss of life or any economical damage. Any such use by the licensee is entirely at the licensee's own risk, and the licensee agrees to hold the original copyright holders of JpGraph harmless from any claims or losses relating to any such unauthorized use.

### **14. Limited Warranty**

JpGraph warrants that licensor is owner of the software with authority to license the software to licensee and that the software does not infringe third party intellectual property rights. Licensor agrees

to indemnify, defend and hold harmless licensee from any claims either that licensor does not own the software or that the software infringes a third party's intellectual property.

THE SOFTWARE AND THIS LICENSE DOCUMENT ARE PROVIDED AS IS. THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, OF ANY KIND, INCLUDING WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

## B.2. Bulk (Re-seller license)

1. The JpGraph Professional License will be referred to as "The license" in the remainder of this text.
2. This irrevocable, perpetual license covers versions 1.x & 2.x of JpGraph
3. This license gives the licensee the right to distribute the JpGraph libraries, as part of its product portfolio and to use the Jpgraph libraries with its hosted service offerings. Specifically this license gives the licensee the right to deploy an unlimited number of JpGraph installations as part of their product portfolio and offer an unlimited number of hosted services that use the JpGraph libraries to an unlimited number of users.
4. The license holder is allowed to make modifications to the JpGraph sources but in no event will the original copyright holders of this library be held responsible for action or actions resulting from any modifications of the source.
5. JpGraph libraries is only licensed to be redistributed as a part of the licensees products. Specifically it may not be sold or re-distributed on it's own.
6. In no event shall the copyright notice in any of the source files supplied in JpGraph be removed or modified.
7. The names "JpGraph" or "Aditus" must not be used to endorse or promote products derived from this software without prior written permission.
8. The wording of this license may change without notice for future versions of JpGraph.
9. By acquiring a license the licensee agrees to all terms and conditions in this license text.

### 10. Limitations of Liability

In no event, except for intellectual property claim, shall the initial developers or copyright holders be liable for any damages whatsoever, including - but not restricted to - lost revenue or profits or other direct, indirect, special, incidental or consequential damages, even if they have been advised of the possibility of such damages, except to the extent invariable law, if any, provides otherwise.

In addition, in no event does this license authorize you to use JpGraph in applications or systems where JpGraphs failure to perform can reasonably be expected to result in a physical injury, loss of life or any economical damage. Any such use by the licensee is entirely at the licensees own risk, and the licensee agrees to hold the original copyright holders of JpGraph harmless from any claims or losses relating to any such unauthorized use.

### 11. Limited Warranty

JpGraph warrants that licensor is owner of the software with authority to license the software to licensee and that the software does not infringe third party intellectual property rights. Licensor agrees to indemnify, defend and hold harmless licensee from any claims either that licensor does not own the software or that the software infringes a third party's intellectual property.

THE SOFTWARE AND THIS LICENSE DOCUMENT ARE PROVIDED AS IS. THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, OF ANY KIND, INCLUDING WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

---

# Appendix C. FAQ

Revision: \$Id: FAQ.xml 1233 2009-05-24 22:29:53Z ljp \$

## C.1. Where should I install the library?

The simple answer is: Anywhere You like! Since the necessary library files are included in Your own script with an "include" (or more likely a require\_once) statement it doesn't matter where You put the library as long as Your scripts can access it. A common place is to store the library is in a directory where PHP searches for include files (as specified in Your php .ini file), for example in "/usr/share/php" or perhaps "/usr/local/php" depending on Your PHP configuration.

An even simpler way is to just store the library in a directory where Your scripts are executed from. However , this is not really recommended.

## C.2. Can I use JpGraph with Windows/Unix/Linux etc?

Yes. The library itself is OS-agnostic. As long as you have a WEB server which supports PHP 4.3.x (or above) and have the GD library. Please note though that the behavior of PHP/Apache on Unix and Windows seems to have some subtle differences which might affect things like path specification using '/' or '\'.

### Caution

The library does not support any 64bit OS and there are known issues with running the library on a 64bit OS (either Unix or Windows). There are currently no plans to address this in the near future.

## C.3. My Web-admin does not allow PHP any write permissions and hence I can't use the cache feature of JpGraph. Is there any way to completely turn off the caching feature?

Yes, set the constant "USE\_CACHE" to FALSE in jpg-config.php. This way will skip writing file even if a filename has been submitted in the Graph() creation call. You can also accomplish this by not having any file name at all in the call to Graph().

## C.4. After installation all I get is an empty page when I try any of the examples?

If the HTTP server response is truly absolutely empty (verify that by looking at the source of the page that gets returned to the browser) then the only answer is that the HTTP server has crashed. Most likely it is the PHP process within the server that has crashed. Even though PHP is fairly stable there are still ways of crashing it. To investigate this further You might try to:

- ... investigate the system log for traces of a server crash.
- ... try increase the maximum allowed memory for PHP in php.ini. As a rule it should be a minimum of 32MB (and preferable 64MB)
- ... upgrade PHP to the latest stable version (and use the compile options shown further down in this FAQ).
- ... upgrade PHP to the latest stable version (and use the compile options shown further down in this FAQ).
- ... enable all errors and warning for PHP (in php.ini) to help track down the problem.
- ... disable output buffering to get errors send back to the browser as soon as possible (in php .ini)

**C.5.** How can I send the generated graph to both the browser and a file at the same time?

This is not directly supported by means of any particular option or setting in the library but with small "trick" this is easy enough to accomplish.

The idea is to use the `_IMG_HANDLER` option that forces the `Graph::Stroke()` to just return the image handler and then stop. We can then manually first send the image to the chosen file and then stream it back to the browser using some internal methods in the library.

The following code snippet shows how this is done

```
<?php
$graph = new Graph(400,300);
<... code to generate a graph ...>
// Get the handler to prevent the library from sending the
// image to the browser
$gdImgHandler = $graph->Stroke(_IMG_HANDLER);

// Stroke image to a file and browser
// Default is PNG so use ".png" as suffix
$fileName = "/tmp/imagefile.png"; $graph->img->Stream($fileName);

// Send it back to browser
$graph->img->Headers();
$graph->img->Stream();
?>
```

**C.6.** What is this 'auto' I see in all the `Graph()` creating in the Examples directory? Do I need it?

No. The third argument in the graph call is an optional cache file name and is not necessary unless You want to use the caching feature of JpGraph. By specifying 'auto' the filename is automatically created based on the name of the script file.

**C.7.** The Client side image map examples does not work on my machine?

In order to run the examples the code generates HTML to read the image from the cache directory as Apache sees it. You must adjust the image tag to have the correct path to your cache directory.

**C.8.** When I run '`testsuit_jpgraph.php`' I get a warning "Due to insufficient permission the source files can't be automatically created"

This is not a serious problem. When the testsuit is run it tries to create a symbolic link with the same base name as the individual testfile but with extension ".phps". This is just so it is possible to click on an image and then view the source syntax colored. If Apache doesn't have write permissions to the directory where '`testsuit_jpgraph.php`' is executed from you will get this warning.

If you want to use this feature just change the permissions so Apache may write to the directory.

**C.9.** How can I pass data to my image generating script?

There are basically three(four) ways:

- Read the data from a file
- Read the data from a DB
- Pass the data in the `<img>` tag using URL parameter passing. For example `` This method is not suitable for

large data sets since there is an upper bound to the length of an URL specification (around 255 characters)

- A fourth way might be to use session variables to pass the data between different scripts. For large data sets the only practical way is to read the data from a file or from a DB to construct the data vectors.

#### C.10. How can I use data from an Excel spreadsheet?

Export the data from Excel as a CSV (Comma separated Values) file to be able to easily read the file. Then either write your own routine to read and parse the data from that file or use the utility class "ReadCSV" in "jpgraph\_utils.php".

A basic example on how to do this for a line graph assuming the name of the data file is "data.csv".

```
<?php include ("../jpgraph.php");
include ("../jpgraph_line.php");
include ("../jpgraph_utils.inc");
$n = ReadFileData::FromCSV('data.csv', $datay);
if($n == false) {
 die("Can't read data.csv.");
}

// Setup the graph $graph = new Graph(400,300);
$graph->title->Set('Reading CSV file with '.$n.' data points');
$graph->SetScale("intlin");

$l1 = new LinePlot($datay);
$graph->Add($l1);
$graph->Stroke();
?>
```

#### C.11. How do I pass data from a Database (e.g. MySQL) to a script to produce a graph?

By adding some SQL statements in Your graph scripts that creates the data arrays from the DB.

Watch out for empty values and strings. JpGraph doesn't know how to plot strings. Any value passed into JpGraph should be only valid numeric data. See the online discussions on populating from a DB Populating DB or the discussion on how to read data from file Reading from file.

See How to populate arrays from MySQL [<http://jpgraph.intellit.nl/viewtopic.php?t=8>]

#### C.12. I change my data but my image does not get updated? The old image is still send back to the browser?

What you are seeing is a cached version of Your image. Depending on your setup it could either be that the browser has a cached image or You are using the built-in cache system in the library by accident.

**Fixing the browser cache** Since the script from the browser point of view is the same it just displays the old image. To fix this you need to turn off the browser image caching.

Another "trick" that can be used since normally it is impossible to control the browser for the end user is to add a "dummy" URL argument which changes. This way the browser will always see a new URL and then always reload that file. This could be done for example as:

```
echo ('');
```

### Fixing the library cache

The other reason might be that You have enabled the built-in cache feature in JpGraph. To turn it off You can either

- Not specify a cache name in the call to Graph(). Just call new Graph(200,200) for example.
- Turn off the reading from the cache by setting READ\_CACHE to false (in jpg-config.inc)
- Turn off the reading and writing from the cache by setting USE\_CACHE to false (in jpg-config.php)
- Use the cache but with a timeout so that the image will be re-generated if the cached image is older then the specified timeout value in the call to Graph().

**C.13.** Can I run JpGraph in batch-mode just outputting an image directly to a file and not streamed back to the browser?

Yes. Just specify an absolute file name in the call to Graph::Stroke(). To store the image in the local temp directory use:

```
$myGraph->Stroke("/tmp/myimage.png");
```

### Caution

Note: You must match the file ending Yourself to the format you have chosen to generate the image in.

**C.14.** Is there a mailing list for JpGraph?

No, but there is a user community forum [<http://jpgraph.intellit.nl/index.php>] at <http://jpgraph.intellit.nl/index.php> where users discuss various aspects of the library.

**C.15.** Do you offer a commercial version of JpGraph with full support and no QPL?

Yes, For commercial (non-open-source) you will need to get the "JpGraph Professional License". This allows you to deploy JpGraph in a commercial context. Further details upon contact.

**C.16.** Some of my script just return a blank image and no error messages?

This could potentially be a lot of things depending on which browser you are using. One thing you might want to check is that your script doesn't take too long time. In that case they might hit the time limit for running a PHP script as specified in php.ini. You can adjust that time limit with a call to set\_time\_limit().

If You are seeing just an empty page (really empty) then this is a sign that Your server has crashed. Most likely an old version of PHP is being used which has taken down the HTTP server process.

**C.17.** I can see the generated image fine in the browser but I can't print it?

This is again browser dependent. Some browser does not like to print an image which is generated on-line and does exist as a \*.jpg, \*.png etc file. No good workaround has been identified short of changing browser. Some browsers (and versions) can print others not. The only real safe way (if you need printing) is to generate the image on disk and then have the browser read the image from there.

**C.18.** How can I generate an image directly to disk without streaming it back to the browser?

Two ways: (Preferred way) If you supply a filename in the call to \$graph->Stroke() the image will be written to that file and not be streamed back to the browser. If you prefer to have the image going to the cache directory but not streamed back then you could just set the '\$aInlineFlag' in the initial call to Graph() (fifth argument) to false as in

```
$mygraph = new Graph(200,200,"auto",0,false);
```

The "auto" argument makes an automatic file name in the cache based on the script name. The 0 indicates that we have no timeout. The advantage with this method is that you can specify a timeout to avoid having the image generated to often.

**C.19.** When I try to plot graphs with many 1000:s of data points I only get a blank page. Is there a limit in JpGraph to the number of data points you can plot?

No, there is no such limit. The problem is most likely that your script takes so long time that the PHP execution limit (as specified in php.ini) aborts the script. If you have more than 10,000 data points JpGraph will work but doing this kind of heavy data analysis on a WEB-server extension is probably not a good idea to start with.

The other reason could be that the script exhausts the allowed memory limit for PHP as specified in the "php.ini" file. By default this is often set to 8Mb which is to low for a complex JpGraph script. Try increasing the memory limit to a minimum of 16Mb. For very complex graphs with 1000s of data points there might be a need to increase the memory even further.

**C.20.** How can I use an image in a dynamically created PDF?

You can do this in two ways, you can either stroke the image created with the library directly to a file where it can be used with Your PDF library of choice. For example, using FPDF ([www.fpdf.org/](http://www.fpdf.org/)) the Image() function would have to be used to read an image from the filesystem.

If You are using the PDF library (APIs included in the PHP distribution but the library itself might require a license depending on Your usage.) you can avoid the storage of the file by getting the image handler for the image directly and use it with PDF.

With the PDF library you must first open the in memory image with a call to pdf\_open\_memory\_image(\$pdf, \$im). The following script places an image from JpGraph directly into a PDF page that is returned when running this script.

```
<?php

$graph = new Graph(...);

// Code to create the graph
//

// Put the image in a PDF page
$im = $graph->Stroke(_IMG_HANDLER);
$pdf = pdf_new();
pdf_open_file($pdf, '');

// Convert the GD image to something the
// PDFlibrary knows how to handle
$pimg = pdf_open_memory_image($pdf, $im);
```

```
pdf_begin_page($pdf, 595, 842);
pdf_add_outline($pdf, 'Page 1');
pdf_place_image($pdf, $pimg, 0, 500, 1);
pdf_close_image($pdf, $pimg);
pdf_end_page($pdf);
pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);

// Send PDF mime headers
header('Content-type: application/pdf');
header("Content-Length: $len");
header("Content-Disposition: inline; filename=foo.pdf");

// Send the content of the PDF file

echo $buf;

// .. and clean up
pdf_delete($pdf);
?>
```

- C.21.** How can I force the user to get a question to download the generated image instead of just displaying it in the browser?

Change your code for creating the image:

```
<?php
$im=$graph->Stroke(_IMG_HANDLER); $filename="chart";
$file_type = "image/png";
$file_ending = "png";
$filename=$filename.".".$file_ending;

header("Content-Type: application/$file_type");
header("Content-Disposition:attachment; filename=".$filename);
header("Pragma: no-cache");
header("Expires:0");
ImagePNG($im);
?>
```

A html link, which directs to the php page (assuming the filename is chart.php): Download Then, if you click on the link, a download dialog appears and you can download the "chart.png".

### Note

If you directly open chart.php, the image is directly streamed to the browser and the download dialog will not open.

- C.22.** How can I open up a CSIM link in a graph in a separate window?

To open a target CSIM (drill down) in a separate window some javascript in the target link is necessary since this action must be performed in the browser. The target URL needs to be in the form of an action to open up a window (with any optional URL argument that might be needed). For

example, assume we have a bar graph and that the user can click on each bar to get more details. When the user clicks on a bar we open up the details with a call to the helper script "bar\_details.php" and send the parameter "id" to the script to separate the individual bars. We would then use the following URL link (with javascript)

```
"javascript:window.open('bar_details.php?id=%d','_new','width=500,height=300')
```

Where the "%d" placeholder must be replaced by the appropriate Id for each bar. To assign the appropriate id we could use the following construct

```
<?php
// Create targets for the image maps so that the details are opened
// in a separate window
$fmtStr = "javascript:window.open('bar_details.php?id=%d','_new','width=500,he
void(0)";
$n = count($datay);
$targ=array();
$alts=array();
for($i=0; $i < $n; ++$i) {
 $targ[$i] = sprintf($fmtStr,$i+1);
 $alts[$i] = 'val=%d';
}

$bplot->SetCSIMTargets($targ,$alts);
$graph->Add($bplot);
...
// Send back the HTML page to browser
$graph->StrokeCSIM();
?>
```

It should be noted that in the above Javascript code we have used the special window name '\_new' which means that a new window will be opened each time a user clicks on a bar. By giving the window a specific name the same window would be re-used every time.

### C.23. No TTF fonts are displayed.

There are potentially many reasons for this. Please start by trying the following bare bones PHP script to see if basic TTF support is available in the PHP setup.

```
<?php
// Change this defines to where Your fonts are stored
DEFINE("TTF_DIR" , "/usr/X11R6/lib/X11/fonts/truetype/");

// Change this define to a font file that You know that You have
DEFINE("TTF_FONTFILE" , "arial.ttf");

// Text to display
DEFINE("TTF_TEXT" , "Hello World!");

$im = imagecreatetruecolor (400, 100);
$white = imagecolorallocate($im, 255, 255, 255);
$black = imagecolorallocate ($im, 0, 0, 0);

imagefilledrectangle($im,0,0,399,99,$white);
imagettftext ($im, 30, 0, 10, 40,$black, TTF_DIR.TTF_FONTFILE,TTF_TEXT);
```

```
header ("Content-type: image/png");
imagepng ($im);
?>
```

The above script when run should produce an image with a black text string on a white background. If this doesn't work then there is a setup problem with PHP + TTF.

As a first step to resolve this recompile Your PHP4 setup using the following basic configuration (depending on the local conditions some options/path might have to be changed). The critical options are marked in bold.

```
./configure
--prefix=/usr/share \
--datadir=/usr/share/php \
--with-apxs=/usr/sbin/apxs \
--libdir=/usr/share --includedir=/usr/include \
--bindir=/usr/bin \
--with-config-file-path=/etc \
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd \
--enable-gd-imgstrttf \
--enable-gd-native-ttf \
--with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib \
--with-jpeg-dir=/usr/lib \
--with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib \
--enable-memory-limit --enable-safe-mode \
--bindir=/usr/bin \
--enable-bcmath --enable-calendar \
--enable-ctype --with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--with-iconv
```

If there is still no TTF fonts displayed then the next step is to upgrade Your TTF libraries to the latest version. The FreeType version 2.1.9 is known to work well.

#### C.24. All my TTF fonts are yellow no matter what color I specify?

You have a old buggy version of the FreeType library, most likely the old FreeType I library is being used. Please upgrade to the latest version of FreeType II.

#### C.25. I get an error "This PHP build has not been configured with TTF support. You need to recompile your PHP installation with FreeType support."

This error is reported if the built-in PHP/GD function "imagettfbbox()" [http://www.php.net/manual/en/function.imagettfbbox.php] is not available. This is a crucial method to make TTF fonts work with JpGraph. The most likely reason for this problem is that the PHP configuration is faulty and does not have correct support for TTF fonts. Please recompile PHP as shown above and make sure that the built-in version of GD is being used and not the stand-alone library.

**C.26.** I get an error "*Fatal error: Font cant be found*"

You have either not specified a correct path where your fonts are stored (see top of jpg-config.inc) or you have tried to use a font you do not have installed.

**C.27.** I get the error "*Can't find font file FONT\_PATHarial.ttf*" (or some other font)

You are most likely running on Windows. This seems to be caused by a bug in PHP/Apache (or PHP/IIS) specifically on Windows. The problem seems to be that PHP doesn't remember a DEFINE() setting across two or more include directive. A simple workaround is to hard-code the font path in class TTF in file jpgraph.php. Note: If you have this problem you will most likely also have a problem with the path to the cache directory.

**C.28.** I get an error "*Font file ... is not readable or does not exist*"

This error is caused by the inability of PHP/JpGraph to read the font file. This could have several reasons:

- The file path or name is wrong and the file really doesn't exist. Please check that the file and path (as shown in the error message) really exists.
- If the file exists it might still not be possible for PHP to read the file due to permission problem. Please make sure that PHP can read a file in the TTF directory. This can be verified with the following short script

```
<?php
// Change this defines to where Your fonts are stored
DEFINE("TTF_DIR" , "/usr/X11R6/lib/X11/fonts/truetype/");

// Change this define to a font file that You know that You have
DEFINE("TTF_FONTFILE" , "arial.ttf");

$f = TTF_DIR.TTF_FONTFILE;
if(file_exists($f) === false || is_readable($f) === false) {
 echo "FAIL: Can not read font file \\"$f\\".";
} else {
 echo "PASS: Successfully read font file \\"$f\\".";
}
?>
```

If the script reports *FAIL* then it must be investigated whether the file names or permissions are wrong.

**C.29.** How can I install additional TTF font families?

See the Adding additional TTF font families [phptip04.php] section on this site for a full explanation of the necessary steps.

**C.30.** How can I display special characters that isn't available on the keyboard?

By using the standard "&#dddd" encoding of the symbol in a text string. For example, to get the character for the "copyright" symbol &copy; which has the numeric code "00A9" (in Hex)

```
$copy = sprintf('&#%04d;', hexdec('00A9'));
```

In a similar way this can be used to show Greek letters, for example "&#0946;" (beta) is displayed by

```
$beta = sprintf('&#%04d;', hexdec('03B2'));
```

## Tip

Use the class "SymChar" defined in `jpgraph_utils.php` to get easy access to greek and special mathematical symbols. This class allow the specification of common symbols by name. Some examples:

```
// δ (delta)
$gamma = SymChar::Get('delta');

// μ (mu)
$mu = SymChar::Get('mu');

// π (pi)
$pi = SymChar::Get('pi');

// λ (lambda)
$lambda = SymChar::Get('lambda');

// ∫ (integral sign)
$integral = SymChar::Get('int');

// ≈ (approximation sign)
$approx = SymChar::Get('approx');
```

### C.31. When I put a text object on a graph and use Set90AndMargin() it doesn't show?

The main cause for this behaviour is that the text object position is rotated 90 degrees. For example, putting a text object at `Pos(0,0)` it will become "invisible". The reason for this behaviour is that the texts position (not the text itself) is rotated around the center of the image 90 degrees clockwise.

For example, assuming the image is `WxH` in size. Rotating position `(0,0)` which is the top left corner of the image gives the rotated position:

```
(0,0) => rotate 90 degrees => (W,0)
```

Now, if a string was put at position `(W,0)` it will be drawn outside the image and hence become invisible.

If You need to position text with rotated images just keep this in mind and swap the meaning of the X and Y position since the X-position will become the Y position and vice versa after a 90 degree rotation. Some experiments will soon allow You to get the hang of it.

## Tip

Start with an image that is equal in `W` and `H` since this makes it easier to understand the rotation.

### C.32. I get an error "*Error: Cannot add header information - headers already sent ...*"

First, this is not a problem with `JpGraph` per se. What has happened is that your PHP file which produces the image has already returned some data to the browser before the image has been sent. This is most often caused by one or more spaces before the first "`<?php`". The browser will now implicitly set the header to plain text.

When later JpGraph tries to send the correct image header the browser gets confused since it has already received a header for the current document. (Each document can only have one and only one type). Check your files for any output (even a single space) before the call to Graph::Graph() (or Graph::Stroke())

If you are running on Windows this problem could be caused by a blank line at the end of any of the jpggraph\_\*.php files. All files in jpggraph ends in a newline which some configurations on Windows doesn't like. Remove the newline so that the file ends just after the final "?>"

Also remember that when you include external file using include/include\_once and so on Php include the whole content of the file; this content of the file also include any potential carriage return/line feed or "blank" space before "<?php" and after "?>" These "dirty characters" will stop the JpGraph being properly sent back because they are included in the image streaming.

### C.33. I get an error "*Unknown function imagecreate*"

You have not compiled PHP with support for the GD graphic library. See PHP documentation for details. Note that you also need to configure PHP to use GD and/or TTF fonts to get JpGraph full functionality

### C.34. I get an error "*Unknown function imagecreatejpeg*"

You have not compiled PHP with support for the JPG format. See PHP documentation for details

### C.35. I get an error "*Warning: MkDir failed (Permission denied) in jpggraph.php on line XXXX*"

This is a permission problem. To use the cache Apache/PHP must have write permission to the cache directory. Make sure that Apache/PHP have sufficient permissions to write to the specified cache directory.

### C.36. I get an error "*Fatal error: Undefined class name "parent" in jpggraph.php on line xxxx*"

You are using an older version of PHP. JpGraph 1.x requires at least PHP 4.3

## Note

Remember that JpGraph 1.x has reached its end of life and is no longer maintained.

### C.37. I get an error "*Your data contains non-numeric values.*"

Most likely Your data really contains non-numeric data which You need to further investigate (for example by printing out the array with a var\_dump( ) . One additional thing to watch out for is if the data looks like ".56" (or "-.56") which is a shortform of "0.56". The problem is that the number starts with an "." (dot) which is non-numeric. The solution is to replace the single dot with a "0."

### C.38. I get an error "*Date is outside specified scale range*" trying to create a Gantt chart

As the error says, you start or end date for a activity is larger/smaller than the max/min value in the scale. If you get this error and you are using automatic scale then you are probably using a null value (empty string "") as start or end date. Check your data

### C.39. I get an error "*Invalid file handle in jpggraph.php at line XXXX*"

This is a permission problem. Check that your cache directory has the right permissions to let JpGraph access and write to it.

**C.40.** I get an error "*Fatal error: Call to undefined function: getcsimareas() ...*"

You have enabled the `DEFINE( "JPG_DEBUG" , true )` in `jpgraph.php`. This is an internal debug flag which should always be left untouched.

**C.41.** I get an error "*Fatal error: Call to undefined function: imagetypes()*"

You have not installed the core GD library correct. `imagetypes()` is a standard GD function. Please refer to the installation section in the manual for more information on how to verify You setup.

**C.42.** I get an error saying something like "*Error: 'parent::' not valid in context...*"

You are most likely using the Zend cache. There is a bug with some versions of the Zend cache together with the "parent" meta class. If the referenced parent class is in another file that is included this problem seems to occur. The workaround seems to be to put all classes in the same physical file or upgrade the Zend cache to a more recent version.

**C.43.** I get an error "*Image cannot be displayed because it contains errors..*"

First of all this is an error message from the browser and not the library. There could be various reasons for this but the most likely scenario is that Your installation of PHP has both enabled output buffering and enabled strict error checking.

What could now happen is that with output buffering enabled PHP will buffer up any text error messages. This could for example be som "Notice:" warnings if You have forgotten to initialize some variables. When then the library adds the proper image header and the buffered error messages together with the image gets sent back the browser it all falls apart.

The browser tries to interpret both the error messages and the image data as an image and it of course fails and gives this error message. Why? Because the textual error messages that was buffered is interpreted as if they would belong to the image which of course is nonsense.

The solution is to disable output-buffering until You are convinced that Your code is without any problem. On a development server it is recommended to always have output buffering disabled.

**C.44.** When I use PHP5 + JpGraph 2.x I get "*A plot has an illegal scale...*"

There have been a few reports of this problem and it seems to be due to a faulty PHP 5.x installation. For example a standard SuSE 9.3 PHP5 installation gives this problem. The exact reason for the problem has not been established but it all points to a PHP5 problem since it can be solved by adjusting the PHP installation without any change in the library.

**Step 1:**

First check that Your `php.ini` file have compatibility mode turned off (in `php.ini`)

```
zend.zel_compatibility_mode=off
```

**Step 2:**

If turning the compatibility mode off still doesn't solve the problem the only remaining solution is to re-compile PHP 5. For example on our development servers we use the following configuration Note: We place "`php.ini`" in `"/etc/php5/apache2"` and You might want to change this to suit Your specific setup.

```
./configure
--prefix=/usr/share --datadir=/usr/share/php \
```

```
--with-apxs2=/usr/sbin/apxs2-prefork \
--libdir=/usr/share
--includedir=/usr/include \
--bindir=/usr/bin \
--with-config-file-path=/etc/php5/apache2 \
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd --enable-gd-imgstrttf --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib --with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-memory-limit
--enable-safe-mode \
--bindir=/usr/bin \
--enable-bcmath --enable-calendar \
--enable-ctype --with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--with-bz2 \
--with-iconv
```

- C.45.** I try to mix bar and line graphs and the bars doesn't get centered and the labels are not correctly aligned?

This is a known problem. The best way to partly solve this is to make sure you add the bar plot as the last plot to the graph. The reason behind this problem is that plot that gets added to the graph internally makes modification to the alignment of the plot. Since each plot does not know what other plot has been added it will happily overwrite previous plots settings in the graph. Hence the workaround by adding the bar last. (conflicting settings.)

- C.46.** How do I turn off anti-aliasing for TTF fonts?

There is not any built in support in JpGraph to do this. Since the whole purpose of TTF fonts is to provide a smooth font experience there is really no point in wanting to do this. If you want non-aliased fonts you should use the built-in bitmap fonts.

- C.47.** The auto-scaling algorithm chooses very tight limits around my Y-values. How do I get more "space" between the end of the scale and the min/max values?

Use the `SetGrace()` to specify some extra space (as percentage of the scale) between the min/max value and the limits of the scale. For example `$graph->yscale->SetGrace(10)` gives 10% extra space at the ends.

- C.48.** I want to use autoscaling to handle the maximum value but I always want the Y-axis to start at 0 regardless of the actual minimum value?

Use the `SetAutoMin()` method. For example as in

```
...
$graph = new Graph(400,350);
$graph->SetScale("intlin");
$graph->yaxis->scale->SetAutoMin(0);
...
```

## Tip

If you need to limit the maximum value but auto-scale the minimum value use the `SetAutoMax()` method.

- C.49.** I want the position of the X-axis to be at the bottom of the Y-scale and not always at the position Y=0?

To adjust the position of the Axis Use the `Axis::SetPos()` method. The argument to this method can be either the numerical scale value on the Y-axis where the X-axis should be positioned or it can be one of the two special value "min" or "max". In the latter case the axis will be positioned on either the minimum or the maximum value of the scale. For example

```
...
$graph->xaxis->SetPos("min");
...
```

Will put the X-axis at the lowest value of the Y-axis.

## Note

The same positioning is possible for the Y-axis.

- C.50.** I specify color X in the image but the color displayed is not exactly what I specified?

This is a result of a finite color palette for GIF, PNG formats. If you are using anti-aliasing, perhaps a background image or even gradient fill you might exhaust the color palette supported in the GD library. Try set the constant `USE_APPROX_COLORS` to false and generate the picture again. If you now get an error message saying that no more colors can be allocated this is the problem. There is no good workaround if you are using GD 1.x since for PNG the GD 1.x library does not support "True-color".

The only real solution is to upgrade to GD 2.x to get full true-color support. If you are using a background image try to "degrade" it to have a smaller color palette or turn off anti-aliasing and you might have enough free palette entries to cater for all the colors you want and still use GD 1.x

## Note

This problem is only seen in very, very old installations that use GD 1.x library which has been deprecated since around 2005

- C.51.** Can I have labels at an arbitrary angle on the X-axis?

Yes, almost any text can be stroked at an arbitrary angle. For example to have the labels at 45 degrees angle use

```
...
$graph->xaxis->SetTickLabels($labels);
$graph->xaxis->SetLabelAngle(45);
...
```

## Note

Internal fonts support 0 and 90 degrees text. If you need to use, say 45 degree (or any other arbitrary angle), you must use TTF fonts.

**C.52.** How do I invert the Y-scale so that it has the minimum value on top and the largest value at the bottom?

The easiest way to do this is by following a two step process

a) Negating all Your values and then

b) Create a callback function for the Y-axis that negates all the display values.

What will happen now is that after the negative values have been feed into the graph it will create a scale starting at the lowest value, say -8, then go up to the highest, say -1. If these values are then inverted when printed it will in affect achieve the inverted axis appearance. The code snippet below shows a basic example of this technique.

```
... // Callback to negate the argument
function _cb_negate($aVal) {
 return round(-$aVal);
} // A fake depth curve
$ydata =
// Negate all data
$n = count($ydata);
for($i=0; $i<$n; ++$i) {
 $ydata[$i] = -$ydata[$i];
}

// Basic graph setup
$graph = new Graph(400,300);
$graph->SetScale("linlin");

// Setup axis
$graph->yaxis->SetLabelFormatCallback("_cb_negate");
$plot = new LinePlot($ydata,$xdata);
$graph->Add($plot);
$graph->Stroke();
```

Two scripts that uses this technique can be found in the Examples/ directory in the distribution, "inyaxisex1.php" and "inyaxisex2.php".

**C.53.** Can I have the auto-scaling algorithm restrict itself to whole numbers?

Yes, use the "int" scale specification, for example \$graph->SetScale('inttext') will make the Y-axis have an integer scale and the X-axis a text-scale

**C.54.** Is it possible to have horizontal bar graphs?

Yes, Just create a normal bar and call the method Set90AndMargin() to rotate the graph.

**C.55.** Line weight does not seem to work in my graphs? I specify a thick line but still get a thin line?

You have probably enabled Anti-aliasing. If anti-aliasing is enabled setting line-weight will have no affect.

**C.56.** How can I have more space between the Y-axis and the title?

Use the Axis::SetTitleMargin() method. For example to have a 25 pixels margin for the Y-title you could use:

```
$mygraph->yaxis->SetTitleMargin(25);
```

**C.57.** How can I display both X and Y values above each data point in a line, bar or scatter plot?

You need to use a value formatting callback. The callback function is called for each data point and is passed the Y-value as argument. This means that if You also need to display the x-value it must be available in the callback function, perhaps as a global data array.

Something along the lines of

```
<?php
$datay = array(...);
$datax = array(...);
$idx=0;

function xyCallback($yval) {
 global $datax, $idx;
 return '('. $datax[$idx++] . ", $yval)";
}

// Some more code ...

$graph = new Graph(...);
$graph->SetScale("intlin");
$p1 = new LinePlot($datay,$datax);
$p1->value->SetFormatCallback('xyCallback');
$p1->value->Show();

// Some more code ...

$graph->Stroke();
?>
```

**C.58.** Can I display stock box-charts (open, close, high, low) with JpGraph?

Yes, just create a `StockPlot()` graph.

**C.59.** Is there any way to specify a newline in the legend box for a specific legend?

Yes. From version 1.8 full multi-line text paragraphs are supported. Just make sure that your text uses double-quotes and a newline character.

**C.60.** How can I print unicode characters?

Use `&#XXXX;` format in your strings where XXXX is the decimal value for the unicode character. You may find a list of Unicode characters and their encodings at [www.unicode.org](http://www.unicode.org) [<http://www.unicode.org/charts/>] Please observe that the encoding in the lists are given in hexadeciml and these values must be converted to decimal.

Note: If You are working in an UTF-8 environment then the characters may be input directly.

**C.61.** My background images doesn't display properly. It just shows a black solid square?

You are using a very old GD version (probably 2.0.1). Background images only work with a true-color image, (enable the `USE_TRUECOLOR` constant). Some people have reported that it works

as long as the background image is not in PNG format. The drawback with truecolor images is that truefont doesn't work properly in GD versions < 2.0.8

**C.62.** How do I make each bar in a bar plot have it's own color?

Specify an array as argument to BarPlot::SetColor( ) as in

```
$mybarplot->SetColor(array("red","green","blue","gray"));
```

**C.63.** How can I adjust the position of the axis title?

You can add an alignment parameter when you specify the title. You specify one of "high", "low" or "center" as in

```
$mygraph->xaxis->SetTitle("mytitle","high");
$mygraph->xaxis->SetTitle("mytitle","center");
$mygraph->xaxis->SetTitle("mytitle","low");
```

**C.64.** How can I change the image format, i.e jpeg, gif or png?

Use the Image::SetImgFormat( ) method at runtime. You can also change the default value with the DEFAULT\_GFORMAT define in jpg-config.php. This is normally set to 'auto' format which means that JpGraph will automatically choose the format depending on what is available.

For example to use the JPEG format :

```
$graph->img->SetImgFormat('jpeg');
```

**C.65.** How do I specify the font for legends?

Use the Legend::SetFont() method. As in

```
$graph->legend->SetFont(FF_FONT1,FS_BOLD);
```

**C.66.** When I rotate text and labels the text is not rotated correctly?

Red Hat 7.2 has a bug in it's TTF libraries (FreeType 2.0.3) that comes with the distribution. Upgrade to FreeType 2.0.9.

**C.67.** When I rotate a paragraph (multi-line) text the paragraph is always left aligned even though I have specified another alignment. It seems fine for non-rotated paragraphs though. What is the problem?

This is a limitation with the current implementation. (The reason is that GD does not support inter-paragraph alignment so all that logic is done within the libraries string-layout engine.)

At the moment the logic only implements this paragraph alignment for non-rotated paragraphs.) For rotated paragraphs left-alignment will always be used regardless of specified paragraph alignment.

**C.68.** I have a huge number of data points and I only want labels (which I want to specify as text string) and ticks on every n:th data. Can I do this?

Yes. What most users stumble on is the fact that the label array must still contain data for all the data points even though only every n:th data label is displayed.

This means that if your original data array only contains just the display values it is necessary first to augment the label array with padding spaces to fill it out.

It is then possible to use the method Axis::SetTextTickInterval( ) to specify that only every n:th tick+label should be displayed. Something like:

```
// Setup the X-axis
$graph->xaxis->SetTickLabels($datax);
$graph->xaxis->SetTextTickInterval(...);
```

**C.69.** How can I make the labels have a 1000 comma separator?

The easiest way is to use the PHP built-in function "number\_format()" as a callback (see php manual. number\_format() [http://www.php.net/manual/en/function.number-format.php])

For example, to have the line plot display values using this format first setup and install the callback function as:

```
$lineplot->value->SetFormatCallback("number_format");
```

If you instead want the Y-axis label to use this kind of formatting install the callback using

```
$graph->yaxis->SetLabelFormatCallback("number_format");
```

**C.70.** The Y2 axis is no longer positioned at the right side of the graph

This will happen after that You have manually specified the scale and/or the tick spacing. If the user manipulates the default scale the library defaults to setting the position of the Y2 axis to the 0 X-position. It is easy to re-adjust the Y2 axis position by calling SetPos() as the following code snippet demonstrates

```
$graph->y2axis->SetPos('max');
$graph->y2axis->SetTitleSide('right');
```

**C.71.** How can I include several graphs in the same image?

This can be solved by a bit of extra GD code since the library by default do not directly support this functionality. The easiest way to solve this is to manually create a big enough empty graph with plain GD commands. Then get the image handles from the Stroke() methods in the graphs that should be included. It is then possible to use these handles together with the GD copy command and copy the individual graphs to the previously created empty large GD image.

This is illustrated in the following code snippet. In the example below we assume that graph1 is the same size as the final combined graph we would like to create. This way our combined graph get a background and color as specified in the first graph.

```
// Assume we would like to combine graph1,2 and 3

// create graph 1 here.....
$handle1 = $graph1->Stroke(_IMG_HANDLER);

// create graph 2 here.....
$handle2 = $graph2->Stroke(_IMG_HANDLER);

// create graph 3 here.....
$handle3 = $graph3->Stroke(_IMG_HANDLER);

// Now create the "melting graph" which should contain all three graphs
// $WIDTH1 and $HEIGHT1 are width and height of graph 1 ($handle1)
// $WIDTH2 and $HEIGHT2 are width and height of graph 2 ($handle2)
```

```
// $WIDTH3 and $HEIGHT3 are width and height of graph 3 ($handle3)
// $x2,$x3 and $y2,$y3 shift from top left of global graph (ie position of
// graph2 and graph3 in global graph)
$image = imagecreatetruecolor($WIDTH1,$HEIGHT1);
imagecopy($image, $handle1, 0, 0, 0, $WIDTH1,$HEIGHT1);
imagecopy($image, $handle1,$x1,0,0,$WIDTH2,$HEIGHT2);
imagecopy($image, $handle1,$x2,$y2,0,0,$WIDTH3,$HEIGHT3);

// Stream the result back as a PNG
image header("Content-type: image/png");
imagepng ($image);
```

**C.72.** How can I have an exploded slice with 3D pie plots?

Use either `Pi3DPlot::ExplodeSlice()` or `Pie3DPlot::Explode()`

**C.73.** How can I display values for each slice close to the pie?

Use the method `PiePlot::value::Show()`

**C.74.** How can I change between percentage and absolute values for pie slices?

Use `PiePlot::SetValueType($aType)` where `$aType` is either `PIE_VALUE_ABS` or `PIE_VALUE_PERCENTAGE`. To hide/show values on the pie you access the `value` property of the plot (just like in line plots) If you want some special format of the string you also need to specify a format string. By default just a number gets printed. If you (for example) want percent with a "%" sign you should use a format string like "%d%" (assuming you just want to display whole numbers)

```
$mypieplot->value->SetFormat("%d%");
$mypieplot->value->Show(); // Defaults to TRUE
$mypieplot->SetValueType(PIE_VALUE_PERCENTAGE);
```

**C.75.** Can I display the actual value as labels on the pie bar instead of the percentage?

Yes, use

```
$pieplot->SetLabelType(PIE_LABEL_ABS);
```

and make sure labels are not hidden.

**C.76.** How can I adjust the start angle of the first slice?

Use

```
$pieplot->SetStartAngle(45);
```

**C.77.** When I use IE v5 or v6 I can only save the generated graph image in the browser as a BMP file even if it was generated as a JPEG or PNG encoded image?

This is a bug in the way IE handles HTTP headers. By default the library sends no-cache headers to instruct the browser to always make a HTTP request to the server. This is necessary since graph scripts normally have the same name but the graphs may change over time and we don't want the browser to be fooled by thinking it has already fetched the graph before.

The only known workaround is to tell the library not to send back any no-cache headers as illustrated below.

```
$graph->img->SetExpired(false);
$graph->Stroke();
```

The `SetExpired(false)` call instructs the library to not send back any has-expired headers.

---

# Appendix D. Named color list

See Chapter 7, *Color handling* for a discussion on how to specify colors for graphic objects in the graph.

**Table D.1. Named color list**

Color name	RGB Triple	Hex	Color
AntiqueWhite1	(255,239,219)	#ffefdb	
AntiqueWhite2	(238,223,204)	#eedfcc	
AntiqueWhite3	(205,192,176)	#cdc0b0	
AntiqueWhite4	(139,131,120)	#8b8378	
aliceblue	(240,248,255)	#f0f8ff	
antiquewhite	(250,235,215)	#faebd7	
aqua	(0,255,255)	#00ffff	
aquamarine	(127,255,212)	#7fffad	
aquamarine1	(127,255,212)	#7fffad	
aquamarine2	(118,238,198)	#76eec6	
aquamarine3	(102,205,170)	#66cdcc	
aquamarine4	(69,139,116)	#458b74	
azure	(240,255,255)	#f0ffff	
azure1	(240,255,255)	#f0ffff	
azure2	(224,238,238)	#e0eeee	
azure3	(193,205,205)	#c1cdcc	
azure4	(131,139,139)	#838b8b	
beige	(245,245,220)	#f5f5dc	
bisque	(255,228,196)	#ffcc4c	
bisque1	(255,228,196)	#ffcc4c	
bisque2	(238,213,183)	#eed5b7	
bisque3	(205,183,158)	#cdb79e	
bisque4	(139,125,107)	#8b7d6b	
black	(0,0,0)	#000000	
blanchedalmond	(255,235,205)	#ffebcd	
blue	(0,0,255)	#0000ff	
blueviolet	(138,43,226)	#8a2be2	
brown	(165,42,42)	#a52a2a	
brown1	(255,64,64)	#ff4040	
brown2	(238,59,59)	#ee3b3b	
brown3	(205,51,51)	#cd3333	
brown4	(139,35,35)	#8b2323	
burlywood	(222,184,135)	#deb887	

Color name	RGB Triple	Hex	Color
burlywood1	(255,211,155)	#ffd39b	
burlywood2	(238,197,145)	#eec591	
burlywood3	(205,170,125)	#cdcaa7d	
burlywood4	(139,115,85)	#8b7355	
cadetblue	(95,158,160)	#5f9ea0	
cadetblue1	(152,245,255)	#98f5ff	
cadetblue2	(142,229,238)	#8ee5ee	
cadetblue3	(122,197,205)	#7ac5cd	
cadetblue4	(83,134,139)	#53868b	
chartreuse	(127,255,0)	#7fff00	
chartreuse1	(127,255,0)	#7fff00	
chartreuse2	(118,238,0)	#76ee00	
chartreuse3	(102,205,0)	#66cd00	
chartreuse4	(69,139,0)	#458b00	
chocolate	(210,105,30)	#d2691e	
chocolate1	(255,127,36)	#ff7f24	
chocolate2	(238,118,33)	#ee7621	
chocolate3	(205,102,29)	#cd661d	
chocolate4	(139,69,19)	#8b4513	
coral	(255,127,80)	#ff7f50	
coral1	(255,114,86)	#ff7256	
coral2	(238,106,80)	#ee6a50	
coral3	(205,91,69)	#cd5b45	
coral4	(139,62,47)	#8b3e2f	
cornflowerblue	(100,149,237)	#6495ed	
cornsilk	(255,248,220)	#fff8dc	
cyan	(0,255,255)	#00ffff	
cyan1	(0,255,255)	#00ffff	
cyan2	(0,238,238)	#00eeee	
cyan3	(0,205,205)	#00cdcd	
cyan4	(0,139,139)	#008b8b	
darkblue	(0,0,139)	#00008b	
darkcyan	(0,139,139)	#008b8b	
darkgoldenrod	(184,134,11)	#b8860b	
darkgoldenrod1	(255,185,15)	#ffb90f	
darkgoldenrod2	(238,173,14)	#eedad0e	
darkgoldenrod3	(205,149,12)	#cd950c	
darkgoldenrod4	(139,101,8)	#8b6508	

Color name	RGB Triple	Hex	Color
darkgray	(100,100,100)	#646464	
darkgreen	(0,100,0)	#006400	
darkkhaki	(189,183,107)	#bdb76b	
darkmagenta	(139,0,139)	#8b008b	
darkolivegreen	(85,107,47)	#556b2f	
darkolivegreen1	(202,255,112)	#caff70	
darkolivegreen2	(188,238,104)	#bcee68	
darkolivegreen3	(162,205,90)	#a2cd5a	
darkolivegreen4	(110,139,61)	#6e8b3d	
darkorange	(255,140,0)	#ff8c00	
darkorange1	(255,127,0)	#ff7f00	
darkorange2	(238,118,0)	#ee7600	
darkorange3	(205,102,0)	#cd6600	
darkorange4	(139,69,0)	#8b4500	
darkorchid	(153,50,204)	#9932cc	
darkorchid1	(191,62,255)	#bf3eff	
darkorchid2	(178,58,238)	#b23aee	
darkorchid3	(154,50,205)	#9a32cd	
darkorchid4	(104,34,139)	#68228b	
darkred	(139,0,0)	#8b0000	
darksalmon	(233,150,122)	#e9967a	
darkseagreen	(143,188,143)	#8fbcb8	
darkseagreen1	(193,255,193)	#c1fffc1	
darkseagreen2	(180,238,180)	#b4eeb4	
darkseagreen3	(155,205,155)	#9bcd9b	
darkseagreen4	(105,139,105)	#698b69	
darkslateblue	(72,61,139)	#483d8b	
darkslategray	(47,79,79)	#2f4f4f	
darkslategray1	(151,255,255)	#97ffff	
darkslategray2	(141,238,238)	#8deeee	
darkslategray3	(121,205,205)	#79cdcd	
darkslategray4	(82,139,139)	#528b8b	
darkturquoise	(0,206,209)	#00ced1	
darkviolet	(148,0,211)	#9400d3	
deepindigo	(138,43,226)	#8a2be2	
deeppink	(255,20,147)	#ff1493	
deeppink1	(255,20,147)	#ff1493	
deeppink2	(238,18,137)	#ee1289	

Color name	RGB Triple	Hex	Color
deeppink3	(205,16,118)	#cd1076	
deeppink4	(139,10,80)	#8b0a50	
deepskyblue	(0,191,255)	#00bfff	
deepskyblue1	(0,191,255)	#00bfff	
deepskyblue2	(0,178,238)	#00b2ee	
deepskyblue3	(0,154,205)	#009acd	
deepskyblue4	(0,104,139)	#00688b	
dimgray	(105,105,105)	#696969	
dodgerblue	(30,144,255)	#1e90ff	
dodgerblue1	(30,144,255)	#1e90ff	
dodgerblue2	(28,134,238)	#1c86ee	
dodgerblue3	(24,116,205)	#1874cd	
dodgerblue4	(16,78,139)	#104e8b	
eggplant	(144,176,168)	#90b0a8	
electricindigo	(102,0,255)	#6600ff	
firebrick	(178,34,34)	#b22222	
firebrick1	(255,48,48)	#ff3030	
firebrick2	(238,44,44)	#ee2c2c	
firebrick3	(205,38,38)	#cd2626	
firebrick4	(139,26,26)	#8b1a1a	
forestgreen	(34,139,34)	#228b22	
gainsboro	(220,220,220)	#dcdcdc	
gold	(255,215,0)	#ffd700	
gold1	(255,215,0)	#ffd700	
gold2	(238,201,0)	#eec900	
gold3	(205,173,0)	#cdad00	
gold4	(139,117,0)	#8b7500	
goldenrod	(218,165,32)	#daa520	
goldenrod1	(255,193,37)	#ffc125	
goldenrod2	(238,180,34)	#eeb422	
goldenrod3	(205,155,29)	#cd9b1d	
goldenrod4	(139,105,20)	#8b6914	
gray	(190,190,190)	#bebebe	
gray1	(10,10,10)	#0a0a0a	
gray2	(40,40,30)	#28281e	
gray3	(70,70,70)	#464646	
gray4	(100,100,100)	#646464	
gray5	(130,130,130)	#828282	

Color name	RGB Triple	Hex	Color
gray6	(160,160,160)	#a0a0a0	
gray7	(190,190,190)	#bebebe	
gray8	(210,210,210)	#d2d2d2	
gray9	(240,240,240)	#f0f0f0	
green	(0,255,0)	#00ff00	
greenyellow	(173,255,47)	#adff2f	
honeydew	(193,205,193)	#c1cdc1	
hotpink	(255,105,180)	#ff69b4	
hotpink1	(255,110,180)	#ff6eb4	
hotpink2	(238,106,167)	#ee6aa7	
hotpink3	(205,96,144)	#cd6090	
hotpink4	(139,58,98)	#8b3a62	
indianred	(205,92,92)	#cd5c5c	
indianred1	(255,106,106)	#ff6a6a	
indianred2	(238,99,99)	#ee6363	
indianred3	(205,85,85)	#cd5555	
indianred4	(139,58,58)	#8b3a3a	
indigo	(75,0,130)	#4b0082	
indigodye	(0,65,106)	#00416a	
ivory	(255,255,240)	#fffff0	
ivory1	(255,255,240)	#fffff0	
ivory2	(238,238,224)	#eeee0	
ivory3	(205,205,193)	#cdcfc1	
ivory4	(139,139,131)	#8b8b83	
khaki	(240,230,140)	#f0e68c	
khaki1	(255,246,143)	#fff68f	
khaki2	(238,230,133)	#eee685	
khaki3	(205,198,115)	#cdc673	
khaki4	(139,134,78)	#8b864e	
lavender	(230,230,250)	#e6e6fa	
lavenderblush	(255,240,245)	#fff0f5	
lavenderblush1	(255,240,245)	#fff0f5	
lavenderblush2	(238,224,229)	#eee0e5	
lavenderblush3	(205,193,197)	#cdc1c5	
lavenderblush4	(139,131,134)	#8b8386	
lawngreen	(124,252,0)	#7cfcc0	
lemonchiffon	(255,250,205)	#fffacd	
lemonchiffon1	(255,250,205)	#fffacd	

Color name	RGB Triple	Hex	Color
lemonchiffon2	(238,233,191)	#eee9bf	
lemonchiffon3	(205,201,165)	#cdc9a5	
lemonchiffon4	(139,137,112)	#8b8970	
lightblue	(173,216,230)	#add8e6	
lightblue1	(191,239,255)	#bfefef	
lightblue2	(178,223,238)	#b2dfee	
lightblue3	(154,192,205)	#9ac0cd	
lightblue4	(104,131,139)	#68838b	
lightcoral	(240,128,128)	#f08080	
lightcyan	(224,255,255)	#e0ffff	
lightcyan1	(224,255,255)	#e0ffff	
lightcyan2	(209,238,238)	#d1eeee	
lightcyan3	(180,205,205)	#b4cdcd	
lightcyan4	(122,139,139)	#7a8b8b	
lightgoldenrod	(238,221,130)	#eedd82	
lightgoldenrod1	(255,236,139)	#ffec8b	
lightgoldenrod2	(238,220,130)	#eedc82	
lightgoldenrod3	(205,190,112)	#cdbe70	
lightgoldenrod4	(139,129,76)	#8b814c	
lightgoldenrodyellow	(250,250,210)	#fafad2	
lightgray	(211,211,211)	#d3d3d3	
lightgreen	(144,238,144)	#90ee90	
lightpink	(255,182,193)	#ffb6c1	
lightpink1	(255,174,185)	#ffaeb9	
lightpink2	(238,162,173)	#eea2ad	
lightpink3	(205,140,149)	#cd8c95	
lightpink4	(139,95,101)	#8b5f65	
lightred	(211,167,168)	#d3a7a8	
lightsalmon	(255,160,122)	#ffa07a	
lightsalmon1	(255,160,122)	#ffa07a	
lightsalmon2	(238,149,114)	#ee9572	
lightsalmon3	(205,129,98)	#cd8162	
lightsalmon4	(139,87,66)	#8b5742	
lightseagreen	(32,178,170)	#20b2aa	
lightskyblue	(135,206,250)	#87cefa	
lightskyblue1	(176,226,255)	#b0e2ff	
lightskyblue2	(164,211,238)	#a4d3ee	
lightskyblue3	(141,182,205)	#8db6cd	

Color name	RGB Triple	Hex	Color
lightskyblue4	(96,123,139)	#607b8b	
lightslateblue	(132,112,255)	#8470ff	
lightslategray	(119,136,153)	#778899	
lightsteelblue	(176,196,222)	#b0c4de	
lightsteelblue1	(202,225,255)	#caelff	
lightsteelblue2	(188,210,238)	#bcd2ee	
lightsteelblue3	(162,181,205)	#a2b5cd	
lightsteelblue4	(110,123,139)	#6e7b8b	
lightyellow	(255,255,200)	#fffffc8	
lime	(0,255,0)	#00ff00	
limegreen	(50,205,50)	#32cd32	
linen	(250,240,230)	#faf0e6	
magenta	(255,0,255)	#ff00ff	
magenta1	(255,0,255)	#ff00ff	
magenta2	(238,0,238)	#ee00ee	
magenta3	(205,0,205)	#cd00cd	
magenta4	(139,0,139)	#8b008b	
maroon	(176,48,96)	#b03060	
maroon1	(255,52,179)	#ff34b3	
maroon2	(238,48,167)	#ee30a7	
maroon3	(205,41,144)	#cd2990	
maroon4	(139,28,98)	#8b1c62	
mediumaquamarine	(102,205,170)	#66cdaa	
mediumblue	(0,0,205)	#0000cd	
mediumorchid	(186,85,211)	#ba55d3	
mediumorchid1	(224,102,255)	#e066ff	
mediumorchid2	(209,95,238)	#d15fee	
mediumorchid3	(180,82,205)	#b452cd	
mediumorchid4	(122,55,139)	#7a378b	
mediumpurple	(147,112,219)	#9370db	
mediumpurple1	(171,130,255)	#ab82ff	
mediumpurple2	(159,121,238)	#9f79ee	
mediumpurple3	(137,104,205)	#8968cd	
mediumpurple4	(93,71,139)	#5d478b	
mediumred	(140,34,34)	#8c2222	
mediumseagreen	(60,179,113)	#3cb371	
mediumslateblue	(123,104,238)	#7b68ee	
mediumspringgreen	(0,250,154)	#00fa9a	

Color name	RGB Triple	Hex	Color
mediumturquoise	(72,209,204)	#48d1cc	
mediumvioletred	(199,21,133)	#c71585	
midnightblue	(25,25,112)	#191970	
mintcream	(245,255,250)	#f5ffffa	
mistyrose	(255,228,225)	#ffe4e1	
mistyrose1	(255,228,225)	#ffe4e1	
mistyrose2	(238,213,210)	#eed5d2	
mistyrose3	(205,183,181)	#cdb7b5	
mistyrose4	(139,125,123)	#8b7d7b	
moccasin	(255,228,181)	#ffe4b5	
navajowhite	(255,222,173)	#ffdead	
navajowhite1	(255,222,173)	#ffdead	
navajowhite2	(238,207,161)	#eecfa1	
navajowhite3	(205,179,139)	#cdb38b	
navajowhite4	(139,121,94)	#8b795e	
navy	(0,0,128)	#000080	
oldlace	(253,245,230)	#fdf5e6	
olivedrab	(107,142,35)	#6b8e23	
olivedrab1	(192,255,62)	#c0ff3e	
olivedrab2	(179,238,58)	#b3ee3a	
olivedrab3	(154,205,50)	#9acd32	
olivedrab4	(105,139,34)	#698b22	
orange	(255,165,0)	#ffa500	
orange1	(255,165,0)	#ffa500	
orange2	(238,154,0)	#ee9a00	
orange3	(205,133,0)	#cd8500	
orange4	(139,90,0)	#8b5a00	
orangered	(255,69,0)	#ff4500	
orangered1	(255,69,0)	#ff4500	
orangered2	(238,64,0)	#ee4000	
orangered3	(205,55,0)	#cd3700	
orangered4	(139,37,0)	#8b2500	
orchid	(218,112,214)	#da70d6	
orchid1	(255,131,250)	#ff83fa	
orchid2	(238,122,233)	#ee7ae9	
orchid3	(205,105,201)	#cd69c9	
orchid4	(139,71,137)	#8b4789	
palegoldenrod	(238,232,170)	#eee8aa	

Color name	RGB Triple	Hex	Color
palegreen	(152,251,152)	#98fb98	
palegreen1	(154,255,154)	#9aff9a	
palegreen2	(144,238,144)	#90ee90	
palegreen3	(124,205,124)	#7ccd7c	
palegreen4	(84,139,84)	#548b54	
paleturquoise	(175,238,238)	#afeeee	
paleturquoise1	(187,255,255)	#bbffff	
paleturquoise2	(174,238,238)	#aeeeeee	
paleturquoise3	(150,205,205)	#96cdcd	
paleturquoise4	(102,139,139)	#668b8b	
palevioletred	(219,112,147)	#db7093	
palevioletred1	(255,130,171)	#ff82ab	
palevioletred2	(238,121,159)	#ee799f	
palevioletred3	(205,104,137)	#cd6889	
palevioletred4	(139,71,93)	#8b475d	
papayawhip	(255,239,213)	#ffefed5	
peachPuff1	(255,218,185)	#ffdab9	
peachpuff	(255,218,185)	#ffdab9	
peachpuff2	(238,203,173)	#eecbad	
peachpuff3	(205,175,149)	#cdaf95	
peachpuff4	(139,119,101)	#8b7765	
peru	(205,133,63)	#cd853f	
pigmentindigo	(75,0,130)	#4b0082	
pink	(255,192,203)	#ffcc0cb	
pink1	(255,181,197)	#fffb5c5	
pink2	(238,169,184)	#eea9b8	
pink3	(205,145,158)	#cd919e	
pink4	(139,99,108)	#8b636c	
plum	(221,160,221)	#ddaa0dd	
plum1	(255,187,255)	#ffbbff	
plum2	(238,174,238)	#eeaeeee	
plum3	(205,150,205)	#cd96cd	
plum4	(139,102,139)	#8b668b	
powderblue	(176,224,230)	#b0e0e6	
purple	(160,32,240)	#a020f0	
purple1	(155,48,255)	#9b30ff	
purple2	(145,44,238)	#912cee	
purple3	(125,38,205)	#7d26cd	

Color name	RGB Triple	Hex	Color
purple4	(85,26,139)	#551a8b	
red	(255,0,0)	#ff0000	
rosybrown	(188,143,143)	#bc8f8f	
rosybrown1	(255,193,193)	#ffc1c1	
rosybrown2	(238,180,180)	#eeb4b4	
rosybrown3	(205,155,155)	#cd9b9b	
rosybrown4	(139,105,105)	#8b6969	
royalblue	(65,105,225)	#4169e1	
royalblue1	(72,118,255)	#4876ff	
royalblue2	(67,110,238)	#436eee	
royalblue3	(58,95,205)	#3a5fcfd	
royalblue4	(39,64,139)	#27408b	
saddlebrown	(139,69,19)	#8b4513	
salmon	(250,128,114)	#fa8072	
salmon1	(255,140,105)	#ff8c69	
salmon2	(238,130,98)	#ee8262	
salmon3	(205,112,84)	#cd7054	
salmon4	(139,76,57)	#8b4c39	
sandybrown	(244,164,96)	#f4a460	
seagreen	(46,139,87)	#2e8b57	
seagreen1	(84,255,159)	#54ff9f	
seagreen2	(78,238,148)	#4eee94	
seagreen3	(67,205,128)	#43cd80	
seagreen4	(46,139,87)	#2e8b57	
seashell	(255,245,238)	#ffff5ee	
seashell1	(255,245,238)	#fff5ee	
seashell2	(238,229,222)	#eee5de	
seashell3	(205,197,191)	#cdc5bf	
seashell4	(139,134,130)	#8b8682	
sienna	(160,82,45)	#a0522d	
sienna1	(255,130,71)	#ff8247	
sienna2	(238,121,66)	#ee7942	
sienna3	(205,104,57)	#cd6839	
sienna4	(139,71,38)	#8b4726	
silver	(192,192,192)	#c0c0c0	
skyblue	(135,206,235)	#87ceeb	
skyblue1	(135,206,255)	#87ceff	
skyblue2	(126,192,238)	#7ec0ee	

Color name	RGB Triple	Hex	Color
skyblue3	(108,166,205)	#6ca6cd	
skyblue4	(74,112,139)	#4a708b	
slateblue	(106,90,205)	#6a5acd	
slateblue1	(131,111,255)	#836fff	
slateblue2	(122,103,238)	#7a67ee	
slateblue3	(105,89,205)	#6959cd	
slateblue4	(71,60,139)	#473c8b	
slategray	(112,128,144)	#708090	
slategray1	(198,226,255)	#c6e2ff	
slategray2	(185,211,238)	#b9d3ee	
slategray3	(159,182,205)	#9fb6cd	
slategray4	(108,123,139)	#6c7b8b	
snow1	(255,250,250)	#ffffafa	
snow2	(238,233,233)	#eee9e9	
snow3	(205,201,201)	#cdc9c9	
snow4	(139,137,137)	#8b8989	
springgreen	(0,255,127)	#00ff7f	
springgreen1	(0,255,127)	#00ff7f	
springgreen2	(0,238,118)	#00ee76	
springgreen3	(0,205,102)	#00cd66	
springgreen4	(0,139,69)	#008b45	
steelblue	(70,130,180)	#4682b4	
steelblue1	(99,184,255)	#63b8ff	
steelblue2	(92,172,238)	#5cacée	
steelblue3	(79,148,205)	#4f94cd	
steelblue4	(54,100,139)	#36648b	
tan	(210,180,140)	#d2b48c	
tan1	(255,165,79)	#ffa54f	
tan2	(238,154,73)	#ee9a49	
tan3	(205,133,63)	#cd853f	
tan4	(139,90,43)	#8b5a2b	
teal	(0,128,128)	#008080	
thistle	(216,191,216)	#d8bfd8	
thistle1	(255,225,255)	#fffe1fff	
thistle2	(238,210,238)	#eed2ee	
thistle3	(205,181,205)	#cdb5cd	
thistle4	(139,123,139)	#8b7b8b	
tomato	(255,99,71)	#ff6347	

Color name	RGB Triple	Hex	Color
tomato1	(255,99,71)	#ff6347	
tomato2	(238,92,66)	#ee5c42	
tomato3	(205,79,57)	#cd4f39	
tomato4	(139,54,38)	#8b3626	
turquoise	(64,224,208)	#40e0d0	
turquoise1	(0,245,255)	#00f5ff	
turquoise2	(0,229,238)	#00e5ee	
turquoise3	(0,197,205)	#00c5cd	
turquoise4	(0,134,139)	#00868b	
violet	(238,130,238)	#ee82ee	
violetred	(208,32,144)	#d02090	
violetred1	(255,62,150)	#ff3e96	
violetred2	(238,58,140)	#ee3a8c	
violetred3	(205,50,120)	#cd3278	
violetred4	(139,34,82)	#8b2252	
wheat	(245,222,179)	#f5deb3	
wheat1	(255,231,186)	#ff e7ba	
wheat2	(238,216,174)	#eed8ae	
wheat3	(205,186,150)	#cdba96	
wheat4	(139,126,102)	#8b7e66	
white	(255,255,255)	#ffffff	
whitesmoke	(245,245,245)	#f5f5f5	
yellow	(255,255,0)	#ffff00	
yellow1	(255,255,0)	#ffff00	
yellow2	(238,238,0)	#eeee00	
yellow3	(205,205,0)	#cdcd00	
yellow4	(139,139,0)	#8b8b00	
yellowgreen	(154,205,50)	#9acd32	

# Appendix E. Available plot marks

Plot marks can be added to almost all linear (cartesian) graph types. They are used to emphasise the actual data points in the graph. The plot mark property is available as the instance variable "\$mark" for the plot types that support plot marks.

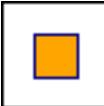
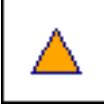
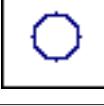
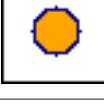
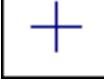
## E.1. Built in basic plot marks

To embellishment a line plot with one of the built-in basic plot marks, say a red diamond mark, at each data point the following line has to be added to the graph script

```
<?php
$lineplot->mark->SetType(MARK_DIAMOND);
$lineplot->mark->SetFillColor('red');
?>
```

The rest of the basic plot marks are handled analogues.

**Table E.1. Built in line based plot marks**

Displayed plot mark	Symbolic name
	MARK_SQUARE
	MARK_UTRIANGLE
	MARK_DTRIANGLE
	MARK_DIAMOND
	MARK_CIRCLE
	MARK_FILLED CIRCLE
	MARK_CROSS

Displayed plot mark	Symbolic name
	MARK_STAR
	MARK_X
	MARK_LEFTTRIANGLE
	MARK_RIGHTTRIANGLE
	MARK_FLASH

## E.2. Built in image plot marks

Since the image based plot marks only supports some colors (since there has to be a unique image for each color) the following sections shows for each major image plot mark what colors are available. The image shows the natural size of the plot mark, i.e. the scale factor is =1. Only the round balls are available natively in three different sizes. Even though it is possible to scale up all the images arbitrarily it will cause the images to become pixelated.

To use one of the built in image plot mark two parameters has to be given to the `SetType()` method. The basic class of plot mark and the wanted color. For example to use a green square as plot marks the following line would have to be added

```
<?php
$lineplot->mark->SetType(MARK_IMG_SQUARE, 'green');
?>
```

If a non-supported color is specified an error will be thrown.

### E.2.1. Image plot mark: Balls

Figure E.1. Small size



Figure E.2. Medium size

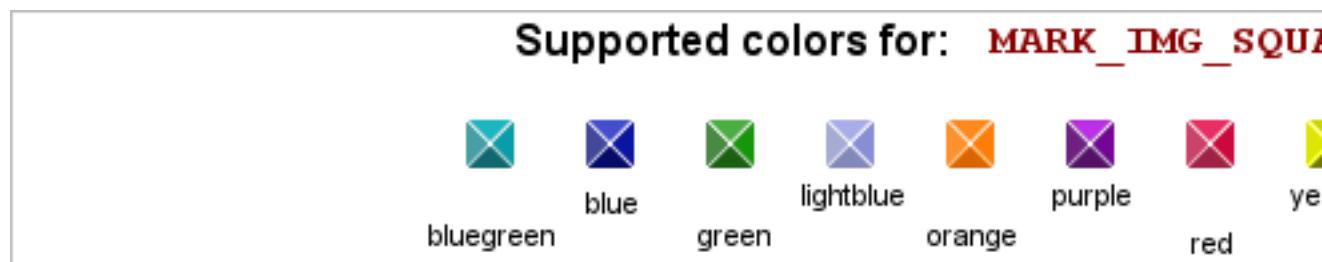


Figure E.3. Large size



## E.2.2. Image plot mark: Squares

Figure E.4. Standard size



## E.2.3. Image plot mark: Diamonds

Figure E.5. Standard size



## E.2.4. Image plot mark: Stars

Figure E.6. Standard size



## E.2.5. Image plot mark: Bevels

Figure E.7. Standard size



## E.2.6. Image plot mark: Pushpins

Figure E.8. Standard size



---

# Appendix F. List of all country flags

All country flags are included in a pre-compiled efficient format with the library. In addition the flags can be scaled arbitrarily and used as either background images or as plot marks. Flags are specified by either it's index or by the using the whole or a unique part of the country name.

The list of all flags are in simple alphabetical order, i.e. by the full name including governance prefix e.g. "Republic". The size of the flags in this table has been chosen as FLAGSIZE2

## Caution

The dynamics of world politics and geographic boundaries will no doubt make the list of included countries obsolete almost at the same time the library is released. The intention is that all currently known country and similar geographic entities should be included. Any missing flags shall and can not be interpreted as any political stand it is merely a consequence of political changes since the library was released or a possible oversight or simply a mistake either human or algorithmically in how the country flags are produced.

For more information on how to use country flags see Section 14.15, “Adding images and country flags to the background of the graph”

It is also possible to easily create a flags for usage outside graphs. The following script takes an index and flag core size as URL arguments and returns the chosen flag.

```
<?php
require_once 'jpgraph.php';
require_once 'jpgraph_flags.php';

if(empty($_GET['size'])) {
 $size = FLAGSIZE1;
}
else {
 $size = $_GET['size'];
}

if(empty($_GET['idx'])) {
 $idx = 'swdn';
}
else {
 $idx = $_GET['idx'];
}

$flags = new FlagImages($size) ;
$img = $flags->GetImgByIdx($idx);
header ("Content-type: image/png");
ImagePng ($img);
?>
```

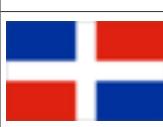
**Table F.1. List of all country flags sorted by country name**

Country name	Short name	Index	Flag
Afghanistan	"afgh"	0	
Alderney	"alde"	3	
Antarctica	"anta"	8	
Arab Republic of Egypt	"egyp"	63	
Argentine Republic	"arge"	9	
Aruba	"arub"	12	
Autonomous Region of Tibet	"tibe"	210	
Azerbaijani Republic	"azer"	15	
Bangladesh	"bngl"	16	
Barbados	"brbd"	28	
Belize	"blze"	24	
Bolivarian Republic of Venezuela	"venz"	231	
British Antarctic Territory	"bant"	17	

List of all country flags

Country name	Short name	Index	Flag
British Indian Ocean Territory	"brin"	29	
British Overseas Territory of Anguilla	"angu"	7	
British Overseas Territory of Bermuda	"berm"	19	
British Overseas Territory of Montserrat	"mont"	138	
British Overseas Territory of the Cayman Islands	"cyis"	55	
British Overseas Territory of the Falkland Islands	"fais"	71	
British Overseas Territory of the Pitcairn Islands	"piis"	171	
Brunei Darussalam	"brun"	30	
Caribbean Community	"cari"	39	
Central African Republic	"cafri"	35	
Commonwealth	"comn"	49	
Commonwealth of Australia	"astrl"	13	
Commonwealth of Dominica	"domn"	60	
Commonwealth of Independent States	"cins"	45	
Commonwealth of Puerto Rico	"purc"	174	

List of all country flags

Country name	Short name	Index	Flag
Commonwealth of the Bahamas	"bhms"	20	
Commonwealth of the Northern Mariana Islands	"nmar"	155	
Cook Islands	"ckis"	46	
Democratic Republic of Congo	"zare"	233	
Democratic Socialist Republic of Sri Lanka	"srsla"	196	
Democratic and Popular Republic of Algeria	"alge"	4	
Dominican Republic	"dore"	61	
Dominion of Canada	"cana"	38	
England	"engl"	65	
Ethiopia	"ethp"	69	
European Union	"euun"	70	
Federal Republic of Germany	"germ"	81	
Federal Republic of Nigeria	"ngr'a"	151	
Federated States of Micronesia	"micr"	133	

List of all country flags

Country name	Short name	Index	Flag
Federation of Malaysia	"mals"	126	
Federative Republic of Brazil	"braz"	27	
French Republic	"fran"	76	
Gabonese Republic	"gabn"	78	
Gibraltar	"gibr"	83	
Grand Duchy of Luxembourg	"luxe"	120	
Hashemite Kingdom of Jordan	"jord"	108	
Hellenic Republic	"grec"	84	
Hong Kong Special Administrative Region	"hokn"	92	
Independent State of Papua New Guinea	"pang"	167	
International Committee of the Red Cross	"icrc"	96	
International Federation of Vexillological Associations	"fiav"	72	
International Olympic Committee	"olym"	163	

List of all country flags

Country name	Short name	Index	Flag
Islamic Republic of Mauritania	"maur"	130	
Islamic Republic of Pakistan	"paks"	165	
Isle of Man	"isma"	102	
Italian Republic	"ital"	104	
Jamaica	"jama"	105	
Japan	"japa"	106	
Kingdom of Bahrain	"bhrn"	21	
Kingdom of Belgium	"belg"	18	
Kingdom of Cambodia	"camb"	36	
Kingdom of Denmark	"dennm"	58	
Kingdom of Lesotho	"lest"	116	
Kingdom of Morocco	"morc"	139	

List of all country flags

Country name	Short name	Index	Flag
Kingdom of Nepal	"nepa"	148	
Kingdom of Norway	"norw"	158	
Kingdom of Saudi Arabia	"saar"	179	
Kingdom of Spain	"span"	194	
Kingdom of Swaziland	"szld"	204	
Kingdom of Sweden	"swdn"	201	
Kingdom of Thailand	"thal"	209	
Kingdom of Tonga	"tong"	214	
Kingdom of the Netherlands	"neth"	149	
Kyrgyz Republic	"kyrg"	113	
League of Arab States	"arle"	10	
Lebanese Republic	"leba"	115	

List of all country flags

Country name	Short name	Index	Flag
Macao Special Administrative Region	"maca"	121	
Midway Islands	"miis"	134	
Netherlands Antilles	"nean"	147	
New Zealand	"nwze"	160	
Niue	"niue"	154	
Nordic Council	"nord"	157	
Nordic Sami Conference	"sami"	181	
North Atlantic Treaty Organization	"nato"	144	
Organization of African Unity	"oaun"	162	
Organization of American States	"oast"	161	
Organization of the Islamic Conference	"isco"	101	
Oriental Republic of Uruguay	"urgy"	226	
Overseas Department of French Guiana	"frgu"	77	

List of all country flags

Country name	Short name	Index	Flag
Overseas Department of Guadeloupe	"guad"	86	
Overseas Department of Martinique	"mart"	129	
Overseas Territory of the British Virgin Islands	"bvis"	34	
Peoples Republic of China	"chin"	43	
Principality of Andorra	"andr"	6	
Principality of Liechtenstein	"liec"	118	
Principality of Monaco	"mona"	136	
Principality of Seborga	"sebo"	184	
Province of Northern Ireland	"noir"	156	
Republic of Albania	"alba"	2	
Republic of Angola	"agla"	1	
Republic of Armenia	"arme"	11	
Republic of Austria	"aust"	14	

List of all country flags

Country name	Short name	Index	Flag
Republic of Belarus	"blru"	22	
Republic of Benin	"bnin"	25	
Republic of Bolivia	"blva"	23	
Republic of Botswana	"bots"	26	
Republic of Bulgaria	"bulg"	32	
Republic of Burkina	"bufa"	31	
Republic of Burundi	"buru"	33	
Republic of Cameroon	"came"	37	
Republic of Cape Verde	"cave"	40	
Republic of Chad	"chad"	41	
Republic of Chile	"chil"	42	
Republic of China	"taiw"	205	
Republic of Colombia	"clmb"	47	

List of all country flags

Country name	Short name	Index	Flag
Republic of Costa Rica	"corc"	52	
Republic of Croatia	"croa"	53	
Republic of Cuba	"cuba"	54	
Republic of Cyprus	"cypr"	56	
Republic of Djibouti	"djib"	59	
Republic of Ecuador	"ecua"	62	
Republic of El Salvador	"elsa"	64	
Republic of Equatorial Guinea	"eqgu"	66	
Republic of Estonia	"estn"	68	
Republic of Fiji	"fiji"	73	
Republic of Finland	"finl"	74	
Republic of Georgia	"geor"	80	
Republic of Ghana	"ghan"	82	
Republic of Guatemala	"guat"	88	

List of all country flags

Country name	Short name	Index	Flag
Republic of Guinea	"guin"	90	
Republic of Haiti	"hait"	91	
Republic of Honduras	"hond"	93	
Republic of Hungary	"hung"	94	
Republic of Iceland	"icel"	95	
Republic of India	"inda"	97	
Republic of Indonesia	"indn"	98	
Republic of Iraq	"iraq"	99	
Republic of Ireland	"irel"	100	
Republic of Kazakhstan	"kazk"	109	
Republic of Kenya	"keny"	110	
Republic of Kiribati	"kirb"	111	
Republic of Korea	"skor"	188	

List of all country flags

Country name	Short name	Index	Flag
Republic of Latvia	"latv"	114	
Republic of Liberia	"libe"	117	
Republic of Lithuania	"lith"	119	
Republic of Macedonia	"mace"	122	
Republic of Madagascar	"mada"	123	
Republic of Malawi	"malw"	128	
Republic of Mali	"mali"	125	
Republic of Malta	"malt"	127	
Republic of Mauritius	"mrts"	141	
Republic of Moldova	"mold"	135	
Republic of Mongolia	"mong"	137	
Republic of Mozambique	"moza"	140	
Republic of Namibia	"namb"	143	

List of all country flags

Country name	Short name	Index	Flag
Republic of Nauru	"naur"	145	
Republic of Nicaragua	"nica"	152	
Republic of Niger	"nigr"	153	
Republic of Palau	"pala"	166	
Republic of Paraguay	"para"	168	
Republic of Peru	"peru"	169	
Republic of Poland	"pola"	172	
Republic of Portugal	"port"	173	
Republic of Rwanda	"rwan"	178	
Republic of San Marino	"sama"	180	
Republic of Serbia	"serb"	185	
Republic of Sierra Leone	"sile"	186	
Republic of Singapore	"sing"	187	

List of all country flags

Country name	Short name	Index	Flag
Republic of Slovenia	"slva"	189	
Republic of Somaliland	"smlnd"	191	
Republic of South Africa	"soaf"	192	
Republic of Suriname	"surn"	199	
Republic of Tajikistan	"tajk"	207	
Republic of Tunisia	"tuns"	217	
Republic of Turkey	"turk"	218	
Republic of Uganda	"ugan"	221	
Republic of Uzbekistan	"uzblk"	228	
Republic of Vanuatu	"vant"	230	
Republic of Yemen	"yemn"	232	
Republic of Zimbabwe	"zbwe"	234	
Republic of the Congo	"cong"	51	

List of all country flags

Country name	Short name	Index	Flag
Republic of the Gambia	"gamb"	79	
Republic of the Marshall Islands	"mais"	124	
Republic of the Philippines	"phil"	170	
Republic of the Sudan	"suda"	198	
Romania	"rmna"	177	
Russian Federation	"russ"	176	
Saint Lucia	"stlu"	197	
Sark	"sark"	182	
Scotland	"scot"	183	
Secretariat of the Pacific Community	"spco"	195	
Slovak Republic	"svka"	200	
Solomon Islands	"sois"	193	
Somali Republic	"smla"	190	

List of all country flags

Country name	Short name	Index	Flag
State of Eritrea	"erit"	67	
State of Grenada	"gren"	85	
State of Israel	"isra"	103	
State of Kuwait	"kuwa"	112	
State of Qatar	"qata"	175	
State of the Vatican City	"vacy"	229	
Sultanate of Oman	"oman"	164	
Swiss Confederation	"swit"	202	
Syrian Arab Republic	"syra"	203	
Taiwan	"taiw"	206	
Territorial Collectivity of Mayotte	"mayt"	131	
Territory of American Samoa	"amsa"	5	

List of all country flags

Country name	Short name	Index	Flag
Territory of Christmas Island	"chms"	44	
Territory of Cocos Islands	"cois"	48	
Territory of French Polynesia	"fpol"	75	
Territory of Guam	"guam"	87	
Territory of New Caledonia and Dependencies	"nwca"	159	
Territory of Norfolk Island	"nfis"	150	
The Bailiwick of Guernsey	"guer"	89	
The Bailiwick of Jersey	"jers"	107	
The Czech Republic	"czec"	57	
Togolese Republic	"togo"	212	
Tokelau	"toke"	213	
Tristan da Cunha	"trdc"	215	
Tromelin	"tris"	216	
Turkish Republic of Northern Cyprus	"ncyp"	146	

List of all country flags

Country name	Short name	Index	Flag
Turkmenistan	"tkst"	211	
Tuvalu	"tuva"	219	
Ukraine	"ukrn"	222	
Union of Myanmar	"myan"	142	
Union of the Comoros	"como"	50	
United Arab Emirates	"uaem"	220	
United Kingdom of Great Britain	"unkg"	223	
United Mexican States	"mexc"	132	
United Nations	"unna"	224	
United Republic of Tanzania	"tanz"	208	
United States of America	"unst"	225	
Virgin Islands of the United States	"usvs"	227	

---

# Appendix G. List of files included in the library

In the distribution these files are stored directly under the `src` directory in the distribution.

In the pro-version of the library there are also two alternative directories which contains variations of the core `src` directory. These additional directories are:

- |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| stripped-src   | These file are suitable to be deployed on a production server. No examples or demo applications are included here and in addition all files have white spaces and comments stripped which makes them significantly smaller and hence reduce the load time of the files on a production server.                                                                                                                                                                                      |
| phpExpress-src | These files are intended to be used with the PHP Accelerator from NuSphere. This accelerometer can be downloaded free of charge from NuSphere corporation [ <a href="http://www.nusphere.com">http://www.nusphere.com</a> ]. For a production server this is the recommended installation since it significantly reduces the load on the production server. See Chapter 11, <i>NuSphere PHP accelerator</i> for information on how to install the freely available PHP Accelerator. |

**Table G.1. List of files included in the library**

File name	Only Pro Version	Description
flags.dat		Raw data for country flag in maximum size
flags_thumb100x100.dat		Raw data for country flag in size 100x100
flags_thumb35x35.dat		Raw data for country flag in size 35x35
flags_thumb60x60.dat		Raw data for country flag in size 60x60
gd_image.inc.php		Interface classes to the low level GD library
imgdata_balls.inc.php		Stored image data for marker images
imgdata_bevels.inc.php		Stored image data for marker images
imgdata_diamonds.inc.php		Stored image data for marker images
imgdata_pushpins.inc.php		Stored image data for marker images
imgdata_squares.inc.php		Stored image data for marker images
imgdata_stars.inc.php		Stored image data for marker images
jpg-config.inc.php		Configuration file for the library
jpgraph_antispam-digits.php		Image data for CAPTCHA digits
jpgraph_antispam.php		Extension module: CAPTCHA figures
jpgraph_bar.php		Extension module: Bargraphs
jpgraph_canvas.php		Extension module: Canvas graphs
jpgraph_canvtools.php		Extension module: Tools for canvas graphs
jpgraph_date.php		Extension module: Date scale handling
jpgraph_errhandler.inc.php		Core module: Error handler

<b>File name</b>	<b>Only Pro Version</b>	<b>Description</b>
jpgraph_error.php		Extension module: Error plots
jpgraph_flags.php		Extension module: Country flags
jpgraph_gantt.php		Extension module: Gantt chart
jpgraph_gb2312.php		Extension module: GB2312 (Chinese) encoding
jpgraph_gradient.php		Extension module: Gradient fill
jpgraph_iconplot.php		Extension module: Icon images in plots
jpgraph_imgtrans.php		Extension module: Image transformation
jpgraph_led.php		Extension module: LED digits/numbers
jpgraph_legend.inc.php		Core module: Legend handling
jpgraph_line.php		Extension module: Line plot
jpgraph_log.php		Extension module: Log scale
jpgraph_mgraph.php		Extension module: Multigraph canvas
jpgraph.php		Core module: JpGraph core module
jpgraph_pie3d.php		Extension module: 3D Pie plot
jpgraph_pie.php		Extension module: 2D Pie plot
jpgraph_plotband.php		Extension module: Plot bands (filled areas in graphs)
jpgraph_plotmark.inc.php		Extension module: Plotmark for line plots
jpgraph_polar.php		Extension module: Polar plots
jpgraph_radar.php		Extension module: Radar plot
jpgraph_regstat.php		Extension module: Spline and Bezier curves
jpgraph_rgb.inc.php		Core module: Color handling
jpgraph_scatter.php		Extension module: Scatter plot
jpgraph_stock.php		Extension module: Stock charts
jpgraph_text.inc.php		Core module: Text handling
jpgraph_ttf.inc.php		Core module: TTF font handling
jpgraph_utils.inc.php		Extension module: Various utility classes
jpgraph_layout_vh.inc.php		Extension utility: classes for automatic layout classes
jpgraph_odo.php	*	Extension module: Odometer plot
jpgraph_windrose.php	*	Extension module: Windrose plots
jpgraph_table.php	*	Extension module: Graphic tables
jpgraph_barcode.php	*	Extension module: 1D Barcodes
jpgraph_matrix.php	*	Extension module: Matrix plots

In addition to the files above the distribution contains the following directories

**Table G.2. List of subdirectories in main `src` directory**

Directory	Only Version	Pro Ver- sion	Description
Examples			JpGraph Example directory
lang			Localized error messages
barcode	*		1D Barcodes
datamatrix	*		2D Barcode Datamatrix directory
pdf417	*		2D Barcode PDF417 directory
QR	*		2D Barcode QR (Quick response) directory
table_examples	*		Table examples
windrose_examples	*		Windrose examples
odometer_examples	*		Odometer examples
matrix_examples	*		Matrix plot examples

---

# Appendix H. Error messages

This list have all error messages that can be thrown by the library. In addition to the English locale there is also a German translation (See Section 6.2.1, “Localizing error messages”). Most error messages takes one (or several) argument that gives additional specific information and the placeholder in the error messages shows where this additional information should be placed.

For more general information on error handling we refer to Chapter 6, *Error handling*

## H.1. Core error messages

The following tables lists all error messages that can be generated by standard core graphs types

**Table H.1. English error messages**

Error code	Error message
10	<table border="1"><tr><td style="color:darkred; font-size:1.2em;"><b>JpGraph Error:</b> HTTP headers have already been sent. Caused by output from file <b>%s</b> at line <b>%d</b>.</td></tr><tr><td><b>Explanation:</b> HTTP headers have already been sent back to the browser indicating the data as text before the library got a chance to send it's image HTTP header to this browser. This makes it impossible for the library to send back image data to the browser (since that would be interpreted as text by the browser and show up as junk text).<p>Most likely you have some text in your script before the call to <i>Graph::Stroke()</i>. If this texts gets sent back to the browser the browser will assume that all data is plain text. Look for any text, even spaces and newlines, that might have been sent back to the browser. <p>For example it is a common mistake to leave a blank line before the opening "<b>&lt;?php</b>".</td></tr></table>
11	No path specified for CACHE_DIR. Please specify CACHE_DIR manually in jpg-config.inc
12	No path specified for TTF_DIR and path can not be determined automatically. Please specify TTF_DIR manually (in jpg-config.inc).
13	The installed PHP version (%s) is not compatible with this release of the library. The library requires at least PHP version %s
1001	Unknown encoder specification: %s
1002	Data validation failed. Can't encode [%s] using encoding "%s"
1003	Internal encoding error. Trying to encode %s is not possible in Code 128
1004	Internal barcode error. Unknown UPC-E encoding type: %s
1005	Internal error. Can't encode character tuple (%s, %s) in Code-128 charset C
1006	Internal encoding error for CODE 128. Trying to encode control character in CHARSET != A
1007	Internal encoding error for CODE 128. Trying to encode DEL in CHARSET != B
1008	Internal encoding error for CODE 128. Trying to encode small letters in CHARSET != B
1009	Encoding using CODE 93 is not yet supported.
1010	Encoding using POSTNET is not yet supported.
1011	Non supported barcode backend for type %s
2001	Number of colors is not the same as the number of patterns in BarPlot::SetPattern()
2002	Unknown pattern specified in call to BarPlot::SetPattern()

Error code	Error message
2003	Number of X and Y points are not equal. Number of X-points: %d Number of Y-points: %d
2004	All values for a barplot must be numeric. You have specified value nr [%d] == %s
2005	You have specified an empty array for shadow colors in the bar plot.
2006	Unknown position for values on bars : %s
2007	Cannot create GroupBarPlot from empty plot array.
2008	Group bar plot element nbr %d is undefined or empty.
2009	One of the objects submitted to GroupBar is not a BarPlot. Make sure that you create the GroupBar plot from an array of BarPlot or AccBarPlot objects. (Class = %s)
2010	Cannot create AccBarPlot from empty plot array.
2011	Acc bar plot element nbr %d is undefined or empty.
2012	One of the objects submitted to AccBar is not a BarPlot. Make sure that you create the AccBar plot from an array of BarPlot objects. (Class=%s)
2013	You have specified an empty array for shadow colors in the bar plot.
2014	Number of datapoints for each data set in accbarplot must be the same
2015	Individual bar plots in an AccBarPlot or GroupBarPlot can not have specified X-coordinates
3001	It is only possible to use either SetDateAlign() or SetTimeAlign() but not both
4002	Error in input data to LineErrorPlot. Number of data points must be a multiple of 3
5001	Unknown flag size (%d).
5002	Flag index %s does not exist.
5003	Invalid ordinal number (%d) specified for flag index.
5004	The (partial) country name %s does not have a corresponding flag image. The flag may still exist but under another name, e.g. instead of "usa" try "united states".
6001	Internal error. Height for ActivityTitles is < 0
6002	You can't specify negative sizes for Gantt graph dimensions. Use 0 to indicate that you want the library to automatically determine a dimension.
6003	Invalid format for Constrain parameter at index=%d in CreateSimple(). Parameter must start with index 0 and contain arrays of (Row,Constrain-To,Constrain-Type)
6004	Invalid format for Progress parameter at index=%d in CreateSimple(). Parameter must start with index 0 and contain arrays of (Row,Progress)
6005	SetScale() is not meaningful with Gantt charts.
6006	Cannot autoscale Gantt chart. No dated activities exist. [GetBarMinMax() start >= n]
6007	Sanity check for automatic Gantt chart size failed. Either the width (=%d) or height (=%d) is larger than MAX_GANTTIMG_SIZE. This could potentially be caused by a wrong date in one of the activities.
6008	You have specified a constrain from row=%d to row=%d which does not have any activity
6009	Unknown constrain type specified from row=%d to row=%d
6010	Illegal icon index for Gantt builtin icon [%d]
6011	Argument to IconImage must be string or integer
6012	Unknown type in Gantt object title specification

Error code	Error message
6015	Illegal vertical position %d
6016	Date string (%s) specified for Gantt activity can not be interpreted. Please make sure it is a valid time string, e.g. 2005-04-23 13:30
6017	Unknown date format in GanttScale (%s).
6018	Interval for minutes must divide the hour evenly, e.g. 1,5,10,12,15,20,30 etc You have specified an interval of %d minutes.
6019	The available width (%d) for minutes are to small for this scale to be displayed. Please use auto-sizing or increase the width of the graph.
6020	Interval for hours must divide the day evenly, e.g. 0:30, 1:00, 1:30, 4:00 etc. You have specified an interval of %d
6021	Unknown formatting style for week.
6022	Gantt scale has not been specified.
6023	If you display both hour and minutes the hour interval must be 1 (Otherwise it doesn't make sense to display minutes).
6024	CSIM Target must be specified as a string. Start of target is: %d
6025	CSIM Alt text must be specified as a string. Start of alt text is: %d
6027	Progress value must in range [0, 1]
6028	Specified height (%d) for gantt bar is out of range.
6029	Offset for vertical line must be in range [0,1]
6030	Unknown arrow direction for link.
6031	Unknown arrow type for link.
6032	Internal error: Unknown path type (=-%d) specified for link.
6033	Array of fonts must contain arrays with 3 elements, i.e. (Family, Style, Size)
7001	Unknown gradient style (-%d).
8001	Mix value for icon must be between 0 and 100.
8002	Anchor position for icons must be one of "top", "bottom", "left", "right" or "center"
8003	It is not possible to specify both an image file and a country flag for the same icon.
8004	In order to use Country flags as icons you must include the "jpgraph_flags.php" file.
9001	Value for image transformation out of bounds. Vanishing point on horizon must be specified as a value between 0 and 1.
10001	LinePlot::SetFilled() is deprecated. Use SetFillColor()
10002	Plot too complicated for fast line Stroke. Use standard Stroke()
10003	Each plot in an accumulated lineplot must have the same number of data points.
11001	Your data contains non-numeric values.
11002	Negative data values can not be used in a log scale.
11003	Your data contains non-numeric values.
11004	Scale error for logarithmic scale. You have a problem with your data values. The max value must be greater than 0. It is mathematically impossible to have 0 in a logarithmic scale.

Error code	Error message
11005	Specifying tick interval for a logarithmic scale is undefined. Remove any calls to SetTextLabelStart() or SetTextTickInterval() on the logarithmic scale.
12001	You are using GD 2.x and are trying to use a background images on a non truecolor image. To use background images with GD 2.x it is necessary to enable truecolor by setting the USE_TRUECOLOR constant to TRUE. Due to a bug in GD 2.0.1 using any truetype fonts with truecolor images will result in very poor quality fonts.
12002	Incorrect file name for MGraph::SetBackgroundImage() : %s Must have a valid image extension (jpg,gif,png) when using auto detection of image type
12003	Unknown file extension (%s) in MGraph::SetBackgroundImage() for filename: %s
12004	The image format of your background image (%s) is not supported in your system configuration.
12005	Can't read background image: %s
12006	Illegal sizes specified for width or height when creating an image, (width=%d, height=%d)
12007	Argument to MGraph::Add() is not a valid GD image handle.
12008	Your PHP (and GD-lib) installation does not appear to support any known graphic formats.
12009	Your PHP installation does not support the chosen graphic format: %s
12010	Can't create or stream image to file %s Check that PHP has enough permission to write a file to the current directory.
12011	Can't create truecolor image. Check that you really have GD2 library installed.
12012	Can't create image. Check that you really have GD2 library installed.
13001	Unknown needle style (%d).
13002	Value for odometer (%f) is outside specified scale [%f,%f]
14001	Pie3D::ShowBorder() . Deprecated function. Use Pie3D::SetEdge() to control the edges around slices.
14002	PiePlot3D::SetAngle() 3D Pie projection angle must be between 5 and 85 degrees.
14003	Internal assertion failed. Pie3D::Pie3DSlice
14004	Slice start angle must be between 0 and 360 degrees.
14005	Pie3D Internal error: Trying to wrap twice when looking for start index
14006	Pie3D Internal Error: Z-Sorting algorithm for 3D Pies is not working properly (2). Trying to wrap twice while stroking.
14007	Width for 3D Pie is 0. Specify a size > 0
15001	PiePlot::SetTheme() Unknown theme: %s
15002	Argument to PiePlot::ExplodeSlice() must be an integer
15003	Argument to PiePlot::Explode() must be an array with integer distances.
15004	Slice start angle must be between 0 and 360 degrees.
15005	PiePlot::SetFont() is deprecated. Use PiePlot->value->SetFont() instead.
15006	PiePlot::SetSize() Radius for pie must either be specified as a fraction [0, 0.5] of the size of the image or as an absolute size in pixels in the range [10, 1000]
15007	PiePlot::SetFontColor() is deprecated. Use PiePlot->value->SetColor() instead.
15008	PiePlot::SetLabelTextType() Type for pie plots must be 0 or 1 (not %d).

Error code	Error message
15009	Illegal pie plot. Sum of all data is zero for Pie Plot
15010	Sum of all data is 0 for Pie.
15011	In order to use image transformation you must include the file jpgraph_imgtrans.php in your script.
16001	Density for pattern must be between 1 and 100. (You tried %f)
16002	No positions specified for pattern.
16003	Unknown pattern specification (%d)
16004	Min value for plotband is larger than specified max value. Please correct.
17001	Polar plots must have an even number of data point. Each data point is a tuple (angle, radius).
17002	Unknown alignment specified for X-axis title. (%s)
17004	Unknown scale type for polar graph. Must be "lin" or "log"
18001	Client side image maps not supported for RadarPlots.
18002	RadarGraph::SupressTickMarks() is deprecated. Use HideTickMarks() instead.
18003	Illegal scale for radarplot (%s). Must be 'lin' or 'log'
18004	Radar Plot size must be between 0.1 and 1. (Your value=%f)
18005	RadarPlot Unsupported Tick density: %d
18006	Minimum data %f (Radar plots should only be used when all data points > 0)
18007	Number of titles does not match number of points in plot.
18008	Each radar plot must have the same number of data points.
19001	Spline: Number of X and Y coordinates must be the same
19002	Invalid input data for spline. Two or more consecutive input X-values are equal. Each input X-value must differ since from a mathematical point of view it must be a one-to-one mapping, i.e. each X-value must correspond to exactly one Y-value.
19003	Bezier: Number of X and Y coordinates must be the same
20001	Fieldplots must have equal number of X and Y points.
20002	Fieldplots must have an angle specified for each X and Y points.
20003	Scatterplot must have equal number of X and Y points.
21001	Data values for Stock charts must contain an even multiple of %d data points.
22001	Total percentage for all windrose legs in a windrose plot can not exceed 100% %n(Current max is: %d)
22002	Graph is too small to have a scale. Please make the graph larger.
22004	Label specification for windrose directions must have 16 values (one for each compass direction).
22005	Line style for radial lines must be on of ("solid", "dotted", "dashed", "longdashed")
22006	Illegal windrose type specified.
22007	To few values for the range legend.
22008	Internal error: Trying to plot free Windrose even though type is not a free windrose
22009	You have specified the same direction twice, once with an angle and once with a compass direction (%f degrees)

Error code	Error message
22010	Direction must either be a numeric value or one of the 16 compass directions
22011	Windrose index must be numeric or direction label. You have specified index=%d
22012	Windrose radial axis specification contains a direction which is not enabled.
22013	You have specified the look&feel for the same compass direction twice, once with text and once with index (Index=%d)
22014	Index for compass direction must be between 0 and 15.
22015	You have specified an undefined Windrose plot type.
22016	Windrose leg index must be numeric or direction label.
22017	Windrose data contains a direction which is not enabled. Please adjust what labels are displayed.
22018	You have specified data for the same compass direction twice, once with text and once with index (Index=%d)
22019	Index for direction must be between 0 and 15. You can't specify angles for a Regular Windplot, only index and compass directions.
22020	Windrose plot is too large to fit the specified Graph size. Please use WindrosePlot::SetSize() to make the plot smaller or increase the size of the Graph in the initial WindroseGraph() call.
22021	It is only possible to add Text, IconPlot or WindrosePlot to a Windrose Graph
23001	This marker "%s" does not exist in color with index: %d
23002	Mark color index too large for marker "%s"
23003	A filename must be specified if you set the mark type to MARK_IMG.
24001	FuncGenerator : No function specified.
24002	FuncGenerator : Syntax error in function specification
24003	DateScaleUtils: Unknown tick type specified in call to GetTicks()
24004	ReadCSV2: Column count mismatch in %s line %d
25001	This PHP installation is not configured with the GD library. Please recompile PHP with GD support to run JpGraph. (Neither function imagetypes() nor imagecreatefromstring() does exist)
25002	Your PHP installation does not seem to have the required GD library. Please see the PHP documentation on how to install and enable the GD library.
25003	General PHP error : At %s:%d : %s
25004	General PHP error : %s
25005	Can't access PHP_SELF, PHP global variable. You can't run PHP from command line if you want to use the 'auto' naming of cache or image files.
25006	Usage of FF_CHINESE (FF_BIG5) font family requires that your PHP setup has the iconv() function. By default this is not compiled into PHP (needs the "--width-iconv" when configured).
25007	You are trying to use the locale (%s) which your PHP installation does not support. Hint: Use " to indicate the default locale for this geographic region.
25008	Image width/height argument in Graph::Graph() must be numeric
25009	You must specify what scale to use with a call to Graph::SetScale()

Error code	Error message
25010	Graph::Add() You tried to add a null plot to the graph.
25011	Graph::AddY2() You tried to add a null plot to the graph.
25012	Graph::AddYN() You tried to add a null plot to the graph.
25013	You can only add standard plots to multiple Y-axis
25014	Graph::AddText() You tried to add a null text to the graph.
25015	Graph::AddLine() You tried to add a null line to the graph.
25016	Graph::AddBand() You tried to add a null band to the graph.
25017	You are using GD 2.x and are trying to use a background images on a non truecolor image. To use background images with GD 2.x it is necessary to enable truecolor by setting the USE_TRUECOLOR constant to TRUE. Due to a bug in GD 2.0.1 using any truetype fonts with truecolor images will result in very poor quality fonts.
25018	Incorrect file name for Graph::SetBackgroundImage() : "%s" Must have a valid image extension (jpg,gif,png) when using auto detection of image type
25019	Unknown file extension (%s) in Graph::SetBackgroundImage() for filename: "%s"
25020	Graph::SetScale(): Specified Max value must be larger than the specified Min value.
25021	Unknown scale specification for Y-scale. (%s)
25022	Unknown scale specification for X-scale. (%s)
25023	Unsupported Y2 axis type: "%s" Must be one of (lin,log,int)
25024	Unsupported Y axis type: "%s" Must be one of (lin,log,int)
25025	Unsupported Tick density: %d
25026	Can't draw unspecified Y-scale. You have either: 1. Specified an Y axis for auto scaling but have not supplied any plots. 2. Specified a scale manually but have forgot to specify the tick steps
25027	Can't open cached CSIM "%s" for reading.
25028	Apache/PHP does not have permission to write to the CSIM cache directory (%s). Check permissions.
25029	Can't write CSIM "%s" for writing. Check free space and permissions.
25030	Missing script name in call to StrokeCSIM(). You must specify the name of the actual image script as the first parameter to StrokeCSIM().
25031	You must specify what scale to use with a call to Graph::SetScale().
25032	No plots for Y-axis nbr:%d
25033	
25034	Can't draw unspecified X-scale. No plots specified.
25035	You have enabled clipping. Clipping is only supported for graphs at 0 or 90 degrees rotation. Please adjust you current angle (=%d degrees) or disable clipping.
25036	Unknown AxisStyle() : %s
25037	The image format of your background image (%s) is not supported in your system configuration.
25038	Background image seems to be of different type (has different file extension) than specified imagetype. Specified: %s File: %s

Error code	Error message
25039	Can't read background image: "%s"
25040	It is not possible to specify both a background image and a background country flag.
25041	In order to use Country flags as backgrounds you must include the "jpgraph_flags.php" file.
25042	Unknown background image layout
25043	Unknown title background style.
25044	Cannot use auto scaling since it is impossible to determine a valid min/max value of the Y-axis (only null values).
25045	Font families FF_HANDWRT and FF_BOOK are no longer available due to copyright problem with these fonts. Fonts can no longer be distributed with JpGraph. Please download fonts from <a href="http://corefonts.sourceforge.net/">http://corefonts.sourceforge.net/</a>
25046	Specified TTF font family (id=%d) is unknown or does not exist. Please note that TTF fonts are not distributed with JpGraph for copyright reasons. You can find the MS TTF WEB-fonts (arial, courier etc) for download at <a href="http://corefonts.sourceforge.net/">http://corefonts.sourceforge.net/</a>
25047	Style %s is not available for font family %s
25048	Unknown font style specification [%s].
25049	Font file "%s" is not readable or does not exist.
25050	First argument to Text::Text() must be a string.
25051	Invalid direction specified for text.
25052	PANIC: Internal error in SuperScript::Stroke(). Unknown vertical alignment for text
25053	PANIC: Internal error in SuperScript::Stroke(). Unknown horizontal alignment for text
25054	Internal error: Unknown grid axis %s
25055	Axis::SetTickDirection() is deprecated. Use Axis::SetTickSide() instead
25056	SetTickLabelMargin() is deprecated. Use Axis::SetLabelMargin() instead.
25057	SetTextTicks() is deprecated. Use SetTextTickInterval() instead.
25058	Text label interval must be specified >= 1.
25059	SetLabelPos() is deprecated. Use Axis::SetLabelSide() instead.
25060	Unknown alignment specified for X-axis title. (%s)
25061	Unknown alignment specified for Y-axis title. (%s)
25062	Labels at an angle are not supported on Y-axis
25063	Ticks::SetPrecision() is deprecated. Use Ticks::SetLabelFormat() (or Ticks::SetFormatCallback()) instead
25064	Minor or major step size is 0. Check that you haven't got an accidental SetTextTicks(0) in your code. If this is not the case you might have stumbled upon a bug in JpGraph. Please report this and if possible include the data that caused the problem
25065	Tick positions must be specified as an array()
25066	When manually specifying tick positions and labels the number of labels must be the same as the number of specified ticks.
25067	Your manually specified scale and ticks is not correct. The scale seems to be too small to hold any of the specified tick marks.

Error code	Error message
25068	A plot has an illegal scale. This could for example be that you are trying to use text auto scaling to draw a line plot with only one point or that the plot area is too small. It could also be that no input data value is numeric (perhaps only '-' or 'x')
25069	Grace must be larger than 0
25070	Either X or Y data arrays contains non-numeric values. Check that the data is really specified as numeric data and not as strings. It is an error to specify data for example as '-2345.2' (using quotes).
25071	You have specified a min value with SetAutoMin() which is larger than the maximum value used for the scale. This is not possible.
25072	You have specified a max value with SetAutoMax() which is smaller than the minimum value used for the scale. This is not possible.
25073	Internal error. Integer scale algorithm comparison out of bound (r=%f)
25074	Internal error. The scale range is negative (%f) [for %s scale] This problem could potentially be caused by trying to use \"illegal\" values in the input data arrays (like trying to send in strings or only NULL values) which causes the auto scaling to fail.
25075	Can't automatically determine ticks since min==max.
25077	Adjustment factor for color must be > 0
25078	Unknown color: %s
25079	Unknown color specification: %s, size=%d
25080	Alpha parameter for color must be between 0.0 and 1.0
25081	Selected graphic format is either not supported or unknown [%s]
25082	Illegal sizes specified for width or height when creating an image, (width=%d, height=%d)
25083	Illegal image size when copying image. Size for copied to image is 1 pixel or less.
25084	Failed to create temporary GD canvas. Possible Out of memory problem.
25085	An image can not be created from the supplied string. It is either in a format not supported or the string is representing an corrupt image.
25086	You only seem to have GD 1.x installed. To enable Alphablending requires GD 2.x or higher. Please install GD or make sure the constant USE_GD2 is specified correctly to reflect your installation. By default it tries to auto detect what version of GD you have installed. On some very rare occasions it may falsely detect GD2 where only GD1 is installed. You must then set USE_GD2 to false.
25087	This PHP build has not been configured with TTF support. You need to recompile your PHP installation with FreeType support.
25088	You have a misconfigured GD font support. The call to imagefontwidth() fails.
25089	You have a misconfigured GD font support. The call to imagefontheight() fails.
25090	Unknown direction specified in call to StrokeBoxedText() [%s]
25091	Internal font does not support drawing text at arbitrary angle. Use TTF fonts instead.
25092	There is either a configuration problem with TrueType or a problem reading font file "%s" Make sure file exists and is in a readable place for the HTTP process. (If 'basedir' restriction is enabled in PHP then the font file must be located in the document root.). It might also be a wrongly installed FreeType library. Try upgrading to at least FreeType 2.1.13 and recompile GD with the correct setup so it can find the new FT library.

Error code	Error message
25093	Can not read font file "%s" in call to Image::GetBBoxTTF. Please make sure that you have set a font before calling this method and that the font is installed in the TTF directory.
25094	Direction for text most be given as an angle between 0 and 90.
25095	Unknown font font family specification.
25096	Can't allocate any more colors in palette image. Image has already allocated maximum of %d colors and the palette is now full. Change to a truecolor image instead
25097	Color specified as empty string in PushColor().
25098	Negative Color stack index. Unmatched call to PopColor()
25099	Parameters for brightness and Contrast out of range [-1,1]
25100	Problem with color palette and your GD setup. Please disable anti-aliasing or use GD2 with true-color. If you have GD2 library installed please make sure that you have set the USE_GD2 constant to true and truecolor is enabled.
25101	Illegal numeric argument to SetLineStyle(): (%d)
25102	Illegal string argument to SetLineStyle(): %s
25103	Illegal argument to SetLineStyle %s
25104	Unknown line style: %s
25105	NULL data specified for a filled polygon. Check that your data is not NULL.
25106	Image::FillToBorder : Can not allocate more colors
25107	Can't write to file "%s". Check that the process running PHP has enough permission.
25108	Can't stream image. This is most likely due to a faulty PHP/GD setup. Try to recompile PHP and use the built-in GD library that comes with PHP.
25109	Your PHP (and GD-lib) installation does not appear to support any known graphic formats. You need to first make sure GD is compiled as a module to PHP. If you also want to use JPEG images you must get the JPEG library. Please see the PHP docs for details.
25110	Your PHP installation does not support the chosen graphic format: %s
25111	Can't delete cached image %s. Permission problem?
25112	Cached imagefile (%s) has file date in the future.
25113	Can't delete cached image "%s". Permission problem?
25114	PHP has not enough permissions to write to the cache file "%s". Please make sure that the user running PHP has write permission for this file if you wan to use the cache system with JpGraph.
25115	Can't set permission for cached image "%s". Permission problem?
25116	Cant open file from cache "%s"
25117	Can't open cached image "%s" for reading.
25118	Can't create directory "%s". Make sure PHP has write permission to this directory.
25119	Can't set permissions for "%s". Permission problems?
25120	Position for legend must be given as percentage in range 0-1
25121	Empty input data array specified for plot. Must have at least one data point.
25122	Stroke() must be implemented by concrete subclass to class Plot
25123	You can't use a text X-scale with specified X-coords. Use a "int" or "lin" scale instead.

Error code	Error message
25124	The input data array must have consecutive values from position 0 and forward. The given y-array starts with empty values (NULL)
25125	Illegal direction for static line
25126	Can't create truecolor image. Check that the GD2 library is properly setup with PHP.
25127	The library has been configured for automatic encoding conversion of Japanese fonts. This requires that PHP has the mb_convert_encoding() function. Your PHP installation lacks this function (PHP needs the "--enable-mbstring" when compiled).
25128	The function imageantialias() is not available in your PHP installation. Use the GD version that comes with PHP and not the standalone version.
25129	Anti-alias can not be used with dashed lines. Please disable anti-alias or use solid lines.
25130	Too small plot area. (%d x %d). With the given image size and margins there is to little space left for the plot. Increase the plot size or reduce the margins.
25131	StrokeBoxedText2() only supports TTF fonts and not built-in bitmap fonts.
25500	Multibyte strings must be enabled in the PHP installation in order to run the LED module so that the function mb_strlen() is available. See PHP documentation for more information.
26000	PDF417: The PDF417 module requires that the PHP installation must support the function bcmod(). This is normally enabled at compile time. See documentation for more information.
26001	PDF417: Number of Columns must be >= 1 and <= 30
26002	PDF417: Error level must be between 0 and 8
26003	PDF417: Invalid format for input data to encode with PDF417
26004	PDF417: Can't encode given data with error level %d and %d columns since it results in too many symbols or more than 90 rows.
26005	PDF417: Can't open file "%s" for writing
26006	PDF417: Internal error. Data files for PDF417 cluster %d is corrupted.
26007	PDF417: Internal error. GetPattern: Illegal Code Value = %d (row=%d)
26008	PDF417: Internal error. Mode not found in mode list!! mode=%d
26009	PDF417: Encode error: Illegal character. Can't encode character with ASCII code=%d
26010	PDF417: Internal error: No input data in decode.
26011	PDF417: Encoding error. Can't use numeric encoding on non-numeric data.
26012	PDF417: Internal error. No input data to decode for Binary compressor.
26013	PDF417: Internal error. Checksum error. Coefficient tables corrupted.
26014	PDF417: Internal error. No data to calculate codewords on.
26015	PDF417: Internal error. State transition table entry 0 is NULL. Entry 1 = (%s)
26016	PDF417: Internal error: Unrecognized state transition mode in decode.
27001	GTextTable: Invalid argument to Set(). Array argument must be 2 dimensional
27002	GTextTable: Invalid argument to Set()
27003	GTextTable: Wrong number of arguments to GTextTable::SetColor()
27004	GTextTable: Specified cell range to be merged is not valid.
27005	GTextTable: Cannot merge already merged cells in the range: (%d,%d) to (%d,%d)

Error code	Error message
27006	GTextTable: Column argument = %d is outside specified table size.
27007	GTextTable: Row argument = %d is outside specified table size.
27008	GTextTable: Column and row size arrays must match the dimensions of the table
27009	GTextTable: Number of table columns or rows are 0. Make sure Init() or Set() is called.
27010	GTextTable: No alignment specified in call to SetAlign()
27011	GTextTable: Unknown alignment specified in SetAlign(). Horizontal=%s, Vertical=%s
27012	GTextTable: Internal error. Invalid alignment specified =%s
27013	GTextTable: Argument to FormatNumber() must be a string.
27014	GTextTable: Table is not initialized with either a call to Set() or Init()
27015	GTextTable: Cell image constraint type must be TIMG_WIDTH or TIMG_HEIGHT
28001	Third argument to Contour must be an array of colors.
28002	Number of colors must equal the number of isobar lines specified
28003	ContourPlot Internal Error: isobarHCrossing: Column index too large (%d)
28004	ContourPlot Internal Error: isobarHCrossing: Row index too large (%d)
28005	ContourPlot Internal Error: isobarVCrossing: Row index too large (%d)
28006	ContourPlot Internal Error: isobarVCrossing: Col index too large (%d)
28007	ContourPlot interpolation factor is too large (>5)
29201	Min range value must be less or equal to max range value for colormaps
29202	The distance between min and max value is too small for numerical precision
29203	Number of color quantification level must be at least %d
29204	Number of colors (%d) is invalid for this colormap. It must be a number that can be written as: %d + k*%d
29205	Colormap specification out of range. Must be an integer in range [0,%d]
29206	Invalid object added to MatrixGraph
29207	Empty input data specified for MatrixPlot
29208	Unknown side specification for matrix labels "%s"

## H.2. QR 2D Barcode error messages

### Note

These error messages are not yet localized

**Table H.2. English error messages**

Error code	Error message
1000	Tilde processing is not yet supported for QR Barcodes.
1001	Inverting the bit pattern is not supported for QR Barcodes.

Error code	Error message
1002	Cannot read data from file %s
1003	Cannot open file %s
1004	Cannot write QR barcode to file %s
1005	Unsupported image format selected. Check your GD installation
1006	Cannot set the selected barcode colors. Check your GD installation and spelling of color name
1007	<p><b>JpGraph Error: HTTP headers have already been sent.</b></p> <p>Caused by output from file %s at line %d.</p> <p>Explanation: HTTP headers have already been sent back to the browser indicating the data as text before the library got a chance to send its image HTTP header to this browser.</p> <p>This makes it impossible for the library to send back image data to the browser (since that would be interpreted as text by the browser and show up as junk text). Most likely you have some text in your script before the call to Graph::Stroke(). If this text gets sent back to the browser the browser will assume that all data is plain text. Look for any text, even spaces and newlines, that might have been sent back to the browser. For example it is a common mistake to leave a blank line before the opening</p>
1008	Could not create the barcode image with image format=%s. Check your GD/PHP installation.
1009	Cannot open log file %s for writing.
1010	Cannot write log info to log file %s.
1100	Internal error: Illegal mask pattern selected
1101	Internal error: Trying to apply masking to functional pattern.
1102	Internal error: applyMaskAndEval(): Found uninitialized module in matrix when applying mask pattern.
1200	Internal error: Was expecting %d bits in version %d to be placed in matrix but got %d bits
1201	Internal error: Trying to position bit outside the matrix x=%d, y=%d, size=%d, bIdx=%d
1202	Internal error: Trying to put data in initialized bit.
1203	Internal error: Mask number for format bits is invalid. (maskidx=%d)
1204	Internal error: Found an uninitialized bit [val=%d] at (%d,%d) when flattening matrix

Error code	Error message
1300	Internal error: QRCapacity::getFormatBits() Was expecting a format in range [0,31] got %d
1301	Internal error: QRCapacity::getVersionBits() Was expecting a version in range [7,40] got %d
1302	Internal error: QRCapacity::_chkVerErr() Was expecting version in range [1,40] and error level in range [0,3] got (%d,%d)
1303	Internal error: QRCapacity::getAlignmentPositions() Expected %d patterns but found %d patterns (len=%d).
1304	Internal error: QRCapacity::%s Was expecting a version in range [1,40] got %d
1400	QR Version must be specified as a value in the range [1,40] got %d
1401	Input data to barcode can not be empty.
1402	Automatic encodation mode was specified but input data looks like specification for manual encodation.
1403	Was expecting an array of arrays as input data for manual encoding.
1404	Each input data array element must consist of two entries. Element \$i has of \$nn entries
1405	Each input data array element must consist of two entries with first entry being the encodation constant and the second element the data string. Element %d is incorrect in this respect.
1406	Was expecting either a string or an array as input data
1407	Manual encodation mode was specified but input data looks like specification for automatic encodation.
1408	Input data too large to fit into one QR Symbol
1409	The selected symbol version %d is too small to fit the specified data and selected error correction level.
1410	Trying to read past the last available codeword in block split.
1411	Internal error: Expected 1 or 2 as the number of block structures.
1412	Internal error: Too many codewords for chosen symbol version. (negative number of pad codewords).
1413	Internal error: splitInBytes: Expected an even number of 8-bit blocks.
1414	Internal error: getCountBits() illegal version number (= %d).

Error code	Error message
1415	Manually specified encodation schema MODE_NUMERIC has no data that can be encoded using this schema.
1416	Manually specified encodation schema MODE_ALPHANUM has no data that can be encoded using this schema.
1417	Manually specified encodation schema MODE_BYTE has no data that can be encoded using this schema.
1418	Unsupported encodation schema specified (%d)
1419	Found character in data stream that cannot be encoded with the selected manual encodation mode.
1420	Encodation using KANJI mode not yet supported.
1421	Internal error: Unsupported encodation mode doAuto().
1422	Found unknown characters in the data stream that can't be encoded with any available encodation mode.
1423	Kanji character set not yet supported.
1424	Internal error: DataStorage:: Unsupported character mode (%d) DataStorage::Remaining()
1425	Internal error: DataStorage:: Trying to extract slice of len=%d (with type=%d) when there are only %d elements left
1426	Internal error: DataStorage:: Trying to read past input data length.
1427	Expected either DIGIT, ALNUM or BYTE but found ASCII code=%d
1428	Internal error: DataStorage::Peek() Trying to peek past input data length.

## H.3. Datamatrix 2D barcode error messages

### Note

These error messages are not yet localized

**Table H.3. English error messages**

Error code	Error message
1	Data is too long to fit specified symbol size
2	The BASE256 data is too long to fit available symbol size
3	Data must have at least three characters for C40 encodation

Error code	Error message
4	Data must have at least three characters for TEXT encodation
5	Internal error: (-5) Trying to read source data past the end
6	Internal error: (-6) Trying to look ahead in data past the end
7	Internal error: (-7) Logic error in TEXT/C40 encodation (impossible branch)
8	The given data can not be encoded using X12 encodation.
9	The "tilde" encoded data is not valid.
10	Data must have at least three characters for X12 encodation
11	Specified data can not be encoded with datamatrix 000 140
12	Can not create image
13	Invalid color specification
14	Internal error: (-14) Index for 140 bit placement matrix out of bounds
15	This PHP installation does not support the chosen image encoding format
16	Internal error: (-16) Cannot instantiate Reed-Solomon
20	The specification for shape of matrix is out of bounds (0,29)
21	Cannot open the data file specifying bit placement for Datamatrix 200
22	Datafile for bit placement is corrupt, crc checks fails.
23	Internal error: (-23) Output matrice is not big enough for mapping matrice
24	Internal error: (-24) Bit sequence to be placed is too short for the chosen output matrice
25	Internal error: (-25) Shape index out of bounds for bit placement
26	Cannot open the data file specifying bit placement for Datamatrix 140
30	The symbol size specified for ECC140 type Data-matrix is not valid
31	Data is to long to fit into any available matrice size for datamatrix 140
32	Internal error: (-32) Cannot instantiate MasterRandom

Error code	Error message
33	Internal error: (-33) Failed to randomize 140 bit stream
34	Cannot open file %s for writing
35	Cannot write to file %s
99	EDIFACT encodation not implemented
100	<p><b>JpGraph Error: HTTP headers have already been sent.</b></p> <p>Caused by output from file %s at line %d.</p> <p>Explanation: HTTP headers have already been sent back to the browser indicating the data as text before the library got a chance to send it's image HTTP header to this browser.</p> <p>This makes it impossible for the library to send back image data to the browser (since that would be interpreted as text by the browser and show up as junk text). Most likely you have some text in your script before the call to Graph::Stroke(). If this texts gets sent back to the browser the browser will assume that all data is plain text. Look for any text, even spaces and newlines, that might have been sent back to the browser. For example it is a common mistake to leave a blank line before the opening</p>

---

# Appendix I. Compiling PHP

Usually the included PHP version is adequate but if problems persists it is a good idea to be able to compile PHP yourself. This way you will also be able to more quickly upgrade to newer version of PHP which might have fix for a particular nasty bug that may have crept in.

In the following sections we give examples of Unix shell scripts that will show typical compile configuration for a downloaded PHP distribution. These compile configuration scripts will make both the GD and FreeType libraries included in the executable.

In order to compile your downloaded PHP distribution first copy and save these scripts to a local file and make that file runnable. Then run one of the selected configurations below and do a normal "make".

## Tip

When running make you can speed up the compilation by telling make to use a number of parallel compile processes. Since most modern system have at least two cores a typical invocation of make would be to make use of three parallel compile time processes. This is done by using the -j argument. For example as

```
make -j3
```

It is possible to compile PHP into (at least) three variants

1. as a command line tool
2. as a Apache extension module (this is probably the most common variant)
3. as a CGI module to be used by a HTTP server (this is slower than running PHP as a module since it needs to be read from disk and the process created every time a PHP script needs to be executed.)

There is one crucial difference of importance when using PHP to generate images. Both the CGI module and the client variant are both standalone executables so what is the difference? The crucial difference is that the CGI module will by default output a MIME header before it outputs data while the client version will not.

The following sections have one compile script for each of the three major versions.

## Caution

You should make sure that the proposed directory paths in the scripts match your particular server setup as this can vary from system to system.

## I.1. Compiling PHP4

### I.1.1. Client version

```
#!/bin/sh
./configure --prefix=/usr/share --datadir=/usr/share/php \
--libdir=/usr/share/php --includedir=/usr/include \
--bindir=/usr/bin \
--with-config-file-path=/etc/php4/cli \
--with-config-file-scan-dir=/etc/php4/cli \
```

```
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd --enable-gd-imgstrttf --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib --with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-memory-limit \
--enable-bcmath -enable-calendar \
--enable-ctype --with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--with-bz2 \
--with-iconv
```

## I.1.2. Apache module

```
#! /bin/sh
./configure --prefix=/usr/share --datadir=/usr/share/php --with-apxs2=/usr/sbin/ap
--libdir=/usr/share --includedir=/usr/include \
--bindir=/usr/bin \
--with-config-file-path=/etc/php4/apache2 \
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd --enable-gd-imgstrttf --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib --with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-memory-limit \
--bindir=/usr/bin \
--enable-bcmath \
--enable-calendar \
--enable-ctype \
--with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--with-bz2 \
--with-iconv
```

## I.1.3. CGI extension

```
#! /bin/sh
./configure --prefix=/usr/share --datadir=/usr/share/php \
--libdir=/usr/share --includedir=/usr/include \
--bindir=/usr/bin \
--with-config-file-path=/etc/php4/apache2 \
--with-config-file-scan=/etc/php4/apache2 \
--enable-mbstring --enable-mbregex \
```

```
--with-mysql \
--with-gd --enable-gd-imgstrttf --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib --with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-memory-limit \
--bindir=/usr/bin \
--enable-bcmath \
--enable-calendar \
--enable-ctype \
--with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--with-bz2 \
--with-iconv
```

## I.2. Compiling PHP5

### I.2.1. Client version

```
#!/bin/sh
mkphp5-cli
Build a command line version of PHP5
./configure \
--prefix=/usr/share/php5 \
--datadir=/usr/share/php5 \
--libdir=/usr/share/php5 \
--includedir=/usr/include/php5 \
--enable-force-cgi-redirect \
--bindir=/usr/bin \
--with-config-file-path=/etc/php5/cli \
--with-config-file-scan-dir=/etc/php5/cli \
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd --enable-gd-imgstrttf --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib --with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-memory-limit \
--enable-safe-mode \
--enable-bcmath -enable-calendar \
--enable-ctype \
--with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--enable-tokenizer \
--with-bz2 \
--with-iconv \
```

```
--with-pear=/usr/share/php5
```

## I.2.2. Apache module

```
#!/bin/sh
mkphp5-sapi
Build a SAPI (Apache module) version of PHP5
./configure --prefix=/usr/share \
--datadir=/usr/share/php --with-apxs2=/usr/sbin/apxs2 \
--libdir=/usr/share --includedir=/usr/include \
--bindir=/usr/bin \
--with-config-file-path=/etc/php5/apache2 \
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd --enable-gd-imgstrttf --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib --with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-memory-limit --enable-safe-mode \
--bindir=/usr/bin \
--enable-bcmath -enable-calendar \
--enable-ctype --with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--with-bz2 \
--with-iconv

!#
```

## I.2.3. CGI extension

```
#!/bin/sh
mkphp5-cgi
Build a CGI version of PHP5
./configure --prefix=/usr/share \
--datadir=/usr/share/php \
--libdir=/usr/share --includedir=/usr/include \
--enable-force-cgi-redirect \
--bindir=/usr/bin \
--with-config-file-path=/etc/php5/apache2 \
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-xpm-dir=/usr/X11R6 \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-safe-mode \
--bindir=/usr/bin \
--enable-bcmath -enable-calendar \
```

```
--enable-ctype \
--enable-magic-quotes \
--enable-inline-optimization \
--enable-tokenizer \
--with-bz2 \
--with-iconv
```

---

# Appendix J. Setting up PHP5 in parallel with PHP4 in SuSE 10.1

Even though PHP4 is officially deprecated and is no longer actively maintained a large number of existing installations are still (and will be) using PHP4. For this reason it can be important to be able to test scripts running both PHP4 and PHP5. This section shows how to do this on Linux SuSE 10.1 installation. Other Linux dialects can use similar but not identical setups.

SuSE 10.1 ships with Apache2 and PHP5 as standard. In order to install PHP4 in parallel some extra work is therefore required. This chapter explains how to setup both PHP4 and PHP5 on the same server by configuring Apache2 using virtual hosts.

We will show how to maintain a simultaneous installation of both PHP4 and PHP5 at the same time without the need to run a switching script to select which PHP version to activate. In this setup we have opted to configure Apache with two virtual hosts based on IP-address, one host running PHP4 as a SAPI module and the other virtual host running PHP5 as a CGI module.

## J.1. Configuration files and directories for Apache2 in SuSE 10.1

Before we start we give a short overview of where important configuration files and directories for Apache2 are located in SuSE

### Note

The configuration files setup by SuSE is slightly different from the standard one-single "/etc/httpd.conf" used by other systems. The way SuSE does it is to create a hierarchy of setup files under "/etc/apache2/". While this might look complicated at first sight it has several advantages specially when You consider that some script have to modify and add configurations to Apache2. Doing automatic (and safe) edits in a large config file that can also be manually edited is almost impossible to guarantee.

Dir: /etc/apache2/

General configuration directory for Apache2, this is where the "httpd.conf" lives.

Dir: /etc/apache2/conf.d/

Module configuration files for loaded modules, for example php4.conf. All the configuration files in this directory will be automatically read by the main httpd.conf by means of an "include conf.d/\*.conf" command so the exact name doesn't really matter as long as the file ends in "\*.conf".

Dir: /etc/apache2/vhosts.d/

Virtual host configuration files. All files in this directory will be automatically read by the main httpd.conf the exact name doesn't really matter as long as the file ends in "\*.conf". Note: When yast2 is used to edit virtual hosts it will add its "yast2\_\*\_.conf" in this directory. Unfortunately the virtual host configuration in yast2 is not without problem (bugs) for IP based virtual hosts so we prefer to create the configuration files manually. This will be shown later on in this article.

File: /etc/sysconfig/apache2

This is the main Apache2 configuration file. This file is the one that is really used to configure apache when it is started. This is also the file that the "Yast2" HTTPD-module edits.

From our point of view the most important thing is that this is the place where we tell Apache2 what external modules to load.

In the SuSE configuration this is done by listing all the modules in the string variable APACHE\_MODULES. In the SuSE configuration there are no static "AddModule" directives in any of the configuration files for Apache. Instead this is dynamically generated each time apache is started (for example by /etc/init.d/apache2 start)

The generation of the actual module file names is quite clever in that the script looks at the core module name in the APACHE\_MODULE variable and automatically determines the name of the file name of the load modules. This means that for PHP we only have to give the name "php4" or "php5" as the name of the module.

The script will then discover that the name of the file load module is in fact "libphp4.so" or "libphp5.so" automatically. The dynamically created list of load modules will be written to "/etc/apache2/sysconfig/loadmodule.conf" just before the startup script activates apache2 daemon which will then read the modules from this file which is included from the main "httpd.conf" file.

## J.2. Making sure you have the correct Apache2 setup

PHP is only guaranteed to work with the Apache2 "Prefork MPM" (Multi-Processing-Module) and you need to have apache2-prefork installed. This also means that the APACHE\_MPM in /etc/sysconfig/apache2 must NOT be set to "worker". You can read more about the reasons for this issues in the Apache Documentation : Thread Safety [[http://httpd.apache.org/docs-2.0/developer/thread\\_safety.html](http://httpd.apache.org/docs-2.0/developer/thread_safety.html)] For general information about MPMs please see Apache Documentation : MPM [<http://httpd.apache.org/docs-2.0/mpm.html>]

If you use Yast2 to install Apache2 and the prefork module then all this will be automatically setup. Before continuing please make sure that You have successfully installed Apache2 on your server. For example by directing your favorite browser to "<http://localhost/>"

### Note

There is no need to install the default SuSE PHP5 module since we must replace that anyway with our own CGI version of PHP5.

## J.3. Approaches to running multiple PHP versions

There are two fundamental ways of running multiple versions of PHP on the same server.

1. Running multiple instances of the HTTPD demon where each instance listens on separate addresses and/or ports.

**Advantage:** This is the only way to run multiple versions of PHP as (SAPI) modules in Apache2. In addition this has some better security since potential crashes will be isolated and not effect the other HTTPD demons.

**Drawback:** Running multiple HTTPD instances will need more system resources in terms of memory and file handlers.

2. Running one instance of the HTTPD demon which is configured to serve multiple virtual hosts. This is the approach we have chosen.

**Advantage:** Minimum system overhead and relatively easy to setup.

**Drawback:** Only one PHP version can be run as a (SAPI) Apache module the other PHP versions must be configured/installed as CGI modules. This has a slight performance impact and might not be suitable for heavily loaded production sites. (Note: that could be overcome with the use of fast-cgi which works by pre-loading an instance of PHP in memory which will then be used by the Apache process. See Apache2 documentation regarding fast-cgi for more details).

## Note

There are actually two versions of virtual hosts with apache. By name pr by IP-address. In this example we have chosen to match the virtual hosts by IP address since for a development server we want to be able to use plain IP addresses and not have the added complexity of setting up a full DNS server. For more details about other differences please see the excellent Apache2 documentation.

## J.4. Outline of the remainder of the chapter

The approach we will use is to setup PHP4 as a (SAPI) Apache module on the default server address and setup PHP5 as a CGI module on a virtual host.

Part I - Installing PHP4 as a SAPI module in Apache.

Part 1 Installing PHP4 as a SAPI module in Apache. This sections details how to configure and compile PHP4 as a SAPI module and then do the necessary Apache configuration modifications to enable this new module. By the end of this section we will have the ability to run PHP4 scripts on our server.

Part II - Creating a virtual host

By assigning an alias IP-address on the server we can configure Apache with a virtual server based on this address. This new virtual server will have its own "cgi-bin/" as well as "htdocs/" directories. This part shows how to enable this by adding suitable configurations in Apache. By the end of this section our server will accept HTTP calls on a secondary IP-Address and use the specified document root for this new IP-address.

Part III - Installing PHP5 as a CGI module on the virtual host.

This final part shows how to configure and compile PHP5 as a CGI module that we then make available for the newly created virtual host in part 2. By the end of this module we will have PHP4 running on the default server address and PHP5 running on the secondary virtual host.

## J.5. Part I - Installing PHP4

### J.5.1. Step one; Compiling PHP4 as a module for Apache2

First download the latest PHP4 tar-ball from php.net or the closest mirror and unpack it in a temporary directory.

Since we will compile PHP4 ourself we need first to make sure a number of libraries and the corresponding header files are installed in the system in order to be able to compile PHP4. This is done by installing a number of "\*-devel.rpm" on your server. Depending your wanted configuration different development libraries must be made available.

At the very minimum you will need the "apache2-devel.rpm" which provides the "/sbin/apxs2" (Apache eXtenSion 2) command used to build modules with Apache2. Other modules you might need are

- jpeg-devel.rpm
- png-devel.rpm
- mm-devel.rpm
- xml2-devel.rpm
- mysql-devel.rpm
- ...

Before you compile PHP4 you need to configure it by running the "./configure" command with the options you want to be included in PHP4.

We use a small shell script called "mkphp4-sapi" to avoid having to re-type all the options each time we compile a new version of PHP. The options we use for a typical development server are (you might want to use other options)

```
#!/bin/sh
./configure --prefix=/usr/share \
--datadir=/usr/share/php4 \
--with-apxs2=/usr/sbin/apxs2 \
--libdir=/usr/share \
--includedir=/usr/include \
--bindir=/usr/bin \
--with-config-file-path=/etc/php4/apache2 \
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd --enable-gd-imgstrttf --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib \
--with-jpeg-dir=/usr/lib --with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib --with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-memory-limit --enable-safe-mode \
--bindir=/usr/bin \
--enable-bcmath -enable-calendar \
```

```
--enable-ctype --with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--with-bz2 \
--with-iconv
```

However there are one thing You should take notice of. We have specified the config file path (where the php.ini resides) to "/etc/php4/apache2/" as You can probably guess from this naming convention it will make it possible to have different ini files for both PHP4 and PHP5. In fact we have four different ini files according to

1. "/etc/php4/apache2/php.ini" Used by the apache SAPI module version of PHP4
2. "/etc/php4/cli/php.ini" Used by the standalone client version of PHP4 (/usr/bin/php4)
3. "/etc/php5/apache2/php.ini" Used by the apache CGI version of PHP5
4. "/etc/php5/cli/php.ini" Used by the standalone client version of PHP5 (/usr/bin/php5)

When you run this you might get some errors saying that the configuration file cannot find some library. This is a sign that you might have the library installed but not yet have the "\*-devel" RPM version added to your system which is needed since this is where all the usual header files needed for compilation would be.

So for example if you get an error like "Cannot find PNG libraries. Please check your that the corresponding "png-devel" library is installed and if not go back to Yast2 and install the needed "\*-devel.rpm" versions of the libraries.

When You have been able to successfully run the ./configuration command it is time to compile. Type "make" as usual but do not type "make install", now wait until the compilation finishes.

### Note

If you are on a Pentium4 HT or on a real dual CPU machine you can speed up the compilation by instead giving the "make -j3" command which will start up 3 concurrent compilation processes.

Again; Do not run "make install" since this will try to modify the configuration files in a way that isn't SuSE friendly.

The resulting PHP4 that you have built can be found in ".libs/libphp4.so". Now we only want to copy this file to the location of the other Apache2 modules.

### Note

Again, PHP is only guaranteed to work with the non-threaded version of Apache2, which means that you should have installed the "apache2-prefork" MPM and NOT the "apache2-worker" MPM.

If you have correctly installed the prefork MPM several existing modules should now be installed in "/usr/lib/apache2-prefork/".

So the only thing that now remains is to copy ".libs/libphp4.so" to "/usr/apache2-prefork/" in order for Apache to find PHP4 as a module.

## J.5.2. Step two; Enable the PHP4 module in the Apache2 configuration

There are three steps to needed to enable PHP4 in Apache.

1. Add php4 to the APACHE\_MODULE string in "/etc/sysconfig/apache2" in order so that the startup script in SuSE will add the appropriate LoadModule statement so that Apache will load PHP4 as a module. In our case our module string will look like

```
APACHE_MODULES="access actions alias auth auth_dbm autoindex cgi \
dir env expires include log_config mime negotiation setenvif ssl \
suexec userdir dav dav_svn php4 "
```

2. Telling Apache to run files ending in \*.php through the PHP4 module. This is done by specifying the MIME type which the PHP4 module registered itself with. In addition we also tell Apache to search for the appropriate PHP index files in case a directory name is given as the URL. We do this by creating a file "php4.conf" with the following content

```
<IfModule sapi_apache2.c>
 AddType application/x-httdp-php .php3
 AddType application/x-httdp-php .php4
 AddType application/x-httdp-php .php
 AddType application/x-httdp-php-source .php3s
 AddType application/x-httdp-php-source .php4s
 AddType application/x-httdp-php-source .phps
 DirectoryIndex index.php3
 DirectoryIndex index.php4
 DirectoryIndex index.php
</IfModule>
```

and place it in the "/etc/apache2/conf.d/" directory. This will guarantee that it will be read upon startup. The "IfModule" statement in the beginning is just to avoid the statements to be executed in case the PHP4 module is not loaded (we test this by checking if the "sapi\_apache2.c" has been activated in Apache).

3. The final step now is to restart Apache by doing (as root)

```
$> /etc/init.d/apache2 restart
```

In order to verify that PHP has been enabled run a standard PHP script; for example by copying the following script to "/srv/www/htdocs/"

```
<?php
phpinfo();
?>
```

and name it as "phpinfo.php". If you now go to your favorite browser and run this script as "http://localhost/phpinfo.php" you should get the standard PHP4 information presented as a quite big table.

## J.6. Part II - Creating a virtual host

### J.6.1. Step 1; Adding an alias IP-address to Your server

In this example we will assume that the server is called "gamma" and have the primary address "192.168.0.50". The virtual host will be called "gamma2" and will be located at address "192.168.0.51". The easiest way to add another address alias is to use yast2 and the network configuration module and simple add a new alias.

## J.6.2. Step 2; Creating different document and cgi roots

In preparation of the new virtual host we want it to have a separate document and cgi (where we will store the PHP5 binary) roots compared with the standard server. For this purpose we add two new directories "/srv/www/gamm2-htdocs/" and "/srv/www/gamma2-cgi-bin/" on the server.

## J.6.3. Step 3; Configure Apache with a virtual host

For this we add a new small config file named "gamma2\_vhost.conf" (the exact name is not important as long as it ends in \*.conf) in the "/etc/apache2/vhosts.d/" directory. The script we add is

```
Setup gamma2 on secondary IP-address
<VirtualHost 192.168.0.51>

 DocumentRoot /srv/www/gamma2-htdocs/
 ServerName gamma2
 ServerAdmin root@localhost

 # We use a separate CGI directory
 ScriptAlias /cgi-bin/ /srv/www/gamma2-cgi-bin/

 # For good measure we also add recognition of PHP5 index
 DirectoryIndex index.php5

 # This is the two critical statement for this virtual
 # host we activate PHP5 as a CGI module
 Action php5-cgi /cgi-bin/php
 AddHandler php5-cgi .php5 .php

 <Directory /srv/www/gamma2-cgi-bin/>
 AllowOverride None
 Options +ExecCGI -Includes
 Order allow,deny
 Allow from all
 </Directory>

 <Directory "/srv/www/gamma2-htdocs/">
 Options None
 AllowOverride None
 Order allow,deny
 Allow from all
 DirectoryIndex index.html index.php
 </Directory>

 UserDir public_html

</VirtualHost>
```

We do not go into any more detail of this configuration since it should be fairly easy to understand. For details we refer to the Apache documentation.

What we have accomplished with this file is that when we call the server on the second address any php file will be recognized by apache as a file to be handled by the "php5-cgi" action. This in turn means that

whenever Apache encounters a \*.php5 (or \*.php) file it will run the program "/cgi-bin/php". This path in turn will be expanded to "/srv/www/gamma2-cgi-bin/php".

In the next section we will show how to compile PHP5 and put the executable CGI version in this directory.

## J.7. Part III - Installing PHP5

We are now ready for the last step which means compiling PHP5 as a CGI module for Apache. This follows the same principle as the compilation for PHP4 as described above. Again, we use a small configuration script "mkphp5-cgi" which is shown below.

```
#!/bin/sh
./configure --prefix=/usr/share \
--datadir=/usr/share/php \
--libdir=/usr/share --includedir=/usr/include \
--enable-force-cgi-redirect \
--bindir=/usr/bin \
--with-config-file-path=/etc/php5/apache2 \
--enable-mbstring --enable-mbregex \
--with-mysql \
--with-gd --enable-gd-imgstrttf --enable-gd-native-ttf \
--with-zlib-dir=/usr/lib \
--with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib \
--with-xpm-dir=/usr/X11R6 \
--with-tiff-dir=/usr/lib --with-ttf-dir=/usr/lib \
--with-freetype-dir=/usr/lib \
--enable-ftp \
--enable-memory-limit --enable-safe-mode \
--bindir=/usr/bin \
--enable-bcmath --enable-calendar \
--enable-ctype --with-ftp \
--enable-magic-quotes \
--enable-inline-optimization \
--with-bz2 \
--with-iconv
```

Notice that as we said before we have a different configuration path for PHP5 compared with PHP4 as shown above. Also note that in order to build the CGI module we do not configure the "apxs2" option. After successful configuration type "**make**" but do not type "**make install**" in order to compile PHP5.

After the compilation have finished copy "sapi/cgi/php" to "/srv/www/gamma2-cgi-bin/php" since this is the place where our virtual host expects to find the PHP5 CGI module.

## J.8. Part IV - Verifying the setup

The only small thing remaining, in case You haven't done so already, is to create suitable "php.ini" scripts that are copied to the previous specified PHP4 and PHP5 config file directories, "/etc/php4/apache2/" and "/etc/php5/apache2/".

### Note

It is recommended to enable all warnings and errors in each php.ini file unless You have very, very good reasons not to do this. Use the default php.ini files in the PHP distribution as the initial template. For use with JpGraph it is recommended to do the following modifications:

1. Increase maximum allowed memory to 32MB
2. Increase maximum allowed script running time to 30s
3. Set full error reporting

Assuming the IP-addresses shown in the configuration above we are now ready to test out setup. In order to do this make sure that each document root have the "phpinfo.php" test script (see above).

We can now test the different setups by using the URLs

1. `http://192.168.0.50/phpinfo.php`

This URL would send back configuration showing that the server is running PHP4. Verify that the config path used is set to "/etc/php4/apache2/".

2. `http://192.168.0.51/phpinfo.php`

This URL would send back configuration showing that the server is running PHP5. Verify that the config path used is set to "/etc/php5/apache2/".

## J.8.1. Troubleshooting

1. If your browser asks you to download content with mime-type "mime/x-httd-application" when you try to visit the PHP script it means that Apache does not yet run PHP as a module (now module have accepted to handle the x-httd-application mime type). Make sure you have included the "php4" in the APACHE\_MODULES string as described above and that you added the "php4.conf" file in the "/etc/apache2/conf.d/" directory.
2. We had some issues with FireFox insisting on downloading "phpinfo.php" as a file even when other browser showed the page properly, using "**etherreal**" we could confirm that Firefox was using a previous cached version before we had enabled PHP in the apache configuration. To solve this we had to clear the Firefox cache.

---

# **Appendix K. Why it is not possible to add a SVG backend to JpGraph**

## **K.1. Background**

We have received many suggestions to add SVG as output from the library to achieve better quality in off-screen reproduction of graphs. Adding SVG output would significantly enhance the print quality as well as adding the possibility of seamless zooming in graphs. For this reason we have done a brief pre-study on the feasibility of such a backend. This short note will describe our findings.

## **K.2. Summary of findings**

To our surprise we have to conclude that with the current SVG standard 1.1 as well as with the upcoming 1.2 (based on SVG-T) it will not be possible to implement a full backend in SVG.

The primary hinder is the lack of adequate text manipulation in the current SVG standard. In addition the implementation of the text supporting feature in current SVG viewers range from poor and erroneous to non-existing.

The only way to solve this would be to drastically reduce some functionality of the library in regards to text handling and require the user of the library to supply text sizes to be used in many places of the library where that is needed.

## **K.3. Detailing the issue**

### **K.3.1. The core problem**

It all boils down to one critical issue:

With the current SVG 1.1 (and draft 1.2) standard there is no way to statically find out the bounding box of an arbitrary text string for later usage in the SVG script.

This very surprising omission in the SVG standard makes it in principal impossible to even do such a simple thing as drawing a frame around a text programatically since there is no easy way to find out the size, in the given coordinate system, of the string.

Since the actual bounding box is dependent on both font, style, size, etc as well as the actual SVG viewer text-layout engine implementation this calculation cannot be done outside the viewer. It must be part of the SVG standard elements.

#### **Note**

Now, anyone who are familiar with SVG would jump in here and point out that this is not entirely correct. For the specific case of a frame around a text it would be possible to use a filter function as specified by the standard but that is a special case that just could be used to draw an effect that looks like a frame around a text (using the objectBoundingBox property). It is still not possible to find out the bounding box.

The second approach would be to add some DOM Javascript code in the SVG script which upon execution of the script could in theory find out the bounding box and adjust suitable attributes in the script.

### K.3.2. Why is this a problem ?

There are many places in the library where it is absolutely essential to find out the bounding box of a text string to adjust the position of other object in the graph. For example margins for titles, column width in gantt charts and legends and so on. Without this functionality it will be impossible to add SVG output without significantly reducing the functionality and in essence create a new version of the library suitable for this reduced functionality that is brought upon us by the use of SVG.

### K.3.3. Possible workarounds

Looking at this from a more positive view instead of explaining why it cannot be done there are in principal only two workarounds (neither which is a 100% solution)

1. Using a single fixed font. Restricting the library to one specific fixed font would make it possible to calculate the bounding box for the string. Due to differences in the existing viewers it would be necessary to have some safety margins built in when doing this calculation. However this would significantly impact the visual appearance of the graphs.
2. Using heuristics By establishing some "good enough" heuristics for a plain font we can try to find a guesstimate of the size of the string. Unfortunately it is a big difference in length between "iiiii" and "wwwww" even though they have the same number of characters. So without fully implementing the same algorithm as some SVG viewer text-layout engine uses this method cannot guarantee that the text will always fit without making the box fit the worst case. In addition this method will have some difficulty in handling rotated text strings.

### K.3.4. What would be required ?

What would be required in the standard to solve this is a new basis element which could be used to record the bounding box of a particular text string for later reference. To just give some idea on what is needed some "pseudo-SVG" that we would need is something along the lines of:

```
<def>
 <boundingbox id="bb1"
 text="This is a text" style=" />
</def>
<rect x="50+#bb1.x1-10" y="50+#bb1.y1-10"
 width="#bb1.width+20"
 height="#bb1.height+20" />
<text x="50" y="50" >
 <tref xlink:href="#bb1" />
</text>
```

The basic idea is that in the def-section all text strings to later be used in the script is defined together with the font (and any other formatting applicable). These text strings are defined in the new SVG element "boundingbox" which will calculate the bounding box of the given text. These text string is later referenced in the actual text with a standard tref element. The bounding box attributes can then be used in the positioning of the text with a "#" reference based on the id of the new introduced element "boundingbox". The above script would then draw a text string positioned at (50,50) with a frame around it with a 10 units margin all around.

### K.3.5. DOM scripting and GetBBox()

Since we make no claim to be experts in all aspects of the SVG standard (which is fairly big) it might be possible that there is some way to still solve this that has eluded us so we would be very interested in

getting a second opinion of these findings. We are aware of the SVG method GetBBox() but this would not work in the library very well. The reason is that this is not a static function but requires the context of a DOM script. This would require a substantially rewrite of the library since there are graphs where every single coordinate would have to be back-patched in the end (possible in multiple passes - since the calculation of one bounding box would be needed to adjust another element).

This means that the script would no longer be static but would require the library to generate "self-modifying" DOM script at the end. The logic of the library assumes that the bounding box of text can be found out at the place of creation and then this bounding box can be used to adjust subsequent coordinates.

So to summarize this we do not feel that the potential back patching of every single element in the SVG image at the end in a DOM script is a solution.

### K.3.6. A final comment

Since we still find it very hard to believe this giant oversight in the standard we would be happy to receive comments on these conclusions.

---

# Appendix L. The JpGraph configuration file

```
<?php
//=====
// File: JPG-CONFIG.INC
// Description: Configuration file for JpGraph library
// Created: 2004-03-27
// Ver: $Id: jpg-config.inc.php 1839 2009-09-25 12:36:15Z ljp $
//
// Copyright (c) Aditus Consulting. All rights reserved.
//=====

// Directories for cache and font directory.
//
// CACHE_DIR:
// The full absolute name of the directory to be used to store the
// cached image files. This directory will not be used if the USE_CACHE
// define (further down) is false. If you enable the cache please note that
// this directory MUST be readable and writable for the process running PHP.
// Must end with '/'
//
// TTF_DIR:
// Directory where TTF fonts can be found. Must end with '/'
//
// The default values used if these defines are left commented out are:
//
// UNIX:
// CACHE_DIR /tmp/jpgraph_cache/
// TTF_DIR /usr/share/fonts/truetype/
// MBTTF_DIR /usr/share/fonts/truetype/
//
// WINDOWS:
// CACHE_DIR $SERVER_TEMP/jpgraph_cache/
// TTF_DIR $SERVER_SYSTEMROOT/fonts/
// MBTTF_DIR $SERVER_SYSTEMROOT/fonts/
//
// -----
// define('CACHE_DIR','/tmp/jpgraph_cache/');
// define('TTF_DIR','/usr/share/fonts/truetype/');
// define('MBTTF_DIR','/usr/share/fonts/truetype');

// -----
// Cache directory specification for use with CSIM graphs that are
// using the cache.
// The directory must be the filesystem name as seen by PHP
// and the 'http' version must be the same directory but as
// seen by the HTTP server relative to the 'htdocs' ddirectory.
// If a relative path is specified it is taken to be relative from where
```

```
// the image script is executed.
// Note: The default setting is to create a subdirectory in the
// directory from where the image script is executed and store all files
// there. As usual this directory must be writeable by the PHP process.
define('CSIMCACHE_DIR','csimcache/');
define('CSIMCACHE_HTTP_DIR','csimcache/');

// Various JpGraph Settings. Adjust accordingly to your
// preferences. Note that cache functionality is turned off by
// default (Enable by setting USE_CACHE to true)

// Default locale for error messages.
// This defaults to English = 'en'
define('DEFAULT_ERR_LOCALE','en');

// Default graphic format set to 'auto' which will automatically
// choose the best available format in the order png,gif,jpeg
// (The supported format depends on what your PHP installation supports)
define('DEFAULT_GFORMAT','auto');

// Should the cache be used at all? By setting this to false no
// files will be generated in the cache directory.
// The difference from READ_CACHE being that setting READ_CACHE to
// false will still create the image in the cache directory
// just not use it. By setting USE_CACHE=false no files will even
// be generated in the cache directory.
define('USE_CACHE',true);

// Should we try to find an image in the cache before generating it?
// Set this define to false to bypass the reading of the cache and always
// regenerate the image. Note that even if reading the cache is
// disabled the cached will still be updated with the newly generated
// image. Set also 'USE_CACHE' below.
define('READ_CACHE',true);

// Determine if the error handler should be image based or purely
// text based. Image based makes it easier since the script will
// always return an image even in case of errors.
define('USE_IMAGE_ERROR_HANDLER',true);

// Should the library examine the global php_errmsg string and convert
// any error in it to a graphical representation. This is handy for the
// occasions when, for example, header files cannot be found and this results
// in the graph not being created and just a 'red-cross' image would be seen.
// This should be turned off for a production site.
define('CATCH_PHPERRMSG',true);

// Determine if the library should also setup the default PHP
// error handler to generate a graphic error message. This is useful
// during development to be able to see the error message as an image
// instead as a 'red-cross' in a page where an image is expected.
define('INSTALL_PHP_ERR_HANDLER',false);
```

```
// Should usage of deprecated functions and parameters give a fatal error?
// (Useful to check if code is future proof.)
define('ERR_DEPRECATED',true);

// The builtin GD function imagettfbbox() fuction which calculates the bounding bo
// text using TTF fonts is buggy. By setting this define to true the library
// uses its own compensation for this bug. However this will give a
// slightly different visual appearance than not using this compensation.
// Enabling this compensation will in general give text a bit more space to more
// truly reflect the actual bounding box which is a bit larger than what the
// GD function thinks.
define('USE_LIBRARY_IMAGETTFBOX',true);

// The following constants should rarely have to be changed !

// What group should the cached file belong to
// (Set to '' will give the default group for the 'PHP-user')
// Please note that the Apache user must be a member of the
// specified group since otherwise it is impossible for Apache
// to set the specified group.
define('CACHE_FILE_GROUP','www');

// What permissions should the cached file have
// (Set to '' will give the default permissions for the 'PHP-user')
define('CACHE_FILE_MOD',0664);

?>
```