

# The JFreeChart Class Library

Version 1.0.9

## Developer Guide

Written by David Gilbert

January 7, 2008

© 2000-2008, Object Refinery Limited. All rights reserved.

### **IMPORTANT NOTICE:**

We work hard to make this document as accurate and informative as we can, but  
cannot guarantee that it is error-free.

# Contents

<b>1</b>	<b>Introduction</b>	<b>16</b>
1.1	What is JFreeChart?	16
1.2	This Document	18
1.3	Acknowledgements	18
1.4	Comments and Suggestions	18
<b>2</b>	<b>Sample Charts</b>	<b>19</b>
2.1	Introduction	19
2.2	Pie Charts	19
2.3	Bar Charts	21
2.4	Line Chart	23
2.5	XY Plots	24
2.6	Time Series Charts	25
2.7	Histograms	26
2.8	Area Charts	27
2.9	Difference Chart	27
2.10	Step Chart	29
2.11	Gantt Chart	30
2.12	Multiple Axis Charts	31
2.13	Combined and Overlaid Charts	32
2.14	Future Development	33
<b>3</b>	<b>Downloading and Installing JFreeChart</b>	<b>34</b>
3.1	Introduction	34
3.2	Download	34
3.3	Unpacking the Files	34
3.4	Running the Demonstration Applications	35
3.5	Configuring JFreeChart for use in IDEs	36
3.6	Compiling the Source	36
3.7	Generating the Javadoc Documentation	36
<b>4</b>	<b>Using JFreeChart</b>	<b>37</b>
4.1	Overview	37
4.2	Creating Your First Chart	37
<b>5</b>	<b>Pie Charts</b>	<b>40</b>
5.1	Introduction	40
5.2	Creating a Simple Pie Chart	40
5.3	Section Colours	40
5.4	Section Outlines	41
5.5	Null, Zero and Negative Values	41
5.6	Section and Legend Labels	42
5.7	Exploded Sections	42

5.8	3D Pie Charts . . . . .	43
5.9	Multiple Pie Charts . . . . .	43
<b>6</b>	<b>Bar Charts</b>	<b>45</b>
6.1	Introduction . . . . .	45
6.2	A Bar Chart . . . . .	45
6.3	The ChartFactory Class . . . . .	48
6.4	Simple Chart Customisation . . . . .	48
6.5	Customising the Renderer . . . . .	49
<b>7</b>	<b>Line Charts</b>	<b>51</b>
7.1	Introduction . . . . .	51
7.2	A Line Chart Based On A Category Dataset . . . . .	51
7.3	A Line Chart Based On An XYDataset . . . . .	56
<b>8</b>	<b>Time Series Charts</b>	<b>61</b>
8.1	Introduction . . . . .	61
8.2	Time Series Charts . . . . .	61
<b>9</b>	<b>Customising Charts</b>	<b>67</b>
9.1	Introduction . . . . .	67
9.2	Chart Attributes . . . . .	67
9.3	Plot Attributes . . . . .	69
9.4	Axis Attributes . . . . .	70
<b>10</b>	<b>Dynamic Charts</b>	<b>72</b>
10.1	Overview . . . . .	72
10.2	Background . . . . .	72
10.3	The Demo Application . . . . .	73
<b>11</b>	<b>Tooltips</b>	<b>77</b>
11.1	Overview . . . . .	77
11.2	Generating Tool Tips . . . . .	77
11.3	Collecting Tool Tips . . . . .	78
11.4	Displaying Tool Tips . . . . .	78
11.5	Disabling Tool Tips . . . . .	78
11.6	Customising Tool Tips . . . . .	78
<b>12</b>	<b>Item Labels</b>	<b>79</b>
12.1	Introduction . . . . .	79
12.2	Displaying Item Labels . . . . .	80
12.3	Item Label Appearance . . . . .	81
12.4	Item Label Positioning . . . . .	82
12.5	Customising the Item Label Text . . . . .	83
12.6	Example 1 - Values Above a Threshold . . . . .	84
12.7	Example 2 - Displaying Percentages . . . . .	87
<b>13</b>	<b>Multiple Axes and Datasets</b>	<b>91</b>
13.1	Introduction . . . . .	91
13.2	An Example . . . . .	91
13.3	Hints and Tips . . . . .	93

<b>14 Combined Charts</b>	<b>94</b>
14.1 Introduction . . . . .	94
14.2 Combined Domain Category Plot . . . . .	94
14.3 Combined Range Category Plot . . . . .	95
14.4 Combined Domain XY Plot . . . . .	96
14.5 Combined Range XY Plot . . . . .	97
<b>15 Datasets and JDBC</b>	<b>99</b>
15.1 Introduction . . . . .	99
15.2 About JDBC . . . . .	99
15.3 Sample Data . . . . .	99
15.4 PostgreSQL . . . . .	100
15.5 The JDBC Driver . . . . .	101
15.6 The Demo Applications . . . . .	102
<b>16 Exporting Charts to Acrobat PDF</b>	<b>103</b>
16.1 Introduction . . . . .	103
16.2 What is Acrobat PDF? . . . . .	103
16.3 iText . . . . .	103
16.4 Graphics2D . . . . .	103
16.5 Getting Started . . . . .	104
16.6 The Application . . . . .	104
16.7 Viewing the PDF File . . . . .	108
16.8 Unicode Characters . . . . .	108
<b>17 Exporting Charts to SVG Format</b>	<b>111</b>
17.1 Introduction . . . . .	111
17.2 Background . . . . .	111
17.3 A Sample Application . . . . .	111
<b>18 Applets</b>	<b>114</b>
18.1 Introduction . . . . .	114
18.2 Issues . . . . .	114
18.3 A Sample Applet . . . . .	115
<b>19 Servlets</b>	<b>118</b>
19.1 Introduction . . . . .	118
19.2 A Simple Servlet . . . . .	118
19.3 Compiling the Servlet . . . . .	120
19.4 Deploying the Servlet . . . . .	121
19.5 Embedding Charts in HTML Pages . . . . .	121
19.6 Supporting Files . . . . .	126
19.7 Deploying Servlets . . . . .	127
<b>20 Miscellaneous</b>	<b>129</b>
20.1 Introduction . . . . .	129
20.2 X11 / Headless Java . . . . .	129
20.3 Java Server Pages . . . . .	129
20.4 Loading Images . . . . .	129
<b>21 Packages</b>	<b>130</b>
21.1 Overview . . . . .	130

<b>22 Package: org.jfree.chart</b>	<b>131</b>
22.1 Overview . . . . .	131
22.2 ChartColor . . . . .	131
22.3 ChartFactory . . . . .	131
22.4 ChartFrame . . . . .	134
22.5 ChartMouseEvent . . . . .	135
22.6 ChartMouseListener . . . . .	136
22.7 ChartPanel . . . . .	136
22.8 ChartRenderingInfo . . . . .	141
22.9 ChartUtilities . . . . .	142
22.10 ClipPath . . . . .	144
22.11 DrawableLegendItem . . . . .	144
22.12 Effect3D . . . . .	144
22.13 HashUtilities . . . . .	145
22.14 JFreeChart . . . . .	145
22.15 LegendItem . . . . .	150
22.16 LegendItemCollection . . . . .	152
22.17 LegendItemSource . . . . .	153
22.18 LegendRenderingOrder . . . . .	154
22.19 PolarChartPanel . . . . .	154
<b>23 Package: org.jfree.chart.annotations</b>	<b>155</b>
23.1 Overview . . . . .	155
23.2 AbstractXYAnnotation . . . . .	155
23.3 CategoryAnnotation . . . . .	157
23.4 CategoryLineAnnotation . . . . .	157
23.5 CategoryPointerAnnotation . . . . .	159
23.6 CategoryTextAnnotation . . . . .	161
23.7 TextAnnotation . . . . .	162
23.8 XYAnnotation . . . . .	164
23.9 XYBoxAnnotation . . . . .	165
23.10 XYDrawableAnnotation . . . . .	166
23.11 XYImageAnnotation . . . . .	167
23.12 XYLineAnnotation . . . . .	168
23.13 XYPointerAnnotation . . . . .	169
23.14 XYPolygonAnnotation . . . . .	171
23.15 XYShapeAnnotation . . . . .	173
23.16 XYTextAnnotation . . . . .	174
<b>24 Package: org.jfree.chart.axis</b>	<b>177</b>
24.1 Overview . . . . .	177
24.2 Axis . . . . .	177
24.3 AxisCollection . . . . .	182
24.4 AxisLocation . . . . .	182
24.5 AxisSpace . . . . .	183
24.6 AxisState . . . . .	183
24.7 CategoryAnchor . . . . .	184
24.8 CategoryAxis . . . . .	184
24.9 CategoryAxis3D . . . . .	189
24.10 CategoryLabelPosition . . . . .	190
24.11 CategoryLabelPositions . . . . .	191
24.12 CategoryLabelWidthType . . . . .	191
24.13 CategoryTick . . . . .	192
24.14 ColorBar . . . . .	192
24.15 CompassFormat . . . . .	192

24.16CyclicNumberAxis . . . . .	193
24.17DateAxis . . . . .	194
24.18DateTickMarkPosition . . . . .	198
24.19DateTick . . . . .	198
24.20DateTickUnit . . . . .	199
24.21ExtendedCategoryAxis . . . . .	200
24.22LogAxis . . . . .	200
24.23LogarithmicAxis . . . . .	203
24.24MarkerAxisBand . . . . .	204
24.25ModuloAxis . . . . .	205
24.26MonthDateFormat . . . . .	206
24.27NumberAxis . . . . .	207
24.28NumberAxis3D . . . . .	211
24.29NumberTick . . . . .	212
24.30NumberTickUnit . . . . .	213
24.31PeriodAxis . . . . .	214
24.32PeriodAxisLabelInfo . . . . .	216
24.33QuarterDateFormat . . . . .	218
24.34SegmentedTimeline . . . . .	218
24.35StandardTickUnitSource . . . . .	219
24.36SubCategoryAxis . . . . .	220
24.37SymbolAxis . . . . .	221
24.38Tick . . . . .	223
24.39TickType . . . . .	223
24.40TickUnit . . . . .	223
24.41TickUnits . . . . .	224
24.42TickUnitSource . . . . .	225
24.43Timeline . . . . .	225
24.44ValueAxis . . . . .	226
24.45ValueTick . . . . .	229
<b>25 Package: org.jfree.chart.block . . . . .</b>	<b>231</b>
25.1 Introduction . . . . .	231
25.2 AbstractBlock . . . . .	231
25.3 Arrangement . . . . .	233
25.4 Block . . . . .	234
25.5 BlockBorder . . . . .	234
25.6 BlockContainer . . . . .	235
25.7 BlockFrame . . . . .	237
25.8 BlockParams . . . . .	237
25.9 BlockResult . . . . .	237
25.10BorderArrangement . . . . .	238
25.11CenterArrangement . . . . .	238
25.12ColorBlock . . . . .	238
25.13ColumnArrangement . . . . .	239
25.14EmptyBlock . . . . .	240
25.15EntityBlockParams . . . . .	240
25.16EntityBlockResult . . . . .	240
25.17FlowArrangement . . . . .	240
25.18GridArrangement . . . . .	241
25.19LabelBlock . . . . .	241
25.20LengthConstraintType . . . . .	243
25.21LineBorder . . . . .	243
25.22RectangleConstraint . . . . .	244

<b>26 Package: org.jfree.chart.editor</b>	<b>246</b>
26.1 Introduction . . . . .	246
26.2 ChartEditor . . . . .	246
26.3 ChartEditorFactory . . . . .	246
26.4 ChartEditorManager . . . . .	247
26.5 DefaultAxisEditor . . . . .	247
26.6 DefaultChartEditor . . . . .	247
26.7 DefaultChartEditorFactory . . . . .	248
26.8 DefaultColorBarEditor . . . . .	248
26.9 DefaultNumberAxisEditor . . . . .	249
26.10 DefaultPlotEditor . . . . .	249
26.11 DefaultTitleEditor . . . . .	249
26.12 PaletteChooserPanel . . . . .	249
26.13 PaletteSample . . . . .	250
<b>27 Package: org.jfree.chart.encoders</b>	<b>251</b>
27.1 Introduction . . . . .	251
27.2 EncoderUtil . . . . .	251
27.3 ImageEncoderFactory . . . . .	252
27.4 ImageEncoder . . . . .	253
27.5 ImageFormat . . . . .	253
27.6 KeyPointPNGEncoderAdapter . . . . .	253
27.7 SunJPEGEncoderAdapter . . . . .	254
27.8 SunPNGEncoderAdapter . . . . .	255
<b>28 Package: org.jfree.chart.entity</b>	<b>256</b>
28.1 Introduction . . . . .	256
28.2 Background . . . . .	256
28.3 CategoryItemEntity . . . . .	256
28.4 ChartEntity . . . . .	258
28.5 ContourEntity . . . . .	258
28.6 EntityCollection . . . . .	259
28.7 LegendItemEntity . . . . .	259
28.8 PieSectionEntity . . . . .	260
28.9 StandardEntityCollection . . . . .	260
28.10 TickLabelEntity . . . . .	261
28.11 XYAnnotationEntity . . . . .	262
28.12 XYItemEntity . . . . .	262
<b>29 Package: org.jfree.chart.event</b>	<b>263</b>
29.1 Introduction . . . . .	263
29.2 AxisChangeEvent . . . . .	263
29.3 AxisChangeListener . . . . .	263
29.4 ChartChangeEvent . . . . .	264
29.5 ChartChangeEvent Type . . . . .	264
29.6 ChartChangeListener . . . . .	265
29.7 ChartProgressEvent . . . . .	265
29.8 ChartProgressListener . . . . .	266
29.9 MarkerChangeEvent . . . . .	266
29.10 MarkerChangeListener . . . . .	267
29.11 PlotChangeEvent . . . . .	267
29.12 PlotChangeListener . . . . .	268
29.13 RendererChangeEvent . . . . .	268
29.14 RendererChangeListener . . . . .	268
29.15 TitleChangeEvent . . . . .	269

29.16TitleChangeListener . . . . .	269
<b>30 Package: org.jfree.chart.imageimap</b>	<b>270</b>
30.1 Overview . . . . .	270
30.2 DynamicDriveToolTipTagFragmentGenerator . . . . .	270
30.3 ImageMapUtilities . . . . .	270
30.4 OverLIBToolTipTagFragmentGenerator . . . . .	271
30.5 StandardToolTipTagFragmentGenerator . . . . .	271
30.6 StandardURLTagFragmentGenerator . . . . .	272
30.7 ToolTipTagFragmentGenerator . . . . .	272
30.8 URLTagFragmentGenerator . . . . .	273
<b>31 Package: org.jfree.chart.labels</b>	<b>274</b>
31.1 Introduction . . . . .	274
31.2 AbstractCategoryItemLabelGenerator . . . . .	274
31.3 AbstractPieItemLabelGenerator . . . . .	275
31.4 AbstractXYItemLabelGenerator . . . . .	276
31.5 BoxAndWhiskerToolTipGenerator . . . . .	278
31.6 BoxAndWhiskerXYToolTipGenerator . . . . .	278
31.7 CategoryItemLabelGenerator . . . . .	278
31.8 CategorySeriesLabelGenerator . . . . .	279
31.9 CategoryToolTipGenerator . . . . .	279
31.10ContourToolTipGenerator . . . . .	280
31.11CustomXYToolTipGenerator . . . . .	280
31.12HighLowItemLabelGenerator . . . . .	280
31.13IntervalCategoryItemLabelGenerator . . . . .	281
31.14IntervalCategoryToolTipGenerator . . . . .	282
31.15ItemLabelAnchor . . . . .	282
31.16ItemLabelPosition . . . . .	283
31.17MultipleXYSeriesLabelGenerator . . . . .	284
31.18PieSectionLabelGenerator . . . . .	285
31.19PieToolTipGenerator . . . . .	286
31.20StandardCategoryItemLabelGenerator . . . . .	286
31.21StandardCategorySeriesLabelGenerator . . . . .	287
31.22StandardCategoryToolTipGenerator . . . . .	288
31.23StandardContourToolTipGenerator . . . . .	289
31.24StandardPieSectionLabelGenerator . . . . .	289
31.25StandardPieToolTipGenerator . . . . .	291
31.26StandardXYItemLabelGenerator . . . . .	292
31.27StandardXYZSeriesLabelGenerator . . . . .	293
31.28StandardXYToolTipGenerator . . . . .	294
31.29StandardXYZToolTipGenerator . . . . .	295
31.30SymbolicXYItemLabelGenerator . . . . .	296
31.31XYItemLabelGenerator . . . . .	296
31.32XYZSeriesLabelGenerator . . . . .	296
31.33XYToolTipGenerator . . . . .	297
31.34XYZToolTipGenerator . . . . .	297
<b>32 Package: org.jfree.chart.needle</b>	<b>299</b>
32.1 Overview . . . . .	299
32.2 ArrowNeedle . . . . .	299
32.3 LineNeedle . . . . .	300
32.4 LongNeedle . . . . .	300
32.5 MeterNeedle . . . . .	300
32.6 PinNeedle . . . . .	301

32.7 PlumNeedle . . . . .	301
32.8 PointerNeedle . . . . .	302
32.9 ShipNeedle . . . . .	302
32.10 WindNeedle . . . . .	302
<b>33 Package: org.jfree.chart.plot . . . . .</b>	<b>304</b>
33.1 Overview . . . . .	304
33.2 CategoryMarker . . . . .	304
33.3 CategoryPlot . . . . .	306
33.4 ColorPalette . . . . .	311
33.5 CombinedDomainCategoryPlot . . . . .	312
33.6 CombinedDomainXYPlot . . . . .	313
33.7 CombinedRangeCategoryPlot . . . . .	315
33.8 CombinedRangeXYPlot . . . . .	316
33.9 CompassPlot . . . . .	317
33.10 ContourPlot . . . . .	320
33.11 ContourPlotUtilities . . . . .	321
33.12 ContourValuePlot . . . . .	321
33.13 CrosshairState . . . . .	321
33.14 DatasetRenderingOrder . . . . .	321
33.15 DefaultDrawingSupplier . . . . .	322
33.16 DialShape . . . . .	323
33.17 DrawingSupplier . . . . .	324
33.18 FastScatterPlot . . . . .	325
33.19 GreyPalette . . . . .	328
33.20 IntervalMarker . . . . .	328
33.21 Marker . . . . .	330
33.22 MeterInterval . . . . .	334
33.23 MeterPlot . . . . .	335
33.24 MultiplePiePlot . . . . .	339
33.25 PieLabelDistributor . . . . .	342
33.26 PieLabelRecord . . . . .	342
33.27 PiePlot . . . . .	342
33.28 PiePlot3D . . . . .	354
33.29 PiePlotState . . . . .	356
33.30 Plot . . . . .	356
33.31 PlotOrientation . . . . .	361
33.32 PlotRenderingInfo . . . . .	361
33.33 PlotState . . . . .	363
33.34 PlotUtilities . . . . .	363
33.35 PolarPlot . . . . .	363
33.36 RainbowPalette . . . . .	368
33.37 RingPlot . . . . .	368
33.38 SeriesRenderingOrder . . . . .	370
33.39 SpiderWebPlot . . . . .	370
33.40 ThermometerPlot . . . . .	375
33.41 ValueAxisPlot . . . . .	382
33.42 ValueMarker . . . . .	382
33.43 WaferMapPlot . . . . .	383
33.44 XYPlot . . . . .	383
33.45 Zoomable . . . . .	393

<b>34 Package: org.jfree.chart.plot.dial</b>	<b>395</b>
34.1 Overview . . . . .	395
34.2 AbstractDialLayer . . . . .	395
34.3 ArcDialFrame . . . . .	397
34.4 DialBackground . . . . .	398
34.5 DialCap . . . . .	399
34.6 DialFrame . . . . .	401
34.7 DialLayer . . . . .	401
34.8 DialLayerChangeEvent . . . . .	402
34.9 DialLayerChangeListener . . . . .	403
34.10 DialPlot . . . . .	403
34.11 DialPointer . . . . .	406
34.12 DialPointer.Pin . . . . .	407
34.13 DialPointer.Pointer . . . . .	408
34.14 DialScale . . . . .	409
34.15 DialTextAnnotation . . . . .	410
34.16 DialValueIndicator . . . . .	411
34.17 SimpleDialFrame . . . . .	414
34.18 StandardDialRange . . . . .	415
34.19 StandardDialScale . . . . .	417
<b>35 Package: org.jfree.chart.renderer</b>	<b>421</b>
35.1 Overview . . . . .	421
35.2 AbstractRenderer . . . . .	421
35.3 AreaRendererEndType . . . . .	440
35.4 DefaultPolarItemRenderer . . . . .	440
35.5 GrayPaintScale . . . . .	441
35.6 LookupPaintScale . . . . .	442
35.7 NotOutlierException . . . . .	443
35.8 Outlier . . . . .	443
35.9 OutlierList . . . . .	444
35.10 OutlierListCollection . . . . .	444
35.11 PaintScale . . . . .	444
35.12 PolarItemRenderer . . . . .	445
35.13 RendererState . . . . .	446
35.14 WaferMapRenderer . . . . .	446
<b>36 Package: org.jfree.chart.renderer.category</b>	<b>447</b>
36.1 Overview . . . . .	447
36.2 AbstractCategoryItemRenderer . . . . .	447
36.3 AreaRenderer . . . . .	451
36.4 BarRenderer . . . . .	452
36.5 BarRenderer3D . . . . .	457
36.6 BoxAndWhiskerRenderer . . . . .	459
36.7 CategoryItemRenderer . . . . .	460
36.8 CategoryItemRendererState . . . . .	470
36.9 CategoryStepRenderer . . . . .	471
36.10 DefaultCategoryItemRenderer . . . . .	473
36.11 GanttRenderer . . . . .	473
36.12 GroupedStackedBarRenderer . . . . .	474
36.13 IntervalBarRenderer . . . . .	475
36.14 LayeredBarRenderer . . . . .	476
36.15 LevelRenderer . . . . .	477
36.16 LineAndShapeRenderer . . . . .	479
36.17 LineRenderer3D . . . . .	483

36.18MinMaxCategoryRenderer . . . . .	485
36.19ScatterRenderer . . . . .	487
36.20StackedAreaRenderer . . . . .	490
36.21StackedBarRenderer . . . . .	490
36.22StackedBarRenderer3D . . . . .	492
36.23StatisticalBarRenderer . . . . .	494
36.24StatisticalLineAndShapeRenderer . . . . .	495
36.25WaterfallBarRenderer . . . . .	497
<b>37 Package: org.jfree.chart.renderer.xy</b>	<b>500</b>
37.1 Overview . . . . .	500
37.2 AbstractXYItemRenderer . . . . .	500
37.3 CandlestickRenderer . . . . .	505
37.4 ClusteredXYBarRenderer . . . . .	508
37.5 CyclicXYItemRenderer . . . . .	509
37.6 DefaultXYItemRenderer . . . . .	510
37.7 DeviationRenderer . . . . .	510
37.8 HighLowRenderer . . . . .	512
37.9 StackedXYAreaRenderer . . . . .	513
37.10StackedXYAreaRenderer2 . . . . .	515
37.11StackedXYBarRenderer . . . . .	516
37.12StandardXYItemRenderer . . . . .	518
37.13VectorRenderer . . . . .	521
37.14WindItemRenderer . . . . .	523
37.15XYAreaRenderer . . . . .	523
37.16XYBarRenderer . . . . .	525
37.17XYBlockRenderer . . . . .	527
37.18XYBoxAndWhiskerRenderer . . . . .	529
37.19XYBubbleRenderer . . . . .	530
37.20XYDifferenceRenderer . . . . .	532
37.21XYDotRenderer . . . . .	534
37.22XYErrorRenderer . . . . .	535
37.23XYItemRenderer . . . . .	537
37.24XYItemRendererState . . . . .	547
37.25XYLineAndShapeRenderer . . . . .	548
37.26XYSplineRenderer . . . . .	554
37.27XYStepRenderer . . . . .	556
37.28XYStepAreaRenderer . . . . .	557
37.29YIntervalRenderer . . . . .	558
<b>38 Package: org.jfree.chart.servlet</b>	<b>560</b>
38.1 Overview . . . . .	560
38.2 ChartDeleter . . . . .	560
38.3 DisplayChart . . . . .	560
38.4 ServletUtilities . . . . .	560
<b>39 Package: org.jfree.chart.title</b>	<b>562</b>
39.1 Overview . . . . .	562
39.2 Events . . . . .	562
39.3 CompositeTitle . . . . .	562
39.4 DateTitle . . . . .	563
39.5 ImageTitle . . . . .	564
39.6 LegendGraphic . . . . .	564
39.7 LegendItemBlockContainer . . . . .	566
39.8 LegendTitle . . . . .	567

39.9 PaintScaleLegend . . . . .	570
39.10 TextTitle . . . . .	572
39.11 Title . . . . .	574
<b>40 Package: org.jfree.chart.urls</b>	<b>577</b>
40.1 Overview . . . . .	577
40.2 CategoryURLGenerator . . . . .	577
40.3 CustomPieURLGenerator . . . . .	578
40.4 CustomXYZURLGenerator . . . . .	579
40.5 PieURLGenerator . . . . .	579
40.6 StandardCategoryURLGenerator . . . . .	579
40.7 StandardPieURLGenerator . . . . .	581
40.8 StandardXYZURLGenerator . . . . .	582
40.9 StandardXYZURLGenerator . . . . .	583
40.10 TimeSeriesURLGenerator . . . . .	583
40.11 URLUtilities . . . . .	583
40.12 XYURLGenerator . . . . .	583
40.13 XYZURLGenerator . . . . .	584
<b>41 Package: org.jfree.chart.util</b>	<b>585</b>
41.1 Overview . . . . .	585
41.2 RelativeDateFormat . . . . .	585
<b>42 Package: org.jfree.data</b>	<b>588</b>
42.1 Introduction . . . . .	588
42.2 ComparableObjectItem . . . . .	588
42.3 ComparableObjectSeries . . . . .	589
42.4 DataUtilities . . . . .	590
42.5 DefaultKeyedValue . . . . .	591
42.6 DefaultKeyedValues . . . . .	592
42.7 DefaultKeyedValues2D . . . . .	594
42.8 DomainInfo . . . . .	596
42.9 DomainOrder . . . . .	596
42.10 KeyedObject . . . . .	597
42.11 KeyedObjects . . . . .	597
42.12 KeyedObjects2D . . . . .	599
42.13 KeyedValue . . . . .	601
42.14 KeyedValueComparator . . . . .	601
42.15 KeyedValueComparatorType . . . . .	601
42.16 KeyedValues . . . . .	602
42.17 KeyedValues2D . . . . .	602
42.18 KeyToGroupMap . . . . .	603
42.19 Range . . . . .	604
42.20 RangeInfo . . . . .	606
42.21 RangeType . . . . .	606
42.22 UnknownKeyException . . . . .	607
42.23 Value . . . . .	607
42.24 Values . . . . .	607
42.25 Values2D . . . . .	608

<b>43 Package: org.jfree.data.category</b>	<b>609</b>
43.1 Introduction . . . . .	609
43.2 CategoryDataset . . . . .	609
43.3 CategoryToPieDataset . . . . .	609
43.4 DefaultCategoryDataset . . . . .	611
43.5 DefaultIntervalCategoryDataset . . . . .	613
43.6 IntervalCategoryDataset . . . . .	616
<b>44 Package: org.jfree.data.contour</b>	<b>617</b>
44.1 Introduction . . . . .	617
44.2 ContourDataset . . . . .	617
44.3 DefaultContourDataset . . . . .	618
44.4 NonGridContourDataset . . . . .	618
<b>45 Package: org.jfree.data.function</b>	<b>619</b>
45.1 Introduction . . . . .	619
45.2 Function2D . . . . .	619
45.3 LineFunction2D . . . . .	619
45.4 NormalDistributionFunction2D . . . . .	620
45.5 PowerFunction2D . . . . .	621
<b>46 Package: org.jfree.data.gantt</b>	<b>622</b>
46.1 Introduction . . . . .	622
46.2 GanttCategoryDataset . . . . .	622
46.3 Task . . . . .	623
46.4 TaskSeries . . . . .	624
46.5 TaskSeriesCollection . . . . .	625
<b>47 Package: org.jfree.data.general</b>	<b>628</b>
47.1 Introduction . . . . .	628
47.2 AbstractDataset . . . . .	628
47.3 AbstractSeriesDataset . . . . .	629
47.4 CombinationDataset . . . . .	630
47.5 CombinedDataset . . . . .	630
47.6 Dataset . . . . .	630
47.7 DatasetChangeEvent . . . . .	631
47.8 DatasetChangeListener . . . . .	631
47.9 DatasetGroup . . . . .	631
47.10DatasetUtilities . . . . .	632
47.11DefaultKeyedValueDataset . . . . .	636
47.12DefaultKeyedValuesDataset . . . . .	636
47.13DefaultKeyedValues2DDataset . . . . .	636
47.14DefaultPieDataset . . . . .	636
47.15DefaultValueDataset . . . . .	638
47.16KeyValueDataset . . . . .	638
47.17KeyedValuesDataset . . . . .	639
47.18KeyedValues2DDataset . . . . .	639
47.19PieDataset . . . . .	639
47.20Series . . . . .	640
47.21SeriesChangeEvent . . . . .	642
47.22SeriesChangeListener . . . . .	642
47.23SeriesDataset . . . . .	642
47.24SeriesException . . . . .	643
47.25SubSeriesDataset . . . . .	643
47.26ValueDataset . . . . .	643

47.27WaferMapDataset . . . . .	644
<b>48 Package: org.jfree.data.io</b>	<b>645</b>
48.1 Introduction . . . . .	645
48.2 CSV . . . . .	645
<b>49 Package: org.jfree.data.jdbc</b>	<b>646</b>
49.1 Introduction . . . . .	646
49.2 JDBCCategoryDataset . . . . .	646
49.3 JDBCPieDataset . . . . .	647
49.4 JDBCXYDataset . . . . .	647
<b>50 Package: org.jfree.data.statistics</b>	<b>649</b>
50.1 Introduction . . . . .	649
50.2 BoxAndWhiskerCalculator . . . . .	649
50.3 BoxAndWhiskerCategoryDataset . . . . .	650
50.4 BoxAndWhiskerItem . . . . .	651
50.5 BoxAndWhiskerXYDataset . . . . .	652
50.6 DefaultBoxAndWhiskerCategoryDataset . . . . .	653
50.7 DefaultBoxAndWhiskerXYDataset . . . . .	656
50.8 DefaultMultiValueCategoryDataset . . . . .	658
50.9 DefaultStatisticalCategoryDataset . . . . .	660
50.10 HistogramBin . . . . .	662
50.11 HistogramDataset . . . . .	663
50.12 HistogramType . . . . .	665
50.13 MeanAndStandardDeviation . . . . .	666
50.14 MultiValueCategoryDataset . . . . .	666
50.15 Regression . . . . .	667
50.16 SimpleHistogramBin . . . . .	668
50.17 SimpleHistogramDataset . . . . .	669
50.18 StatisticalCategoryDataset . . . . .	671
50.19 Statistics . . . . .	671
<b>51 Package: org.jfree.data.time</b>	<b>674</b>
51.1 Introduction . . . . .	674
51.2 DateRange . . . . .	674
51.3 Day . . . . .	674
51.4 DynamicTimeSeriesCollection . . . . .	676
51.5 FixedMillisecond . . . . .	678
51.6 Hour . . . . .	678
51.7 Millisecond . . . . .	679
51.8 Minute . . . . .	680
51.9 Month . . . . .	681
51.10 MovingAverage . . . . .	682
51.11 Quarter . . . . .	683
51.12 RegularTimePeriod . . . . .	684
51.13 Second . . . . .	686
51.14 SimpleTimePeriod . . . . .	687
51.15 TimePeriod . . . . .	687
51.16 TimePeriodAnchor . . . . .	688
51.17 TimePeriodFormatException . . . . .	688
51.18 TimePeriodValue . . . . .	688
51.19 TimePeriodValues . . . . .	689
51.20 TimePeriodValuesCollection . . . . .	691
51.21 TimeSeries . . . . .	693

51.22TimeSeriesCollection . . . . .	698
51.23TimeSeriesDataItem . . . . .	701
51.24TimeSeriesTableModel . . . . .	702
51.25TimeTableXYDataset . . . . .	702
51.26Week . . . . .	704
51.27Year . . . . .	705
<b>52 Package: org.jfree.data.time.ohlc</b>	<b>707</b>
52.1 Introduction . . . . .	707
52.2 OHLC . . . . .	707
52.3 OHLCItem . . . . .	708
52.4 OHLCSeries . . . . .	709
52.5 OHLCSeriesCollection . . . . .	709
<b>53 Package: org.jfree.data.xml</b>	<b>712</b>
53.1 Introduction . . . . .	712
53.2 Usage . . . . .	712
53.3 CategoryDatasetHandler . . . . .	712
53.4 CategorySeriesHandler . . . . .	713
53.5 DatasetReader . . . . .	713
53.6 DatasetTags . . . . .	713
53.7 ItemHandler . . . . .	714
53.8 KeyHandler . . . . .	714
53.9 PieDatasetHandler . . . . .	714
53.10RootHandler . . . . .	715
53.11ValueHandler . . . . .	715
<b>54 Package: org.jfree.data.xy</b>	<b>716</b>
54.1 Introduction . . . . .	716
54.2 AbstractIntervalXYDataset . . . . .	716
54.3 AbstractXYZDataset . . . . .	716
54.4 AbstractXYZDataset . . . . .	717
54.5 CategoryTableXYDataset . . . . .	717
54.6 DefaultHighLowDataset . . . . .	719
54.7 DefaultIntervalXYDataset . . . . .	720
54.8 DefaultOHLCDataset . . . . .	723
54.9 DefaultTableXYDataset . . . . .	724
54.10DefaultWindDataset . . . . .	726
54.11DefaultXYDataset . . . . .	728
54.12DefaultXYZDataset . . . . .	730
54.13IntervalXYDataset . . . . .	732
54.14IntervalXYDelegate . . . . .	733
54.15IntervalXYZDataset . . . . .	734
54.16MatrixSeries . . . . .	734
54.17MatrixSeriesCollection . . . . .	735
54.18NormalizedMatrixSeries . . . . .	736
54.19OHLCDataItem . . . . .	737
54.20OHLCDataset . . . . .	738
54.21TableXYDataset . . . . .	739
54.22Vector . . . . .	739
54.23VectorDataItem . . . . .	740
54.24VectorSeries . . . . .	741
54.25VectorSeriesCollection . . . . .	742
54.26VectorXYDataset . . . . .	744
54.27WindDataset . . . . .	744

54.28XisSymbolic . . . . .	745
54.29XYBarDataset . . . . .	745
54.30XYCoordinate . . . . .	747
54.31XYDataItem . . . . .	748
54.32XYDataset . . . . .	748
54.33XYDatasetTableModel . . . . .	750
54.34XYInterval . . . . .	751
54.35XYIntervalDataItem . . . . .	752
54.36XYIntervalSeries . . . . .	753
54.37XYIntervalSeriesCollection . . . . .	754
54.38XYSeries . . . . .	756
54.39XYSeriesCollection . . . . .	759
54.40XYZDataset . . . . .	761
54.41YInterval . . . . .	761
54.42YIntervalDataItem . . . . .	762
54.43YIntervalSeries . . . . .	763
54.44YIntervalSeriesCollection . . . . .	764
54.45YisSymbolic . . . . .	765
<b>A Migration</b>	<b>767</b>
A.1 Introduction . . . . .	767
A.2 1.0.8 to 1.0.9 . . . . .	767
A.3 1.0.7 to 1.0.8 . . . . .	768
A.4 1.0.6 to 1.0.7 . . . . .	768
A.5 1.0.5 to 1.0.6 . . . . .	769
A.6 1.0.4 to 1.0.5 . . . . .	770
A.7 1.0.3 to 1.0.4 . . . . .	771
A.8 1.0.2 to 1.0.3 . . . . .	772
A.9 1.0.1 to 1.0.2 . . . . .	773
A.10 1.0.0 to 1.0.1 . . . . .	773
A.11 0.9.x to 1.0.0 . . . . .	774
<b>B JCommon</b>	<b>775</b>
B.1 Introduction . . . . .	775
B.2 Align . . . . .	775
B.3 GradientPaintTransformer . . . . .	776
B.4 GradientPaintTransformType . . . . .	776
B.5 PublicCloneable . . . . .	776
B.6 RectangleAnchor . . . . .	777
B.7 RectangleEdge . . . . .	777
B.8 RectangleInsets . . . . .	777
B.9 StandardGradientPaintTransformer . . . . .	779
B.10 TextAnchor . . . . .	781
B.11 UnitType . . . . .	781
<b>C Configuring IDEs for JFreeChart</b>	<b>783</b>
C.1 Introduction . . . . .	783
C.2 Eclipse . . . . .	783
C.3 NetBeans . . . . .	787
<b>D The GNU Lesser General Public Licence</b>	<b>790</b>
D.1 Introduction . . . . .	790
D.2 The Licence . . . . .	790
D.3 Frequently Asked Questions . . . . .	796

# Chapter 1

## Introduction

### 1.1 What is JFreeChart?

#### 1.1.1 Overview

JFreeChart is a free chart library for the Java(tm) platform. It is designed for use in applications, applets, servlets and JSP. JFreeChart is distributed with complete source code subject to the terms of the GNU Lesser General Public Licence, which permits JFreeChart to be used in proprietary or free software applications (see Appendix D for details).

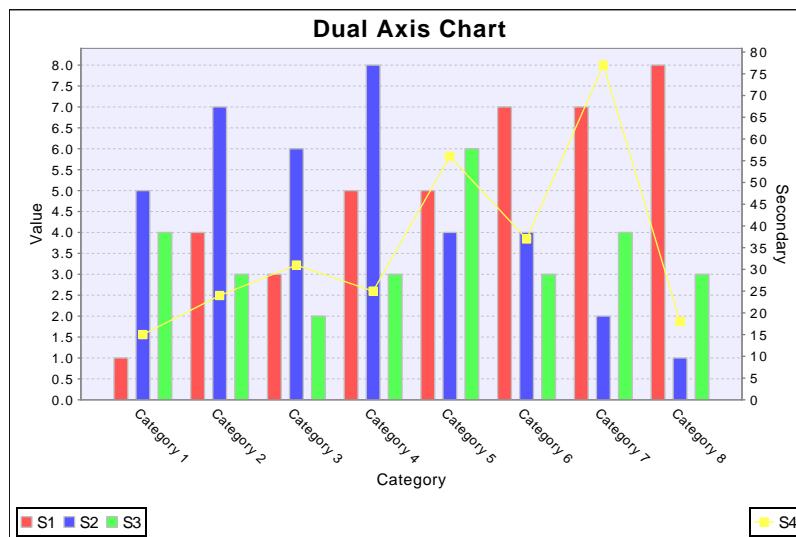


Figure 1.1: A sample chart

Figure 1.1 shows a typical chart created using JFreeChart. Many more examples are shown in later sections of this document.

#### 1.1.2 Features

JFreeChart can generate pie charts, bar charts (regular and stacked, with an optional 3D-effect), line charts, scatter plots, time series charts (including moving averages, high-low-open-close charts and candlestick plots), Gantt charts, meter charts (dial, compass and thermometer), symbol charts, wind plots, combination charts and more.

Additional features include:

- data is accessible from any implementation of the defined interfaces;
- export to PNG and JPEG image file formats (or you can use Java's ImageIO library to export to any format supported by ImageIO);
- export to any format with a `Graphics2D` implementation including:
  - PDF via iText (<http://www.lowagie.com/iText/>);
  - SVG via Batik (<http://xml.apache.org/batik/>);
- tool tips;
- interactive zooming;
- chart mouse events (these can be used for drill-down charts or information pop-ups);
- annotations;
- HTML image map generation;
- works in applications, servlets, JSP (thanks to the Cewolf project<sup>1</sup>) and applets;
- distributed with complete source code subject to the terms of the [GNU Lesser General Public License](#) (LGPL);

JFreeChart is written entirely in Java, and should run on any implementation of the Java 2 platform (JDK 1.3.1 or later). It will also work quite well with free runtimes based on GNU Classpath 0.92 or later.<sup>2</sup>

### 1.1.3 Home Page

The JFreeChart home page can be found at:

<http://www.jfree.org/jfreechart/>

Here you will find all the latest information about JFreeChart, including sample charts, download links, Javadocs, a discussion forum and more.

---

<sup>1</sup>See <http://cewolf.sourceforge.net> for details.

<sup>2</sup>See <http://www.gnu.org/software/classpath/> for details.

## 1.2 This Document

### 1.2.1 Versions

Two versions of this document are available:

- a free version, the “JFreeChart Installation Guide”, is available from the JFreeChart home page, and contains chapters up to and including the instructions for installing JFreeChart and running the demo;
- a premium version, the “JFreeChart Developer Guide”, is available only to those that have paid for it, and includes additional tutorial chapters and reference documentation for the JFreeChart classes.

If you wish to purchase the latter version, please visit the following site:

<http://www.object-refinery.com/jfreechart/guide.html>

We'd like to thank everyone that has supported JFreeChart in the past by purchasing the JFreeChart Developer Guide!

### 1.2.2 Disclaimer

Please note that I have put in considerable effort to ensure that the information in this document is up-to-date and accurate, but I cannot guarantee that it does not contain errors. You must use this document *at your own risk or not use it at all*.

## 1.3 Acknowledgements

JFreeChart contains code and ideas from many people. At the risk of missing someone out, I would like to thank the following people for contributing to the project:

Eric Alexander, Richard Atkinson, David Basten, David Berry, Chris Boek, Zoheb Borbora, Anthony Boulestreau, Jeremy Bowman, Daniel Bridenbecker, Nicolas Brodu, Jody Brownell, David Browning, Søren Caspersen, Chuanhao Chiu, Brian Cole, Pascal Collet, Martin Cordova, Paolo Cova, Michael Duffy, Don Elliott, Rune Fausk, Jonathan Gabbai, Serge V. Grachov, Daniel Gredler, Hans-Jurgen Greiner, Joao Guilherme Del Valle, Nick Guenther, Aiman Han, Cameron Hayne, Jon Iles, Wolfgang Irler, Sergei Ivanov, Adrian Joubert, Darren Jung, Xun Kang, Bill Kelemen, Norbert Kiesel, Gideon Krause, Pierre-Marie Le Biot, Arnaud Lelievre, Wolfgang Lenhard, David Li, Yan Liu, Tin Luu, Craig MacFarlane, Achilleus Mantzios, Thomas Meier, Aaron Metzger, Jim Moore, Jonathan Nash, Barak Naveh, David M. O'Donnell, Krzysztof Paz, Tomer Peretz, Xavier Poinsard, Andrzej Porebski, Luke Quinane, Viktor Rajewski, Eduardo Ramalho, Michael Rauch, Cameron Riley, Klaus Rheinwald, Dan Rivett, Scott Sams, Michel Santos, Thierry Saura, Andreas Schneider, Jean-Luc Schwab, Bryan Scott, Tobias Self, Mofeed Shahin, Pady Srinivasan, Greg Steckman, Roger Studner, Gerald Struck, Irv Thomae, Eric Thomas, Rich Unger, Daniel van Enckevort, Laurence Vanhelsumé, Sylvain Vieujot, Jelai Wang, Mark Watson, Alex Weber, Richard West, Matthew Wright, Benoit Xhenseval, Christian W. Zuckschwerdt, Hari and Sam (oldman).

## 1.4 Comments and Suggestions

If you have any comments or suggestions regarding this document, please send e-mail to:

[david.gilbert@object-refinery.com](mailto:david.gilbert@object-refinery.com)

# Chapter 2

## Sample Charts

### 2.1 Introduction

This section shows some sample charts created using JFreeChart. It is intended to give a reasonable overview of the types of charts that JFreeChart can generate. For other examples, please run the demo application included in the JFreeChart distribution:

```
java -jar jfreechart-1.0.9-demo.jar
```

The complete source code for the demo application is available to purchasers of the JFreeChart Developer Guide.<sup>1</sup>

### 2.2 Pie Charts

JFreeChart can create *pie charts* using any data that conforms to the `PieDataset` interface. Figure 2.1 shows a simple pie chart.

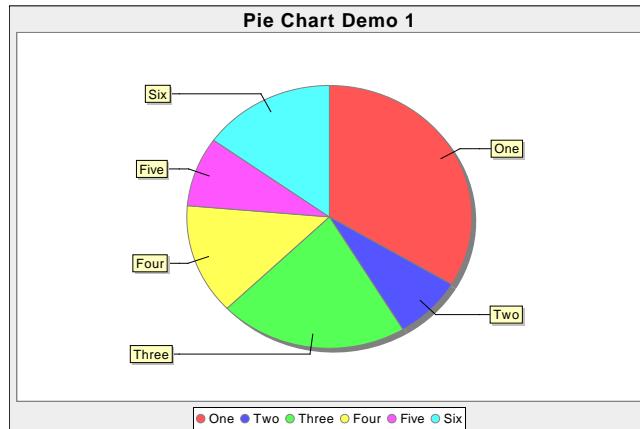


Figure 2.1: A simple pie chart (see `PieChartDemo1.java`)

---

<sup>1</sup>See <http://www.object-refinery.com/jfreechart/guide.html> for details.

Individual pie sections can be “exploded”, as shown in figure 2.2.

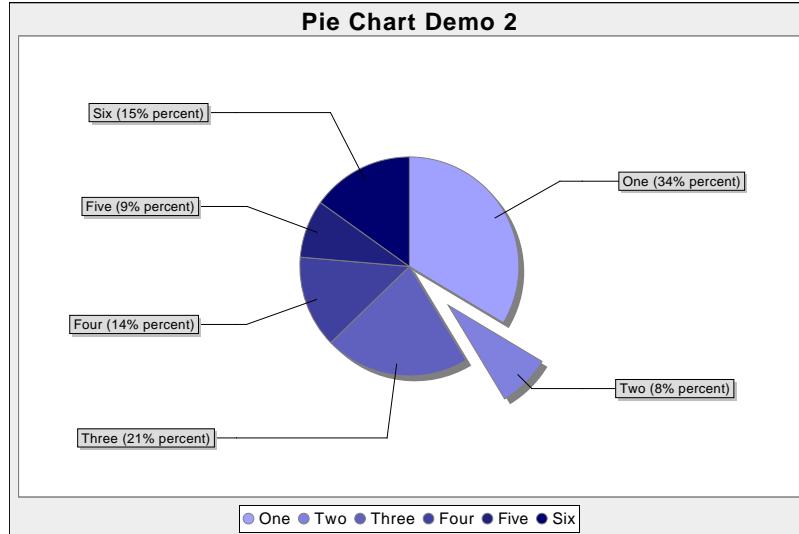


Figure 2.2: A pie chart with an “exploded” section (see `PieChartDemo2.java`)

You can also display pie charts with a 3D effect, as shown in figure 2.3.

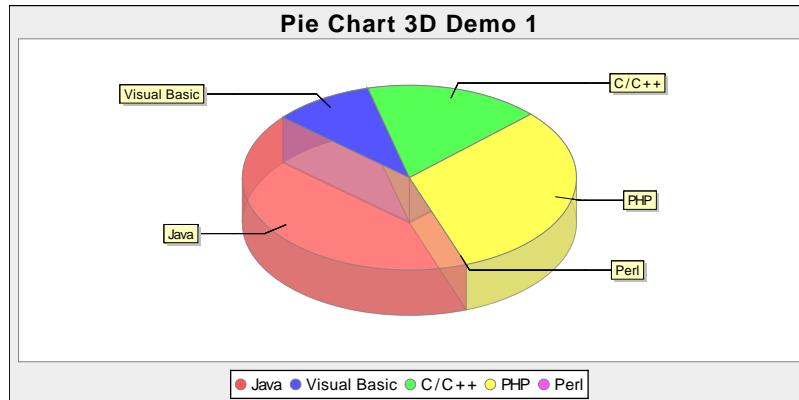


Figure 2.3: A pie chart drawn with a 3D effect (see `PieChart3DDemo1.java`)

At the current time it is *not* possible to explode sections of the 3D pie chart.

## 2.3 Bar Charts

A range of bar charts can be created with JFreeChart, using any data that conforms to the [CategoryDataset](#) interface. Figure 2.4 shows a bar chart with a vertical orientation.

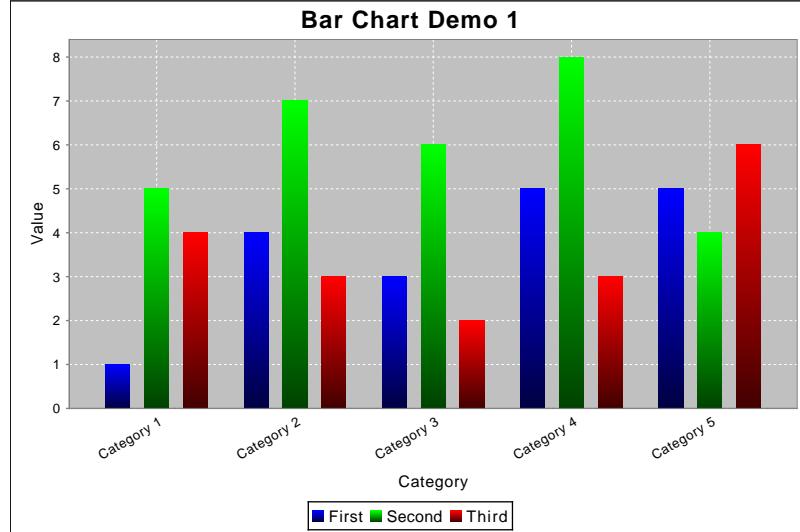


Figure 2.4: A vertical bar chart (see [BarChartDemo1.java](#))

Bar charts can be displayed with a 3D effect as shown in figure 2.5.

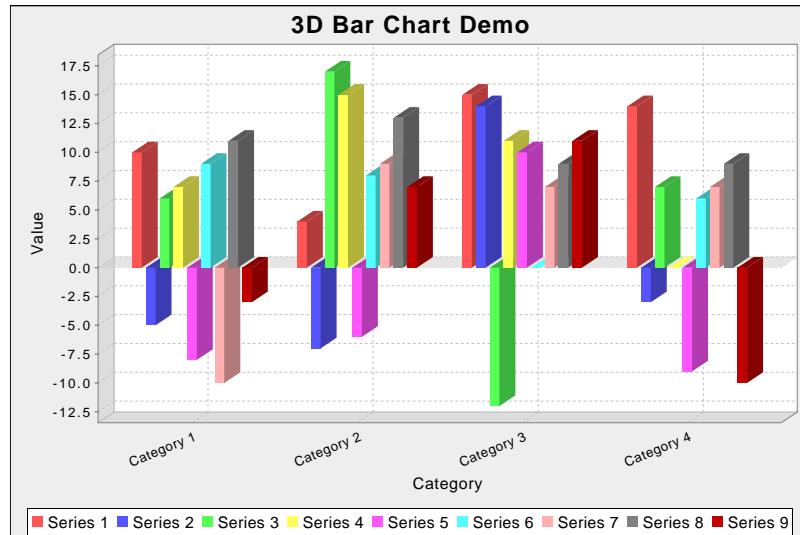


Figure 2.5: A bar chart with 3D effect (see [BarChart3DDemo1.java](#))

Another variation, the *waterfall chart*, is shown in figure 2.6.

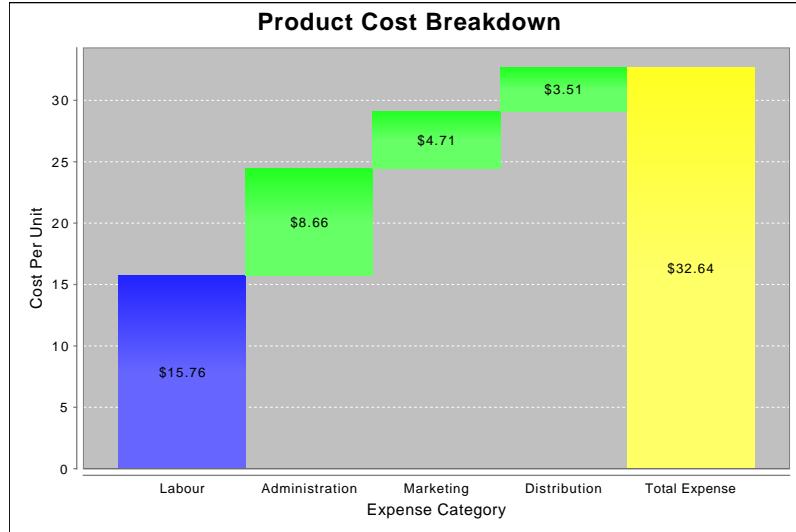


Figure 2.6: A waterfall chart (see `WaterfallChartDemo1.java`)

Bar charts can also be generated from time series data—for example, see figure 2.7:

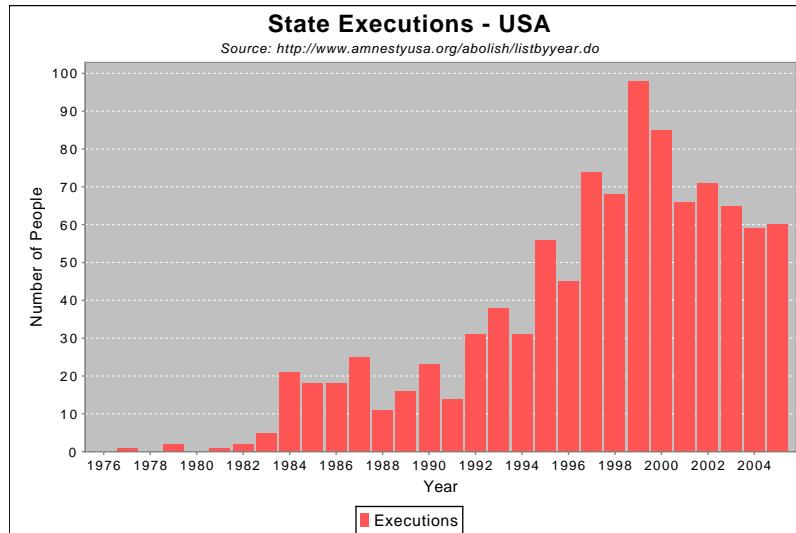


Figure 2.7: An XY bar chart (see `XYBarChartDemo1.java`)

## 2.4 Line Chart

The *line chart* can be generated using the same [CategoryDataset](#) that is used for the bar charts—figure 2.8 shows an example.

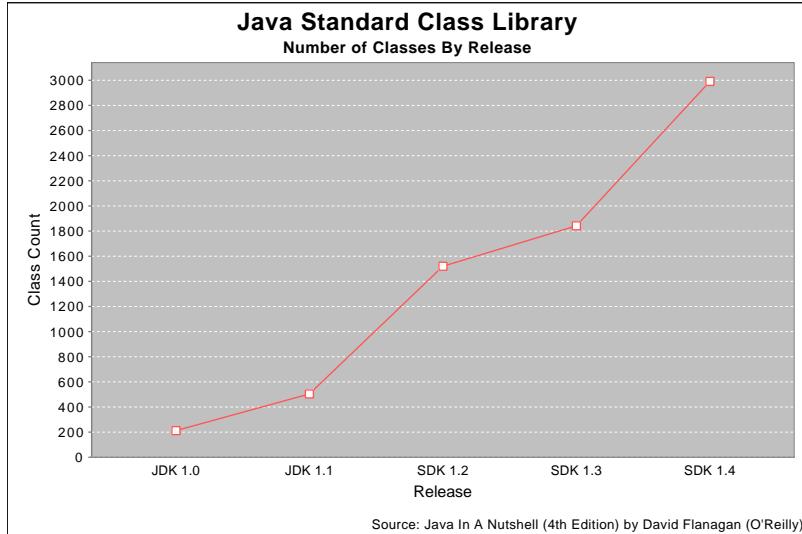


Figure 2.8: A line chart (see *LineChartDemo1.java*)

## 2.5 XY Plots

A third type of dataset, the `XYDataset`, is used to generate a range of chart types.

The standard *XY plot* has numerical x and y axes. By default, lines are drawn between each data point—see figure 2.9.

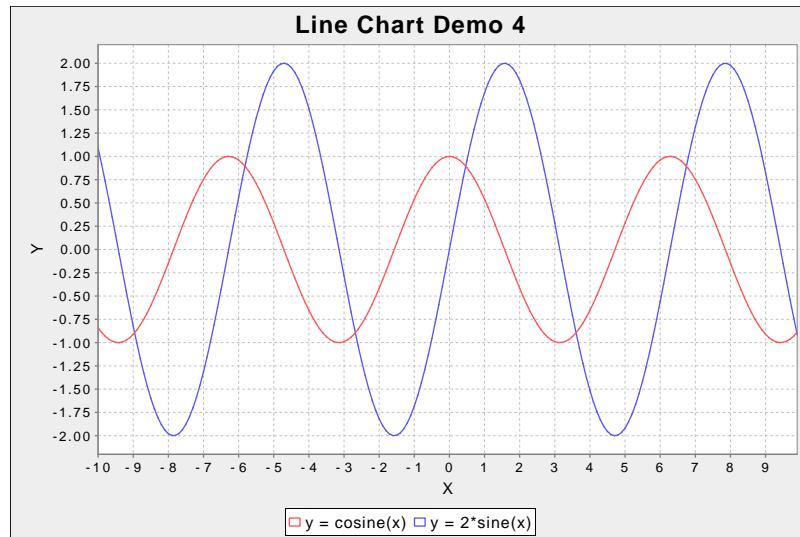


Figure 2.9: A line chart (see `LineChartDemo4.java`)

Scatter plots can be drawn by drawing a shape at each data point, rather than connecting the points with lines—an example is shown in figure 2.10.

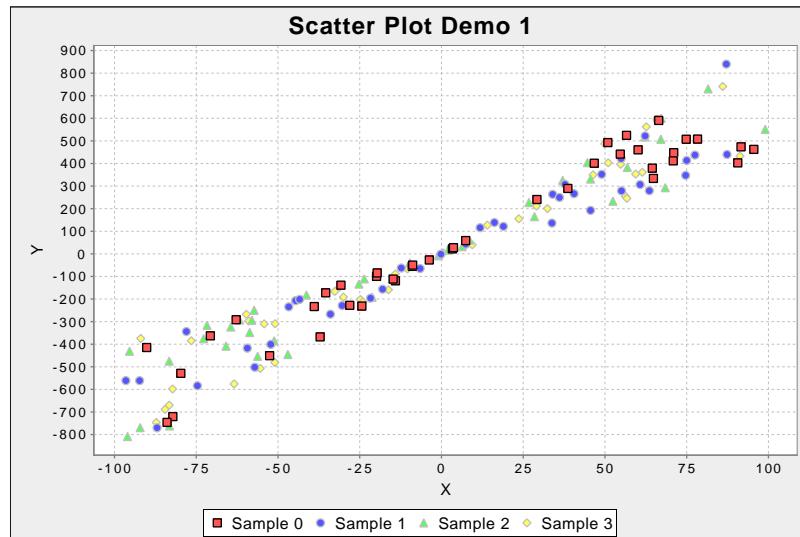


Figure 2.10: A scatter plot (see `ScatterPlotDemo1.java`)

## 2.6 Time Series Charts

JFreeChart supports *time series charts*, as shown in figure 2.11.

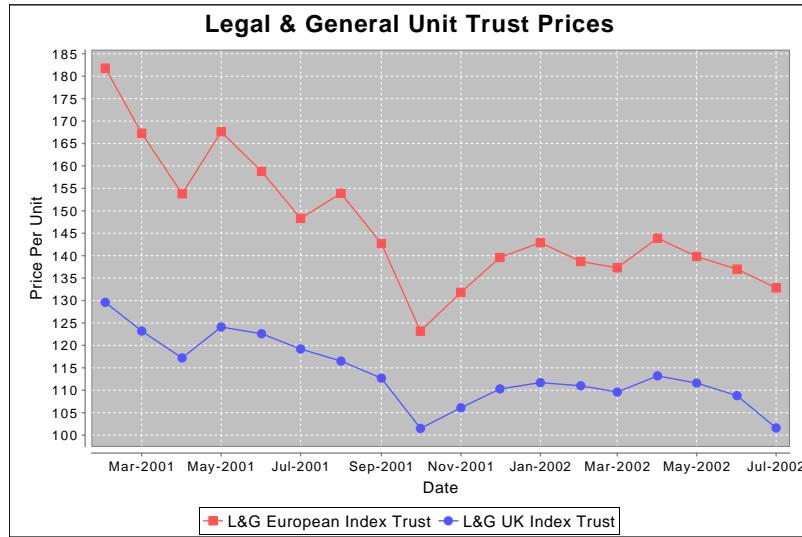


Figure 2.11: A time series chart (see `TimeSeriesDemo1.java`)

It is straightforward to add a moving average line to a time series chart—see figure 2.12 for an example.

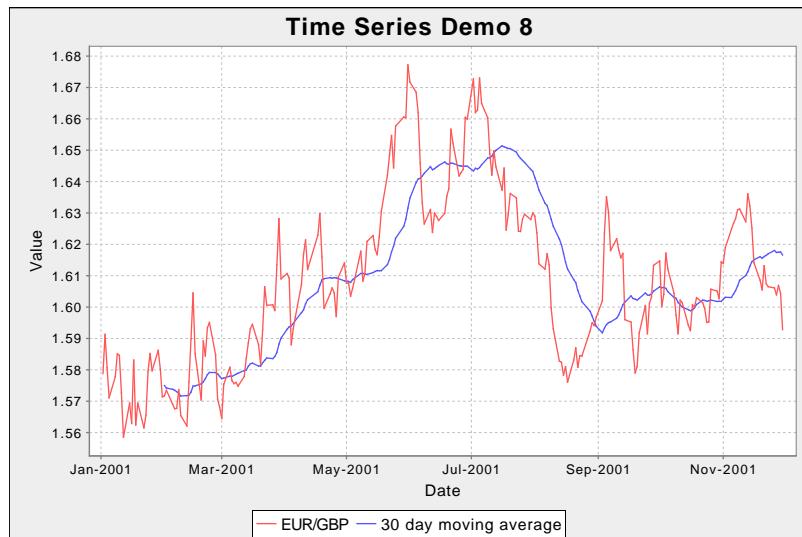


Figure 2.12: A time series chart with a moving average (see `TimeSeriesDemo8.java`)

Using an [OHLCdataset](#) (an extension of [XYDataset](#)) you can display *high-low-open-close* data, see figure 2.13 for an example.

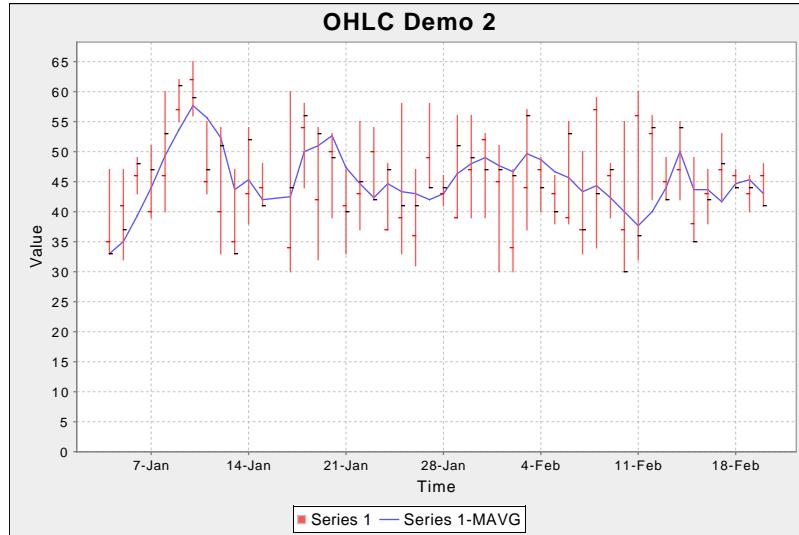


Figure 2.13: A high-low-open-close chart (see `HighLowChartDemo2.java`)

## 2.7 Histograms

Histograms can be generated using an [IntervalXYDataset](#) (another extension of [XYDataset](#)), see figure 2.14 for an example.

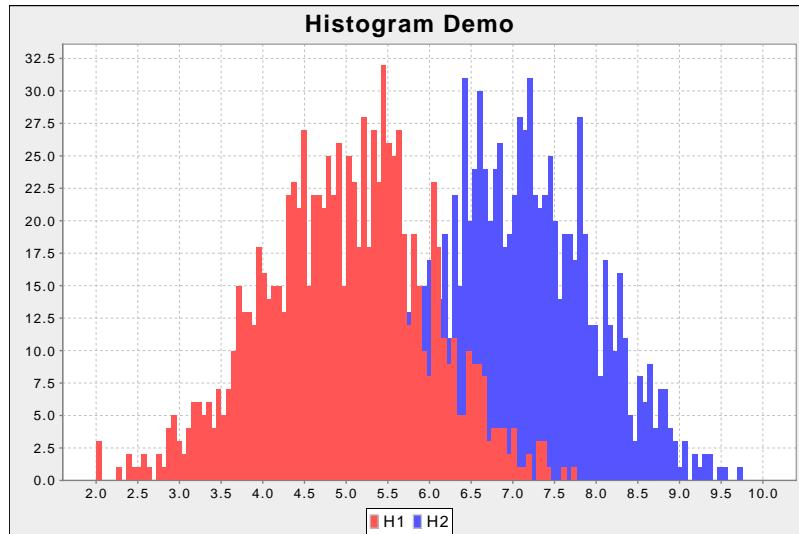


Figure 2.14: A histogram (see `HistogramDemo1.java`)

## 2.8 Area Charts

You can generate an *area chart* for data in a [CategoryDataset](#) or an [XYDataset](#). Figure 2.15 shows an example.

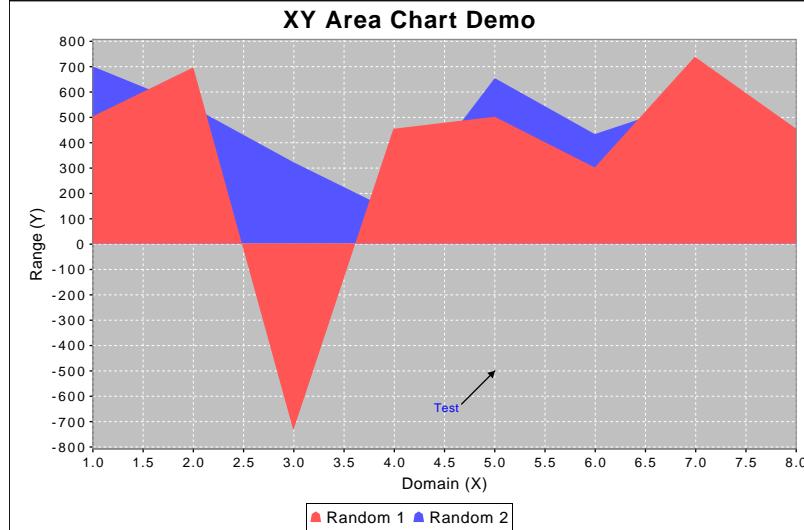


Figure 2.15: An area chart (see [XYAreaChartDemo1.java](#))

JFreeChart also supports the creation of *stacked area charts* as shown in figure 2.16.

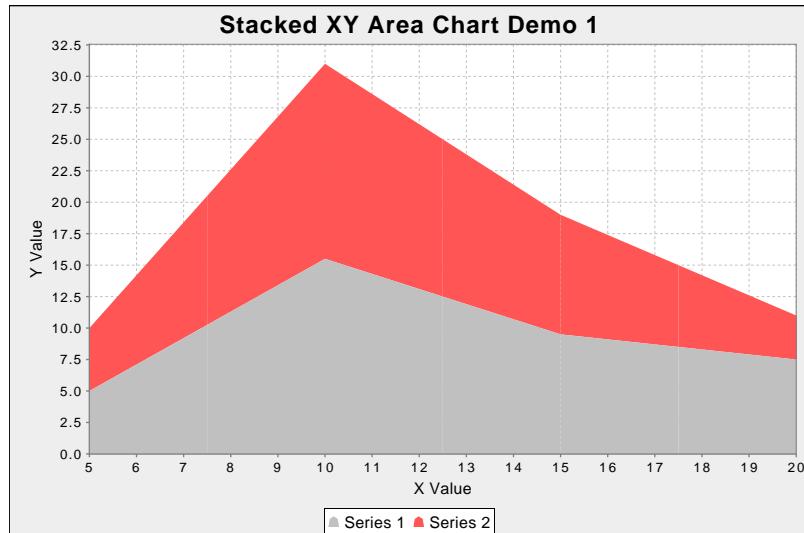


Figure 2.16: A stacked area chart (see [StackedXYAreaChartDemo1.java](#))

## 2.9 Difference Chart

A *difference chart* highlights the difference between two series (see figure 2.17).

A second example, shown in figure 2.18 shows how a date axis can be used for the range values.

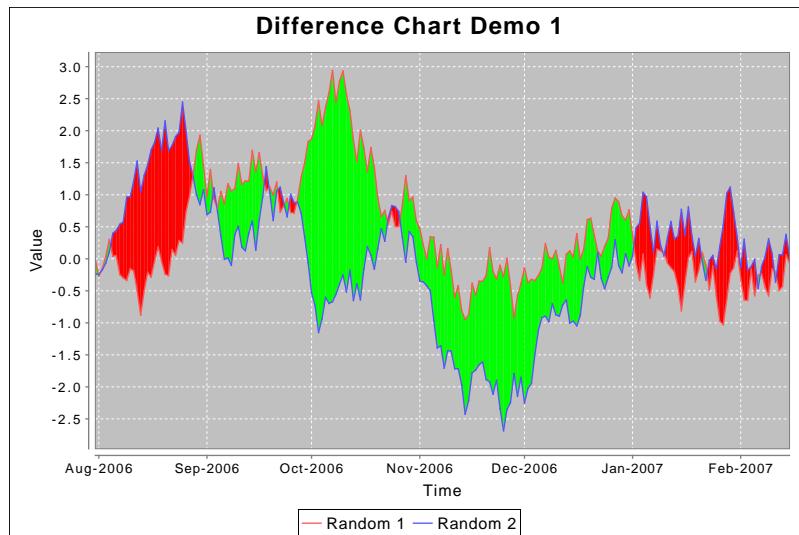


Figure 2.17: A difference chart (see *DifferenceChartDemo1.java*)

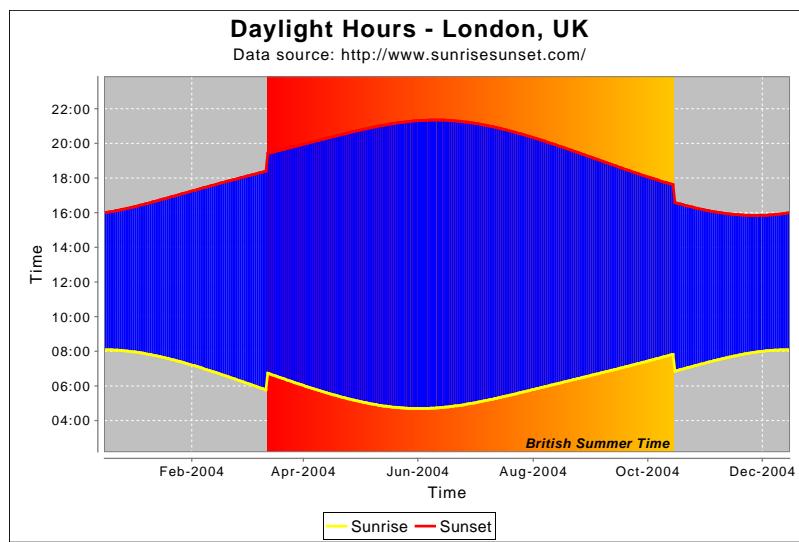


Figure 2.18: A difference chart with times on the range axis (see *DifferenceChartDemo2.java*)

## 2.10 Step Chart

A *step chart* displays numerical data as a sequence of “steps”—an example is shown in figure 2.19.

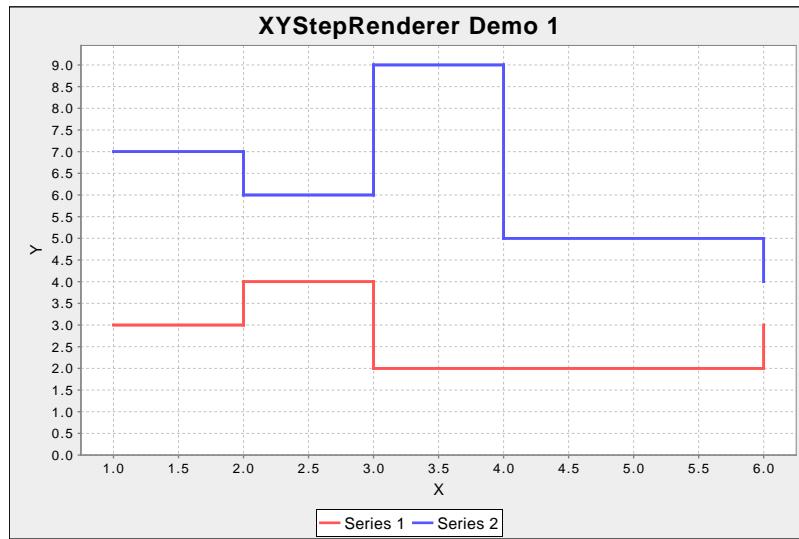


Figure 2.19: A step chart (see `XYStepRendererDemo1.java`)

Step charts are generated from data in an [XYDataset](#).

## 2.11 Gantt Chart

*Gantt charts* can be generated using data from an [IntervalCategoryDataset](#), as shown in figure [2.20](#).

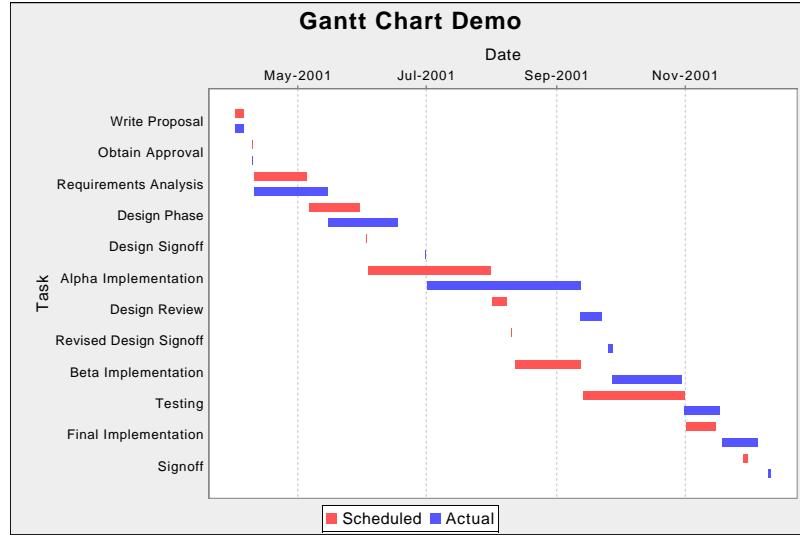


Figure 2.20: A Gantt chart (see [GanttChartDemo1.java](#))

Another example, showing subtasks and progress indicators, is shown in figure [2.21](#).

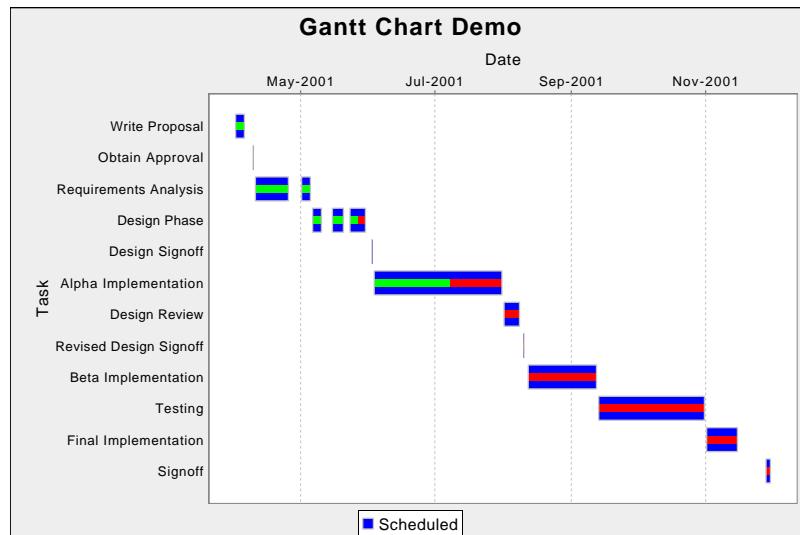


Figure 2.21: A Gantt chart with progress indicators (see [GanttChartDemo2.java](#))

## 2.12 Multiple Axis Charts

JFreeChart has support for charts with multiple axes. Figure 2.22 shows a *price-volume chart* that demonstrates this feature.

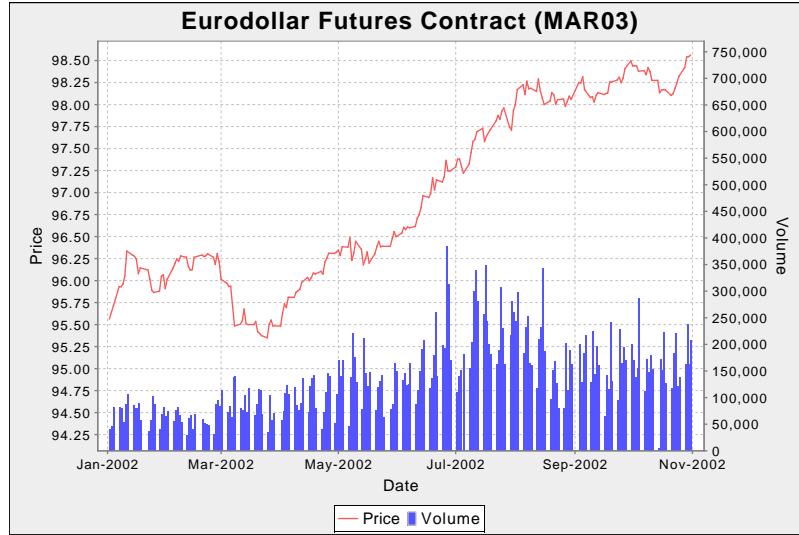


Figure 2.22: A price-volume chart (see `PriceVolumeDemo1.java`)

This feature is supported by the `CategoryPlot` and `XYPlot` classes. Figure 2.23 shows an example with four range axes.

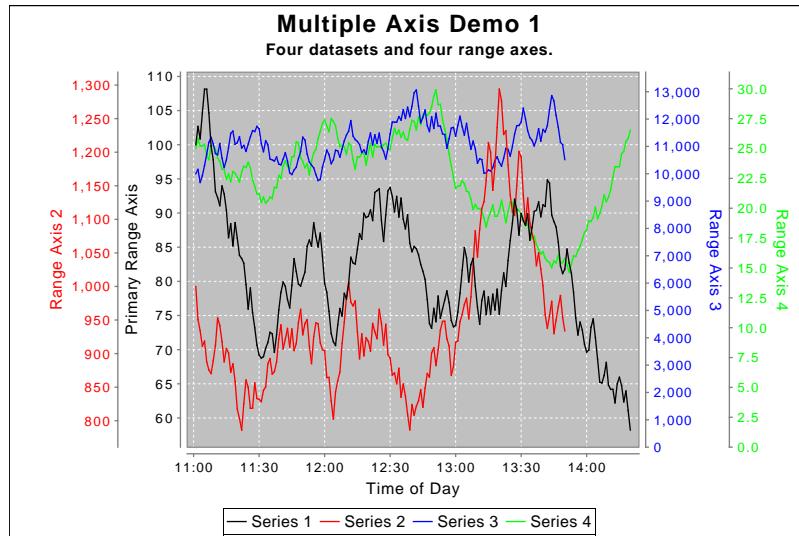


Figure 2.23: A chart with multiple axes (see `MultipleAxisDemo1.java`)

## 2.13 Combined and Overlaid Charts

JFreeChart supports combined and overlaid charts. Figure 2.24 shows a line chart overlaid on top of a bar chart.

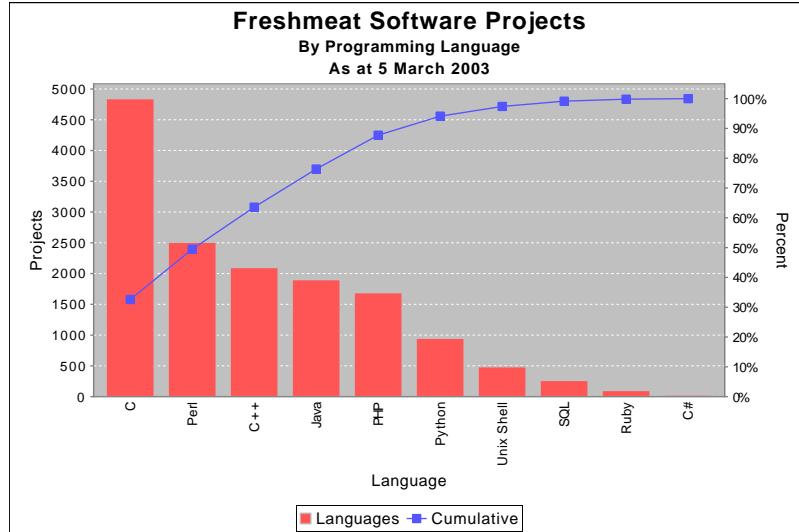


Figure 2.24: An overlaid chart (see `ParetoChartDemo1.java`)

It is possible to combine several charts that share a common domain axis, as shown in figure 2.25.

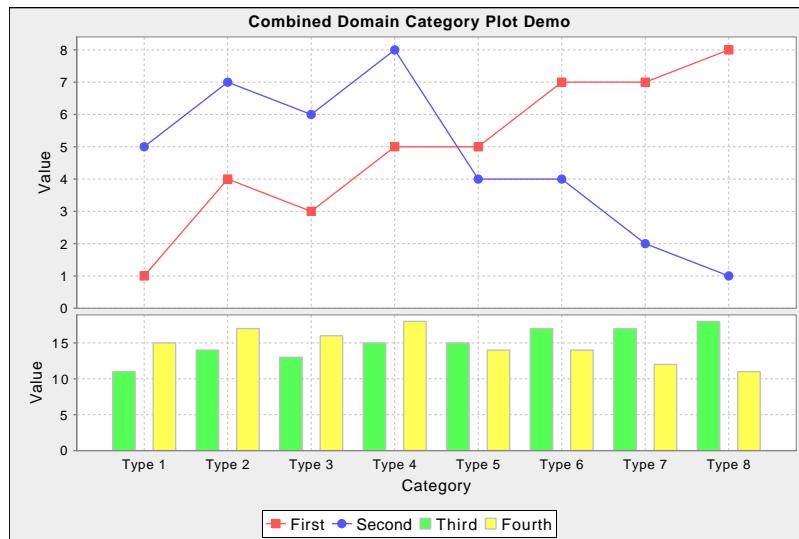


Figure 2.25: A chart with a combined domain (see `CombinedCategoryPlotDemo1.java`)

In a similar way, JFreeChart can combine several charts that share a common range axis, see figure 2.26.

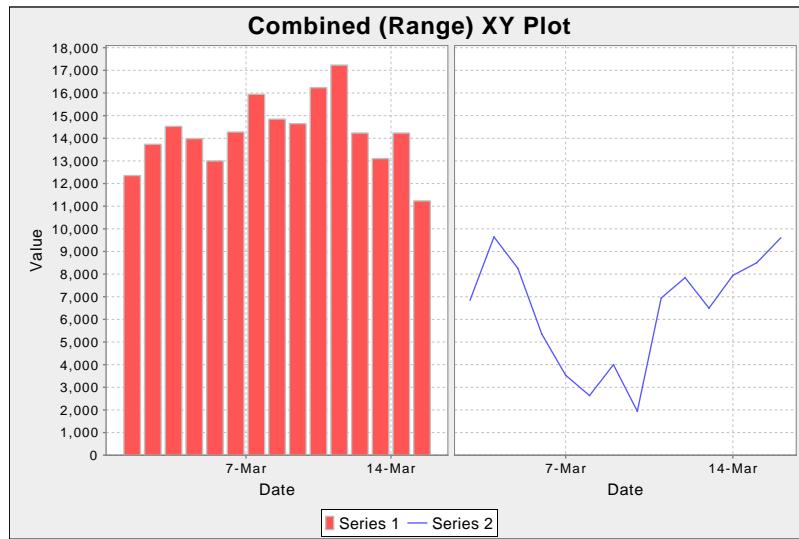


Figure 2.26: A chart with a combined range (see `CombinedXYPlotDemo2.java`)

## 2.14 Future Development

JFreeChart is *free software*,<sup>2</sup> so anyone can extend it and add new features to it. Already, more than 80 developers from around the world have contributed code back to the JFreeChart project. It is likely that many more chart types will be developed in the future as developers modify JFreeChart to meet their requirements. Check the JFreeChart home page regularly for announcements and other updates:

<http://www.jfree.org/jfreechart/>

And if you would like to contribute code to the project, please join in...

---

<sup>2</sup>See <http://www.fsf.org>

# Chapter 3

# Downloading and Installing JFreeChart

## 3.1 Introduction

This section contains instructions for downloading, unpacking, and (optionally) recompiling JFreeChart. Also included are instructions for running the JFreeChart demonstration application, and generating the Javadoc HTML files from the JFreeChart source code.

## 3.2 Download

You can download the latest version of JFreeChart from:

<http://www.jfree.org/jfreechart/download/>

There are two versions of the JFreeChart download:

File:	Description:
jfreechart-1.0.9.tar.gz	JFreeChart for Linux/Unix.
jfreechart-1.0.9.zip	JFreeChart for Windows.

The two files contain the same source code. The main difference is that all the text files in the `zip` download have been recoded to have both carriage return *and* line-feed characters at the end of each line.

JFreeChart uses the JCommon class library (currently version 1.0.12). The JCommon runtime jar file is included in the JFreeChart download, but if you require the source code (recommended) then you should also download JCommon from:

<http://www.jfree.org/jcommon/>

## 3.3 Unpacking the Files

After downloading JFreeChart, you need to unpack the files. You should move the download file to a convenient directory—when you unpack JFreeChart, a new subdirectory (`jfreechart-1.0.9`) will be created in the same location as the `zip` or `tar.gz` archive file.

### 3.3.1 Unpacking on Linux/Unix

To extract the files from the download on Linux/Unix, enter the following command:

```
tar xvzf jfreechart-1.0.9.tar.gz
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-1.0.9`.

### 3.3.2 Unpacking on Windows

To extract the files from the download on Windows, you can use the `jar` utility. Enter the following command:

```
jar -xvf jfreechart-1.0.9.zip
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-1.0.9`.

### 3.3.3 The Files

The top-level directory (`jfreechart-1.0.9`) contains the files and directories listed in the following table:

File/Directory:	Description:
<code>README.txt</code>	Important information - <i>read this first!</i>
<code>NEWS</code>	Project news.
<code>ChangeLog</code>	A detailed log of changes made to JFreeChart.
<code>ant</code>	A directory containing an Ant <code>build.xml</code> script. You can use this script to rebuild JFreeChart from the source code included in the distribution.
<code>checkstyle</code>	A directory containing several Checkstyle property files. These define the coding conventions used in the JFreeChart source code.
<code>experimental</code>	A directory containing source files for classes that are not part of the standard JFreeChart API (yet). We would appreciate feedback on this code. Please note that the API for these classes is subject to change.
<code>lib</code>	A directory containing the JFreeChart jar file, and other libraries used by JFreeChart.
<code>source</code>	A directory containing the source code for JFreeChart.
<code>swt</code>	A directory containing the source code for the experimental SWT code. Please note that the API for these classes is subject to change.
<code>tests</code>	A directory containing the source code for the JFreeChart unit tests.
<code>jfreechart-1.0.9-demo.jar</code>	A runnable jar file containing demo applications.
<code>licence-LGPL.txt</code>	The JFreeChart licence (GNU LGPL).

You should spend some time familiarising yourself with the files included in the download. In particular, you should always read the `README.txt` file.

## 3.4 Running the Demonstration Applications

A demonstration application is included in the distribution that shows a wide range of charts that can be generated with JFreeChart . To run the demo, type the following command:

```
java -jar jfreechart-1.0.9-demo.jar
```

The source code for the demo application is not included in the JFreeChart distribution, but is available to download separately when you purchase the JFreeChart Developer Guide.<sup>1</sup>

---

<sup>1</sup>If you have purchased the guide and you want to download the demo source code, look for the file `jfreechart-1.0.9-demos.zip` on the download page for the JFreeChart Developer Guide.

## 3.5 Configuring JFreeChart for use in IDEs

If, like most developers, you use an integrated development environment (IDE) such as Eclipse or NetBeans for your Java development work, you'll want to configure JFreeChart within that IDE. The procedure for this is IDE-specific—refer to Appendix C for more details.

## 3.6 Compiling the Source

To recompile the JFreeChart classes, you can use the Ant `build.xml` file included in the distribution. Change to the `ant` directory and type:

```
ant compile
```

This will recompile all the necessary source files and recreate the JFreeChart run-time jar file.

To run the script requires that you have Ant 1.5.1 (or later) installed on your system, to find out more about Ant visit:

<http://ant.apache.org/>

It is possible to recompile JFreeChart without using Ant, but there are one or two “gotchas” that you have to take special care to avoid:

- some JFreeChart classes (particularly resource bundles) are not referenced *directly* in the code, and some compilers omit to compile them—this results in runtime errors or problems due to missing class files;
- if you create your own JFreeChart jar file, you need to be sure to include the non-Java files (resource bundle `.properties` files, `gorilla.jpg`, etc.).

In the end, it's simpler to learn Ant and use the script included in the JFreeChart distribution.

## 3.7 Generating the Javadoc Documentation

The JFreeChart source code contains extensive *Javadoc comments*. You can use the `javadoc` tool to generate HTML documentation files directly from the source code.

To generate the documentation, use the `javadoc` target in the Ant `build.xml` script:

```
ant javadoc
```

This will create a `javadoc` directory containing all the Javadoc HTML files, inside the main `jfreechart-1.0.9` directory.

# Chapter 4

## Using JFreeChart

### 4.1 Overview

This section presents a simple introduction to JFreeChart, intended for new users of JFreeChart.

### 4.2 Creating Your First Chart

#### 4.2.1 Overview

Creating charts with JFreeChart is a three step process. You need to:

- create a dataset containing the data to be displayed in the chart;
- create a [JFreeChart](#) object that will be responsible for drawing the chart;
- draw the chart to some output target (often, but not always, a panel on the screen);

To illustrate the process, we describe a sample application ([First.java](#)) that produces the pie chart shown in figure 4.1.

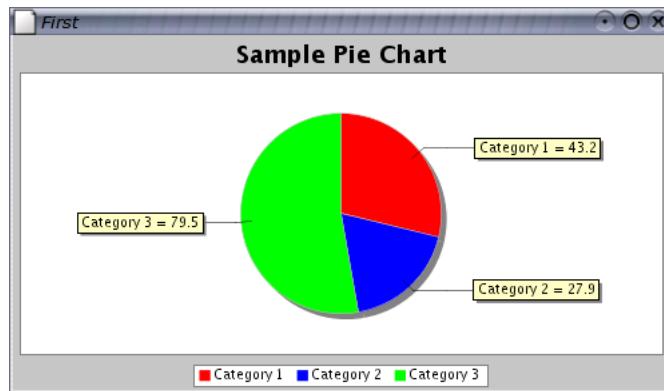


Figure 4.1: A pie chart created using [First.java](#)

Each of the three steps outlined above is described, along with sample code, in the following sections.

#### 4.2.2 The Data

Step one requires us to create a dataset for our chart. This can be done easily using the [DefaultPieDataset](#) class, as follows:

```
// create a dataset...
DefaultPieDataset dataset = new DefaultPieDataset();
dataset.setValue("Category 1", 43.2);
dataset.setValue("Category 2", 27.9);
dataset.setValue("Category 3", 79.5);
```

Note that JFreeChart can create pie charts using data from *any* class that implements the `PieDataset` interface. The `DefaultPieDataset` class (used above) provides a convenient implementation of this interface, but you are free to develop an alternative dataset implementation if you want to.<sup>1</sup>

### 4.2.3 Creating a Pie Chart

Step two concerns how we will present the dataset created in the previous section. We need to create a `JFreeChart` object that can draw a chart using the data from our pie dataset. We will use the `ChartFactory` class, as follows:

```
// create a chart...
JFreeChart chart = ChartFactory.createPieChart(
    "Sample Pie Chart",
    dataset,
    true, // legend?
    true, // tooltips?
    false // URLs?
);
```

Notice how we have passed a reference to the dataset to the factory method. JFreeChart keeps a reference to this dataset so that it can obtain data later on when it is drawing the chart.

The chart that we have created uses default settings for most attributes. There are many ways to customise the appearance of charts created with JFreeChart, but in this example we will just accept the defaults.

### 4.2.4 Displaying the Chart

The final step is to display the chart somewhere. JFreeChart is very flexible about where it draws charts, thanks to its use of the `Graphics2D` class.

For now, let's display the chart in a frame on the screen. The `ChartFrame` class contains the machinery (a `ChartPanel`) required to display charts:

```
// create and display a frame...
ChartFrame frame = new ChartFrame("Test", chart);
frame.pack();
frame.setVisible(true);
```

And that's all there is to it...

### 4.2.5 The Complete Program

Here is the complete program, so that you can see which packages you need to import and the order of the code fragments given in the preceding sections:

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

public class First {

    /**
     * The starting point for the demo.
     *
     * @param args ignored.
     */
    public static void main(String[] args) {
```

<sup>1</sup>This is similar in concept to the way that Swing's `JTable` class obtains data via the `TableModel` interface. In fact, this was the inspiration for using interfaces to define the datasets for JFreeChart.

```
// create a dataset...
DefaultPieDataset dataset = new DefaultPieDataset();
dataset.setValue("Category 1", 43.2);
dataset.setValue("Category 2", 27.9);
dataset.setValue("Category 3", 79.5);

// create a chart...
JFreeChart chart = ChartFactory.createPieChart(
    "Sample Pie Chart",
    dataset,
    true,      // legend?
    true,      // tooltips?
    false     // URLs?
);

// create and display a frame...
ChartFrame frame = new ChartFrame("First", chart);
frame.pack();
frame.setVisible(true);

}

}
```

Hopefully this has convinced you that it is not difficult to create and display charts with JFreeChart. Of course, there is much more to learn...

# Chapter 5

## Pie Charts

### 5.1 Introduction

This chapter provides information about using some of the standard features of the pie charts in JFreeChart, including:

- controlling the color and outline of pie sections;
- handling of `null` and zero values;
- pie section labels (customising the text, altering the space allocated);
- “exploded” sections;
- multiple pie charts.
- displaying charts with a 3D effect;

In addition to this chapter, you should refer to the [PiePlot](#) reference documentation in section [33.27](#).

### 5.2 Creating a Simple Pie Chart

A step-by-step guide to creating a simple pie chart is included in the previous chapter [4](#).

### 5.3 Section Colours

Default fill colours for the pie sections are allocated automatically<sup>1</sup> the first time a plot is rendered. If you don’t like the default colours, you can set them yourself using the `setSectionPaint(Comparable, Paint)` method. For example:

```
PiePlot plot = (PiePlot) chart.getPlot();
plot.setSectionPaint("Section A", new Color(200, 255, 255));
plot.setSectionPaint("Section B", new Color(200, 200, 255));
```

A demo that uses custom colours (`PieChartDemo2.java`) is included in the *JFreeChart demo collection*.

In addition to the per-series section colour attributes, there is also a base or default setting—for more information, refer to the documentation for the [PiePlot](#) class (section [33.27](#)).

---

<sup>1</sup>Inside the `lookupSectionPaint(Comparable, boolean)` method of the [PiePlot](#) class.

## 5.4 Section Outlines

Section outlines are drawn, by default, as a thin grey line around each pie section. The `PiePlot` class provides options to:

- switch off the outlines completely;
- change the outlines for all sections by changing the default values;
- control the outline for particular pie sections independently;

### 5.4.1 Outline Visibility

To switch off the section outlines completely, use the following code:

```
PiePlot plot = (PiePlot) chart.getPlot();
plot.setSectionOutlinesVisible(false);
```

At any time, you can make the outlines visible again using:

```
plot.setSectionOutlinesVisible(true);
```

Calls to this method trigger a `PlotChangeEvent`, which will cause the chart to be repainted immediately if it is displayed in a `ChartPanel`.

### 5.4.2 Outline Appearance

When outlines are visible, you can change the colour and style of the outline for all pie sections (using the base settings) or individual pie sections (using the per series settings).

At the base layer, a default setting is defined—this is used when no higher level settings have been made. You can change the base settings with these methods in the `PiePlot` class:

```
public void setBaseSectionOutlinePaint(Paint paint);
public void setBaseSectionOutlineStroke(Stroke stroke);
```

Sometimes, you may prefer to set the outline paint and stroke on a “per series” basis, perhaps to highlight a particular section in the chart. For this, you can use the series layer settings, defined via these methods:

```
public void setSectionOutlinePaint(Comparable key, Paint paint);
public void setSectionOutlineStroke(Comparable key, Stroke stroke);
```

The first argument for each method is the section key from the dataset. If you set the value for a section to `null`, the base layer setting will be used instead.

## 5.5 Null, Zero and Negative Values

A `PieDataset` can contain `null`, zero or negative values which are awkward or impossible to display in a pie chart. Some special handling is built into the `PiePlot` class for these.

If a zero value is found in the dataset, the `PiePlot` class, by default, will place a label at the position where the pie section would be displayed *if it had a positive value* and will also add an item to the chart’s legend. If you prefer zero values to be ignored, you can set a flag for this, as follows:

```
PiePlot plot = (PiePlot) chart.getPlot();
plot.setIgnoreZeroValues(true);
```

A similar approach is taken for `null` values, which represent a missing or unknown value in the dataset. The default handling is the same as for zero values, and if you prefer `null` values to be ignored, you can set a flag as follows:

```
PiePlot plot = (PiePlot) chart.getPlot();
plot.setIgnoreNullValues(true);
```

There does not seem to be a sensible way to represent negative values in a pie chart, and JFreeChart will always ignore them.

## 5.6 Section and Legend Labels

The text used for the section labels, both on the chart and in the chart's legend, is fully customisable. Default label generators are installed automatically, but if you need to you can change these with the following methods:

```
public void setLabelGenerator(PieSectionLabelGenerator generator);
public void setLegendLabelGenerator(PieSectionLabelGenerator generator);
```

The `StandardPieSectionLabelGenerator` class is typically used as the generator, and provides enough flexibility to handle most custom labelling requirements (but if not, you are free to write your own class that implements the `PieSectionLabelGenerator` interface). The generator works by using Java's `MessageFormat` class to construct labels by substituting values that are derived from the dataset—see table 5.1 for the available substitutions.

Key:	Value:
{0}	The section key as a <code>String</code> .
{1}	The section value.
{2}	The section value as a percentage of the total of all values in the dataset.

Figure 5.1: *StandardPieSectionLabelGenerator substitutions*

By way of example, suppose you have a `PieDataset` containing the following values:

Section Key:	Section Value:
S1	3.0
S2	5.0
S3	null
S4	2.0

Figure 5.2: *A sample dataset*

...then the following format strings would generate the labels shown:

Format String:	Section:	Generated Label:
{0}	0	S1
{0} has value {1}	1	S2 has value 5.0
{0} ({2} percent)	0	S1 (30 percent)
{0} = {1}	2	S3 = null

Figure 5.3: *Format string examples*

The `PieChartDemo2.java` application (included in the *JFreeChart demo collection*) shows a custom label generator in use.

## 5.7 Exploded Sections

The `PiePlot` class supports the display of “exploded” sections, in which a pie section is offset from the centre of the chart to highlight it. For example, the `PieChartDemo2.java` application creates the chart shown in figure 5.6.

The amount by which a section is offset from the chart is specified as a percentage of the radius of the pie chart, for example 0.30 (30 percent) is used in the example.

```
PiePlot plot = (PiePlot) chart.getPlot();
plot.setExplodePercent("Section A", 0.30);
```

To make space for the sections that are offset from the centre of the pie chart, the radius of the main pie is reduced, so a pie chart with exploded sections will appear smaller than a pie chart with no exploded sections.

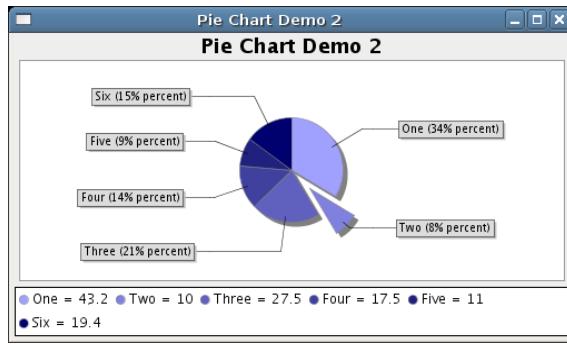


Figure 5.4: A pie chart with an “exploded” section

## 5.8 3D Pie Charts

JFreeChart includes a `PiePlot3D` class that adds a pseudo-3D effect to pie charts—for example, see figure 5.5. `PiePlot3D` is a subclass of `PiePlot`, so you can just substitute it when you create your pie chart. Or if you construct your pie charts using the `ChartFactory` class, it is sufficient to call the `createPieChart3D()` method instead of the `createPieChart()` method.

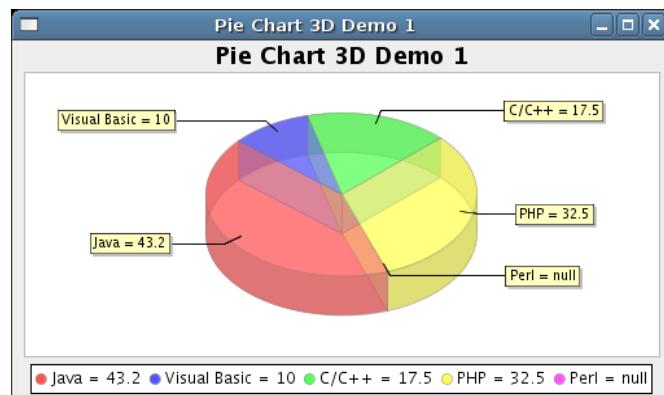


Figure 5.5: A 3D pie chart

There are some limitations with this class:

- exploded sections are not supported;
- it is not possible to set the angle of “rotation” for the 3D effect—if the plot is wider than it is tall, the chart usually looks good, but if the plot is taller than it is wide, the 3D effect is a little distorted.

Some demo applications (`PieChart3DDemo1-3.java`) are included in the *JFreeChart demo collection*.

## 5.9 Multiple Pie Charts

As a convenience, the `MultiplePiePlot` class enables you to create a single chart that displays multiple pie plots using data from a `CategoryDataset`. An example is shown in figure 5.6.

The individual pie charts are created by “rubber stamping” a single pie chart multiple times. For each rendering of the pie chart, a new `PieDataset` is extracted from the next row (or column) of the `CategoryDataset`.

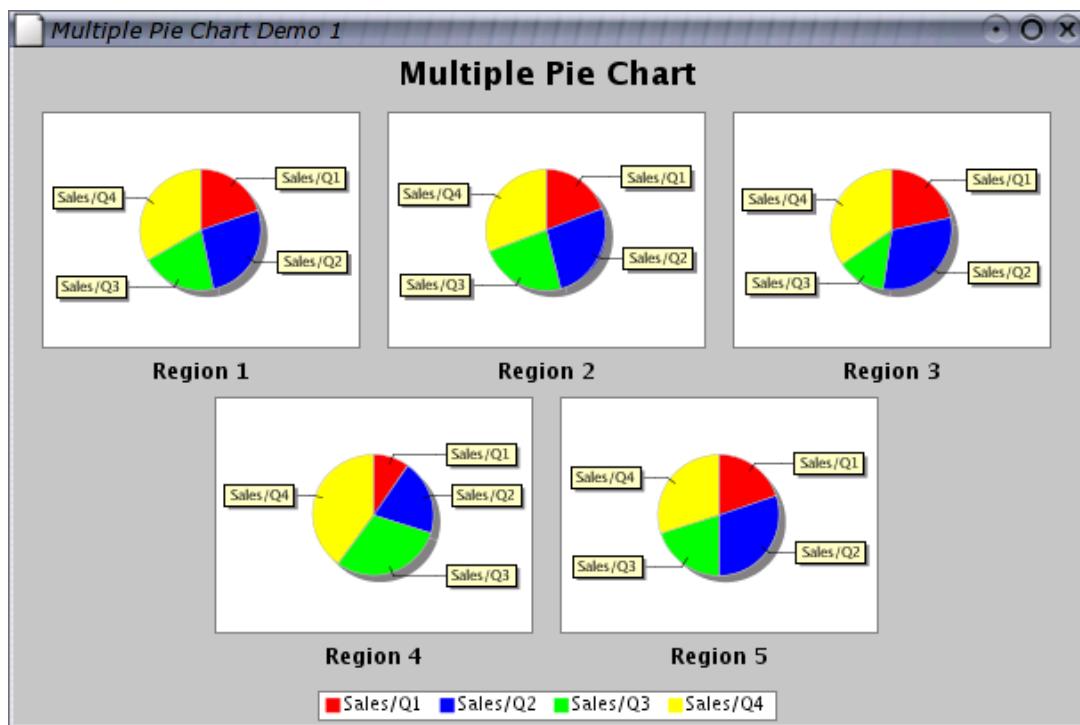


Figure 5.6: A chart using *MultiplePiePlot*

A number of demos (`MultiplePieChartDemo1-4.java`) are included in the *JFreeChart demo collection*.

# Chapter 6

## Bar Charts

### 6.1 Introduction

This chapter provides an introduction to creating bar charts with JFreeChart. We begin with a very simple bar chart, then go on to describe some of the many options that JFreeChart provides for customising the charts. After covering the standard bar chart and its options, we'll move on to some more complex bar chart variants:

- stacked bar charts;
- bar charts for time series data;
- histograms.

By the end of this chapter, you should have a good overview of the features that JFreeChart supports for bar chart creation.

### 6.2 A Bar Chart

#### 6.2.1 Overview

Bar charts are used to provide a visual representation of tabular data. For example, consider the following table, which contains a simple set of data in two rows and three columns:

	Column 1:	Column 2:	Column 3:
Row 1:	1.0	5.0	3.0
Row 2:	2.0	3.0	2.0

Figure 6.1: Sample data

In JFreeChart, this table is referred to as a *dataset*, each column heading is a *category*, and each row in the table is a *series*. Each row heading is a *series name* (or *series key*). A bar chart that presents this data is shown in figure 6.2.

You can see in the sample chart that JFreeChart groups the items from each column (category) together, and uses colours to highlight the data from each row (series). The chart's legend provides the link between the bar colours and the series name/key.

#### 6.2.2 Creating a Dataset

The first step in creating a bar chart is to create an appropriate dataset. The set of methods that JFreeChart uses to access the tabular data for a bar chart is defined by the [CategoryDataset](#) interface.

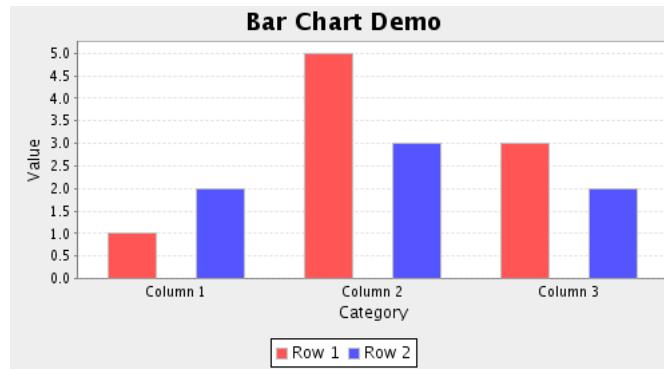


Figure 6.2: A bar chart (see `BarExample1.java`)

This interface defines a *read-only* interface to the dataset, because that is all that JFreeChart requires to draw charts. A dataset can, *but is not required to*, provide methods to update the dataset.

A convenient class that implements this interface is the `DefaultCategoryDataset` class. Here is how you might create a dataset for the values given in table 6.1:

```
DefaultCategoryDataset dataset = new DefaultCategoryDataset();
dataset.addValue(1.0, "Row 1", "Column 1");
dataset.addValue(5.0, "Row 1", "Column 2");
dataset.addValue(3.0, "Row 1", "Column 3");
dataset.addValue(2.0, "Row 2", "Column 1");
dataset.addValue(3.0, "Row 2", "Column 2");
dataset.addValue(2.0, "Row 2", "Column 3");
```

### 6.2.3 Creating a Bar Chart

The next step is to create a `JFreeChart` instance that draws a bar chart for this example dataset. Taking a short-cut, we use the `ChartFactory` class to create the `JFreeChart` instance:

```
JFreeChart chart = ChartFactory.createBarChart(
    "Bar Chart Demo", // chart title
    "Category", // domain axis label
    "Value", // range axis label
    dataset, // data
    PlotOrientation.VERTICAL, // orientation
    true, // include legend
    true, // tooltips?
    false // URLs?
);
```

Most of the arguments to the `createBarChart()` method are obvious, but a few of them demand further explanation:

- the plot orientation can be either horizontal or vertical (for bar charts, this corresponds to the way the bars are drawn, horizontally or vertically);
- the tooltips flag controls whether or not a tool tip generator is added to the chart—in this example, we've set this flag to `true` so that we'll see tool tips when we display the chart in a Swing application;

- the URLs flag is only relevant when creating drill-down reports using HTML image maps, so we set it to `cffalse` here.

After we've completed this first bar chart example, we'll come back and take a closer look at what the `ChartFactory` class is doing "behind the scenes" here.

#### 6.2.4 Displaying the Chart

To complete our first bar chart example, we pass the `JFreeChart` instance to a `ChartPanel` and display it in a simple Swing application. The full source code for this example is listed here:

```
/*
 * -----
 * BarExample1.java
 * -----
 * (C) Copyright 2006, by Object Refinery Limited.
 *
 */

package tutorial;

import java.awt.Dimension;

/**
 * A simple demonstration application showing how to create a bar chart.
 */
public class BarExample1 extends ApplicationFrame {

    /**
     * Creates a new demo instance.
     *
     * @param title  the frame title.
     */
    public BarExample1(String title) {
        super(title);
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.addValue(1.0, "Row 1", "Column 1");
        dataset.addValue(5.0, "Row 1", "Column 2");
        dataset.addValue(3.0, "Row 1", "Column 3");
        dataset.addValue(2.0, "Row 2", "Column 1");
        dataset.addValue(3.0, "Row 2", "Column 2");
        dataset.addValue(2.0, "Row 2", "Column 3");
        JFreeChart chart = ChartFactory.createBarChart(
            "Bar Chart Demo",           // chart title
            "Category",                 // domain axis label
            "Value",                    // range axis label
            dataset,                    // data
            PlotOrientation.VERTICAL,   // orientation
            true,                       // include legend
            true,                       // tooltips?
            false                       // URLs?
        );
        ChartPanel chartPanel = new ChartPanel(chart, false);
        chartPanel.setPreferredSize(new Dimension(500, 270));
        setContentPane(chartPanel);
    }

    /**
     * Starting point for the demonstration application.
     *
     * @param args  ignored.
     */
    public static void main(String[] args) {
        BarExample1 demo = new BarExample1("Bar Demo 1");
        demo.pack();
        RefineryUtilities.centerFrameOnScreen(demo);
        demo.setVisible(true);
    }
}
```

```
}
```

If you compile and run this example, you should see a frame containing the chart in figure 6.2.

### 6.3 The ChartFactory Class

In our first example, the `ChartFactory` class is used to piece together a `JFreeChart` instance that renders a bar chart. Here we take a closer look at how this is done, so we can see a little more of the underlying structure of our bar chart. Understanding this structure is key to being able to customise the appearance of the chart.

Here are the important parts of the code from the `createBarChart()` method in the `ChartFactory` class:

```

1 CategoryAxis categoryAxis = new CategoryAxis(categoryAxisLabel);
2 ValueAxis valueAxis = new NumberAxis(valueAxisLabel);
3 BarRenderer renderer = new BarRenderer();
[snip...]
4 CategoryPlot plot = new CategoryPlot(dataset, categoryAxis, valueAxis,
    renderer);
5 plot.setOrientation(orientation);
6 JFreeChart chart = new JFreeChart(title, JFreeChart.DEFAULT_TITLE_FONT,
    plot, legend);
```

Here's what this code is doing:

- Our bar chart has two axes, one that displays categories from the dataset (a `CategoryAxis`), and another that provides the numerical scale against which the data values are plotted (a `NumberAxis`). You can see these axes being constructed in lines 1 and 2 above, using the axis labels that we passed to the `createBarChart()` method.
- At line 3, a `BarRenderer` is created—this is the class that is used to draw the bar for each data item. The renderer handles most of the drawing work, and you'll see later that you can substitute another type of renderer to change the overall appearance of the chart.
- The dataset, axes and renderer are all managed by a `CategoryPlot`, which coordinates most of the interaction between these components. When you customise charts, you'll often need to get a reference to the chart's plot, which in turn can give you access to the axes, renderer and dataset. At line 4, the plot is created, and the other components are assigned to it.
- Finally, the plot is wrapped in a `JFreeChart` instance, with the specified title. The `JFreeChart` class provides the top-level access to a chart, but most of the “guts” of a chart is defined at the plot level (or in the objects managed by the plot, such as the axes, dataset(s) and renderer(s)).

Armed with this knowledge of the internal structure of our chart, in the following sections, we'll slowly customise our bar chart.

### 6.4 Simple Chart Customisation

Some simple modifications to the appearance of a bar chart can be made by calling methods in the `JFreeChart` and `CategoryPlot` classes. For example, to change the background colours for the chart and plot:

```
chart.setBackgroundPaint(Color.white);
CategoryPlot plot = (CategoryPlot) chart.getPlot();
plot.setBackgroundPaint(Color.lightGray);
plot.setRangeGridlinePaint(Color.white);
```

This code fragment (from `BarExample2.java`) changes the background colour for the chart, then obtains a reference to the chart's plot and modifies it as well—see figure 6.3.

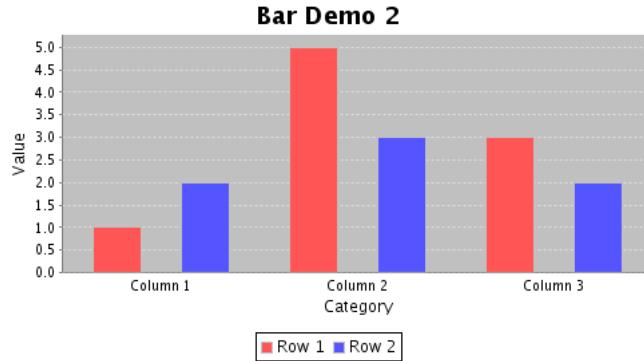


Figure 6.3: A bar chart (see `BarExample2.java`)

A cast of the plot reference (to `CategoryPlot`) is required—it is safe to make this cast, because we know that a `CategoryPlot` is used for this chart type. JFreeChart uses other plot types (`PiePlot`, `XYPlot`, and so on) for different kinds of charts. You almost always need to cast the plot reference to one of these types, because the base class (`Plot`) only defines a few common attributes and methods. As you become more familiar with JFreeChart, you'll learn which `Plot` subclass is used for each type of chart.

In our example, we also use the plot reference to change the colour of the grid lines for the range axis. Take a look through the API documentation for the `CategoryPlot` class, to see what else you could modify here.

## 6.5 Customising the Renderer

Recall from section 6.3 that the `CategoryPlot` manages a renderer which, in the case of a regular bar chart, is an instance of `BarRenderer`. If we obtain a reference to this renderer, a large number of customisation options become available.

### 6.5.1 Bar Colours

To change the colours used for each series in the chart:

```
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setSeriesPaint(0, Color.gray);
renderer.setSeriesPaint(1, Color.orange);
renderer.setDrawBarOutline(false);
```

This results in the chart shown in figure 6.4. Note that the `setSeriesPaint()` method is defined in the `AbstractRenderer` base class—you can use this for all types of renderer.

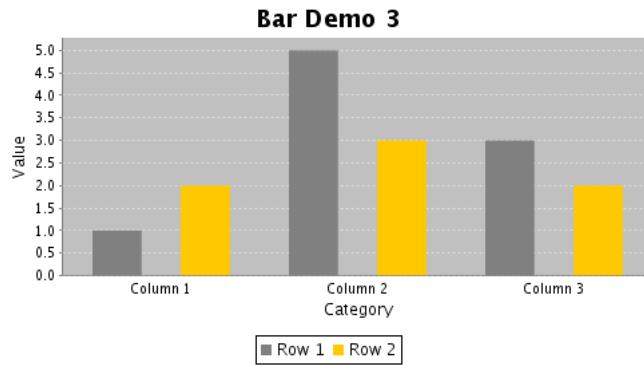


Figure 6.4: A bar chart (see `BarExample3.java`)

### 6.5.2 Bar Spacing Within Categories

Among other things, the renderer controls the spacing of bars within each category.<sup>1</sup> So we could remove all the space between bars in the same category, as follows:

```
renderer.setItemMargin(0.0);
```

This results in the chart shown in figure 6.5. Notice how the bars have grown a little wider—this is because JFreeChart is now allocating less of the overall space to provide gaps between the bars, so the bars themselves resize a little bigger.

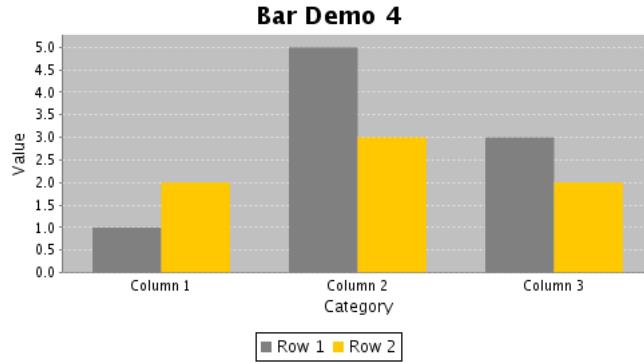


Figure 6.5: A bar chart (see `BarExample4.java`)

---

<sup>1</sup>The spacing between categories is controlled by the `CategoryAxis`. That will be covered later.

# Chapter 7

## Line Charts

### 7.1 Introduction

This section describes the *line charts* that can be created with JFreeChart. It is possible to create line charts using data from either the [CategoryDataset](#) interface or the [XYDataset](#) interface.

### 7.2 A Line Chart Based On A Category Dataset

#### 7.2.1 Overview

A *line chart* based on a [CategoryDataset](#) simply connects each *(category, value)* data item using straight lines. This section presents a sample application that generates the following chart shown in figure 7.1.

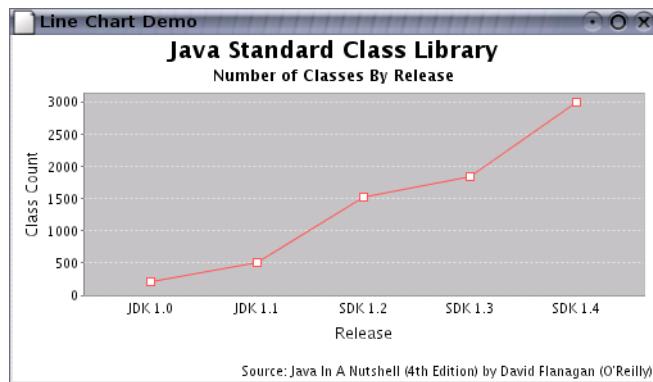


Figure 7.1: A sample line chart

The full source code for this demo (`LineChartDemo1.java`) is available for download with the JFreeChart Developer Guide.

#### 7.2.2 The Dataset

The first step in generating the chart is, as always, to create a dataset. In the example, the [DefaultCategoryDataset](#) class is used:

```
DefaultCategoryDataset dataset = new DefaultCategoryDataset();
dataset.addValue(212, "Classes", "JDK 1.0");
dataset.addValue(504, "Classes", "JDK 1.1");
dataset.addValue(1520, "Classes", "SDK 1.2");
```

```
dataset.addValue(1842, "Classes", "SDK 1.3");
dataset.addValue(2991, "Classes", "SDK 1.4");
```

Note that you can use *any* implementation of the `CategoryDataset` interface as your dataset.

### 7.2.3 Constructing the Chart

The `createLineChart()` method in the `ChartFactory` class provides a convenient way to create the chart. Here is the code:

```
// create the chart...
JFreeChart chart = ChartFactory.createLineChart(
    "Java Standard Class Library", // chart title
    "Release", // domain axis label
    "Class Count", // range axis label
    dataset, // data
    PlotOrientation.VERTICAL, // orientation
    false, // include legend
    true, // tooltips
    false // urls
);
```

This method constructs a `JFreeChart` object with a title, no legend, and plot with appropriate axes, renderer and tooltip generator. The `dataset` is the one created in the previous section.

### 7.2.4 Customising the Chart

The chart will be initialised using default settings for most attributes. You are, of course, free to modify any of the settings to change the appearance of your chart. In this example, we customise the chart in the following ways:

- two subtitles are added to the chart;
- the chart background color is set to white;
- the plot background color is set to light gray;
- the gridline color is changed to white;
- the range axis is modified to display integer values only;
- the renderer is modified to fill shapes with white.

The first subtitle is added at the default position (below the main title):

```
chart.addSubtitle(new TextTitle("Number of Classes By Release"));
```

The next subtitle takes a little extra code, to change the font, place it at the bottom of the chart, and align it to the right side:

```
TextTitle source = new TextTitle(
    "Source: Java In A Nutshell (4th Edition) "
    + "by David Flanagan (O'Reilly)"
);
source.setFont(new Font("SansSerif", Font.PLAIN, 10));
source.setPosition(RectangleEdge.BOTTOM);
source.setHorizontalTextPosition(HorizontalAlignment.RIGHT);
chart.addSubtitle(source);
```

Changing the chart's background color is simple, because this is an attribute maintained by the `JFreeChart` class:

```
chart.setBackgroundPaint(Color.white);
```

To change other attributes, we first need to obtain a reference to the `CategoryPlot` object used by the chart:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
```

To set the background color for the plot, and change the gridline color:

```
plot.setBackgroundPaint(Color.lightGray);
plot.setRangeGridlinePaint(Color.white);
```

The plot is responsible for drawing the data and axes on the chart. Some of this work is delegated to a *renderer*, which you can access via the `getRenderer()` method. The renderer maintains most of the attributes that relate to the appearance of the data items within the chart.

```
LineAndShapeRenderer renderer = (LineAndShapeRenderer) plot.getRenderer(); renderer.setShapesVisible(true);
renderer.setDrawOutlines(true); renderer.setFillPaint(true);
```

The plot also manages the chart's axes. In the example, the range axis is modified so that it only displays integer values for the tick labels:

```
// change the auto tick unit selection to integer units only...
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
```

There are many other ways to customise the chart. Please refer to the reference section of this document, the API documentation and the source code for details of the methods available.

### 7.2.5 The Complete Program

The code for the demonstration application is presented in full, complete with the import statements. The source code is available for download from the same location as the JFreeChart Developer Guide.

```
/*
 * LineChartDemo1.java
 *
 * (C) Copyright 2002-2005, by Object Refinery Limited.
 *
 */

package demo;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.LineAndShapeRenderer;
import org.jfree.chart.title.TextTitle;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.HorizontalAlignment;
import org.jfree.ui.RectangleEdge;
import org.jfree.ui.RefineryUtilities;

/**
 * A simple demonstration application showing how to create a line chart using
 * data from a {@link CategoryDataset}.
 */
public class LineChartDemo1 extends ApplicationFrame {

    /**
     * Creates a new demo.
     *
     * @param title the frame title.
     */
    public LineChartDemo1(String title) {
        super(title);
```

```

CategoryDataset dataset = createDataset();
JFreeChart chart = createChart(dataset);
ChartPanel chartPanel = new ChartPanel(chart);
chartPanel.setPreferredSize(new Dimension(500, 270));
setContentPane(chartPanel);
}

/**
 * Creates a sample dataset.
 *
 * @return The dataset.
 */
private static CategoryDataset createDataset() {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    dataset.addValue(212, "Classes", "JDK 1.0");
    dataset.addValue(504, "Classes", "JDK 1.1");
    dataset.addValue(1520, "Classes", "SDK 1.2");
    dataset.addValue(1842, "Classes", "SDK 1.3");
    dataset.addValue(2991, "Classes", "SDK 1.4");
    return dataset;
}

/**
 * Creates a sample chart.
 *
 * @param dataset a dataset.
 *
 * @return The chart.
 */
private static JFreeChart createChart(CategoryDataset dataset) {

    // create the chart...
    JFreeChart chart = ChartFactory.createLineChart(
        "Java Standard Class Library",           // chart title
        "Release",                             // domain axis label
        "Class Count",                         // range axis label
        dataset,                               // data
        PlotOrientation.VERTICAL,               // orientation
        false,                                 // include legend
        true,                                  // tooltips
        false                                  // urls
    );

    chart.addSubtitle(new TextTitle("Number of Classes By Release"));
    TextTitle source = new TextTitle(
        "Source: Java In A Nutshell (4th Edition) "
        + "by David Flanagan (O'Reilly)"
    );
    source.setFont(new Font("SansSerif", Font.PLAIN, 10));
    source.setPosition(RectangleEdge.BOTTOM);
    source.setHorizontalAlignment(HorizontalAlignment.RIGHT);
    chart.addSubtitle(source);

    chart.setBackgroundPaint(Color.white);

    CategoryPlot plot = (CategoryPlot) chart.getPlot();
    plot.setBackgroundPaint(Color.lightGray);
    plot.setRangeGridlinePaint(Color.white);

    // customise the range axis...
    NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
    rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

    // customise the renderer...
    LineAndShapeRenderer renderer
        = (LineAndShapeRenderer) plot.getRenderer();
    renderer.setShapesVisible(true);
    renderer.setDrawOutlines(true);
    renderer.setUseFillPaint(true);
    renderer.setFillPaint(Color.white);

    return chart;
}

/**
 * Creates a panel for the demo (used by SuperDemo.java).
 *
 * @return A panel.
 */
public static JPanel createDemoPanel() {
    JFreeChart chart = createChart(createDataset());
}

```

```
        return new ChartPanel(chart);
    }

    /**
     * Starting point for the demonstration application.
     *
     * @param args  ignored.
     */
    public static void main(String[] args) {
        LineChartDemo1 demo = new LineChartDemo1("Line Chart Demo");
        demo.pack();
        RefineryUtilities.centerFrameOnScreen(demo);
        demo.setVisible(true);
    }
}
```

## 7.3 A Line Chart Based On An XYDataset

### 7.3.1 Overview

A *line chart* based on an `XYDataset` connects each  $(x, y)$  point with a straight line. This section presents a sample application that generates the chart shown in figure 7.2.

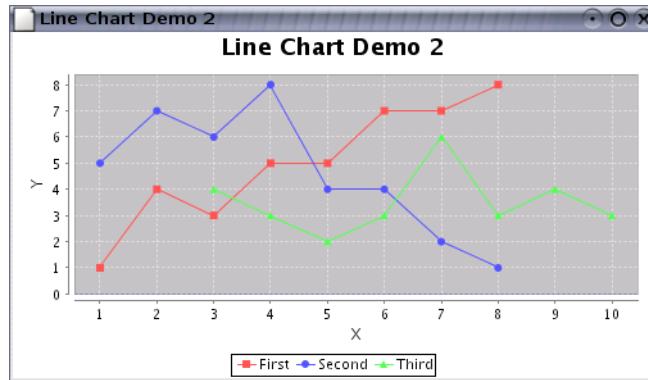


Figure 7.2: A sample line chart using an `XYPlot`

The complete source code (`LineChartDemo2.java`) is available to download with the JFreeChart Developer Guide.

### 7.3.2 The Dataset

For this chart, an `XYSeriesCollection` is used as the dataset (you can use any implementation of the `XYDataset` interface). For the purposes of the self-contained demo, we create this dataset in code, as follows:

```

XYSeries series1 = new XYSeries("First");
series1.add(1.0, 1.0);
series1.add(2.0, 4.0);
series1.add(3.0, 3.0);
series1.add(4.0, 5.0);
series1.add(5.0, 5.0);
series1.add(6.0, 7.0);
series1.add(7.0, 7.0);
series1.add(8.0, 8.0);

XYSeries series2 = new XYSeries("Second");
series2.add(1.0, 5.0);
series2.add(2.0, 7.0);
series2.add(3.0, 6.0);
series2.add(4.0, 8.0);
series2.add(5.0, 4.0);
series2.add(6.0, 4.0);
series2.add(7.0, 2.0);
series2.add(8.0, 1.0);

XYSeries series3 = new XYSeries("Third");
series3.add(3.0, 4.0);
series3.add(4.0, 3.0);
series3.add(5.0, 2.0);
series3.add(6.0, 3.0);
series3.add(7.0, 6.0);
series3.add(8.0, 3.0);
series3.add(9.0, 4.0);
series3.add(10.0, 3.0);

XYSeriesCollection dataset = new XYSeriesCollection();
dataset.addSeries(series1);
dataset.addSeries(series2);
dataset.addSeries(series3);

return dataset;

```

Notice how each series has x-values (not just y-values) that are independent from the other series. The dataset will also accept `null` in place of a y-value. When a `null` value is encountered, no connecting line is drawn, resulting in a discontinuous line for the series.

### 7.3.3 Constructing the Chart

The `createXYLineChart()` method in the `ChartFactory` class provides a convenient way to create the chart:

```
JFreeChart chart = ChartFactory.createXYLineChart(
    "Line Chart Demo 2",           // chart title
    "X",                          // x axis label
    "Y",                          // y axis label
    dataset,                      // data
    PlotOrientation.VERTICAL,
    true,                         // include legend
    true,                         // tooltips
    false                         // urls
);
```

This method constructs a `JFreeChart` object with a title, legend and plot with appropriate axes and renderer. The `dataset` is the one created in the previous section. The chart is created with a legend, and tooltips are enabled (URLs are disabled—these are only used in the creation of HTML image maps).

### 7.3.4 Customising the Chart

The chart will be initialised using default settings for most attributes. You are, of course, free to modify any of the settings to change the appearance of your chart. In this example, several attributes are modified:

- the chart background color;
- the plot background color;
- the axis offsets;
- the color of the domain and range gridlines;
- the renderer is modified to draw shapes as well as lines;
- the tick unit collection for the range axis, so that the tick values always display integer values;

Changing the chart's background color is simple:

```
// set the background color for the chart...
chart.setBackgroundPaint(Color.white);
```

Changing the plot background color, the axis offsets, and the color of the gridlines, requires a reference to the plot. The cast to `XYPlot` is required so that we can access methods specific to this type of plot:

```
// get a reference to the plot for further customisation...
XYPlot plot = (XYPlot) chart.getPlot();
plot.setBackgroundPaint(Color.lightGray);
plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 5.0));
plot.setDomainGridlinePaint(Color.white);
plot.setRangeGridlinePaint(Color.white);
```

The renderer is modified to display filled shapes in addition to the default lines:

```
XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.getRenderer();
renderer.setShapesVisible(true);
renderer.setShapesFilled(true);
```

The final modification is a change to the range axis. We change the default collection of tick units (which allow fractional values) to an integer-only collection:

```
// change the auto tick unit selection to integer units only...
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
```

Refer to the source code, Javadoc API documentation or elsewhere in this document for details of the other customisations that you can make to an [XYPlot](#).

### 7.3.5 The Complete Program

The code for the demonstration application is presented here in full, complete with the import statements:

```
/*
 * LineChartDemo2.java
 *
 * (C) Copyright 2002-2005, by Object Refinery Limited.
 *
 */

package demo;

import java.awt.Color;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RectangleInsets;
import org.jfree.ui.RefineryUtilities;

/**
 * A simple demonstration application showing how to create a line chart using
 * data from an {@link XYDataset}.
 * <p>
 * IMPORTANT NOTE: THIS DEMO IS DOCUMENTED IN THE JFREECHART DEVELOPER GUIDE.
 * DO NOT MAKE CHANGES WITHOUT UPDATING THE GUIDE ALSO!!
 */
public class LineChartDemo2 extends ApplicationFrame {

    /**
     * Creates a new demo.
     *
     * @param title the frame title.
     */
    public LineChartDemo2(String title) {

        super(title);
        XYDataset dataset = createDataset();
        JFreeChart chart = createChart(dataset);
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        setContentPane(chartPanel);

    }

    /**
     * Creates a sample dataset.
     *
     * @return a sample dataset.
     */
    private static XYDataset createDataset() {

        XYSeries series1 = new XYSeries("First");
        series1.add(1.0, 1.0);
        series1.add(2.0, 4.0);

    }
}
```

```

series1.add(3.0, 3.0);
series1.add(4.0, 5.0);
series1.add(5.0, 5.0);
series1.add(6.0, 7.0);
series1.add(7.0, 7.0);
series1.add(8.0, 8.0);

XYSeries series2 = new XYSeries("Second");
series2.add(1.0, 5.0);
series2.add(2.0, 7.0);
series2.add(3.0, 6.0);
series2.add(4.0, 8.0);
series2.add(5.0, 4.0);
series2.add(6.0, 4.0);
series2.add(7.0, 2.0);
series2.add(8.0, 1.0);

XYSeries series3 = new XYSeries("Third");
series3.add(3.0, 4.0);
series3.add(4.0, 3.0);
series3.add(5.0, 2.0);
series3.add(6.0, 3.0);
series3.add(7.0, 6.0);
series3.add(8.0, 3.0);
series3.add(9.0, 4.0);
series3.add(10.0, 3.0);

XYSeriesCollection dataset = new XYSeriesCollection();
dataset.addSeries(series1);
dataset.addSeries(series2);
dataset.addSeries(series3);

return dataset;
}

/**
 * Creates a chart.
 *
 * @param dataset the data for the chart.
 *
 * @return a chart.
 */
private static JFreeChart createChart(XYDataset dataset) {

    // create the chart...
    JFreeChart chart = ChartFactory.createXYLineChart(
        "Line Chart Demo 2",           // chart title
        "X",                          // x axis label
        "Y",                          // y axis label
        dataset,                      // data
        PlotOrientation.VERTICAL,
        true,                         // include legend
        true,                         // tooltips
        false                         // urls
    );

    // NOW DO SOME OPTIONAL CUSTOMISATION OF THE CHART...
    chart.setBackgroundPaint(Color.white);

    // get a reference to the plot for further customisation...
    XYPlot plot = (XYPlot) chart.getPlot();
    plot.setBackgroundPaint(Color.lightGray);
    plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 5.0));
    plot.setDomainGridlinePaint(Color.white);
    plot.setRangeGridlinePaint(Color.white);

    XYLineAndShapeRenderer renderer
        = (XYLineAndShapeRenderer) plot.getRenderer();
    renderer.setShapesVisible(true);
    renderer.setShapesFilled(true);

    // change the auto tick unit selection to integer units only...
    NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
    rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
    // OPTIONAL CUSTOMISATION COMPLETED.

    return chart;
}

```

```
/**  
 * Creates a panel for the demo (used by SuperDemo.java).  
 *  
 * @return A panel.  
 */  
public static JPanel createDemoPanel() {  
    JFreeChart chart = createChart(createDataset());  
    return new ChartPanel(chart);  
}  
  
/**  
 * Starting point for the demonstration application.  
 *  
 * @param args  ignored.  
 */  
public static void main(String[] args) {  
  
    LineChartDemo2 demo = new LineChartDemo2("Line Chart Demo 2");  
    demo.pack();  
    RefineryUtilities.centerFrameOnScreen(demo);  
    demo.setVisible(true);  
}  
}
```

# Chapter 8

## Time Series Charts

### 8.1 Introduction

*Time series charts* are very similar to line charts, except that the values on the domain axis are dates rather than numbers. This section describes how to create time series charts with JFreeChart.

### 8.2 Time Series Charts

#### 8.2.1 Overview

A *time series chart* is really just a *line chart* using data obtained via the `XYDataset` interface (see the example in the previous section). The difference is that the x-values are displayed as dates on the domain axis. This section presents a sample application that generates the chart shown in figure 8.1.



Figure 8.1: A time series chart

The complete source code (`TimeSeriesDemo1.java`) for this example is available for download with the JFreeChart Developer Guide.

#### 8.2.2 Dates or Numbers?

Time series charts are created using data from an `XYDataset`. This interface doesn't have any methods that return dates, so how does JFreeChart create time series charts?

The x-values returned by the dataset are `double` primitives, but the values are interpreted in a special way—they are assumed to represent the number of milliseconds since midnight, 1 January

1970 (the encoding used by the `java.util.Date` class).

A special axis class (`DateAxis`) converts from milliseconds to dates and back again as necessary, allowing the axis to display tick labels formatted as dates.

### 8.2.3 The Dataset

For the demo chart, a `TimeSeriesCollection` is used as the dataset (you can use any implementation of the `XYDataset` interface):

```
TimeSeries s1 = new TimeSeries("L&G European Index Trust", Month.class);
s1.add(new Month(2, 2001), 181.8);
s1.add(new Month(3, 2001), 167.3);
s1.add(new Month(4, 2001), 153.8);
s1.add(new Month(5, 2001), 167.6);
s1.add(new Month(6, 2001), 158.8);
s1.add(new Month(7, 2001), 148.3);
s1.add(new Month(8, 2001), 153.9);
s1.add(new Month(9, 2001), 142.7);
s1.add(new Month(10, 2001), 123.2);
s1.add(new Month(11, 2001), 131.8);
s1.add(new Month(12, 2001), 139.6);
s1.add(new Month(1, 2002), 142.9);
s1.add(new Month(2, 2002), 138.7);
s1.add(new Month(3, 2002), 137.3);
s1.add(new Month(4, 2002), 143.9);
s1.add(new Month(5, 2002), 139.8);
s1.add(new Month(6, 2002), 137.0);
s1.add(new Month(7, 2002), 132.8);

TimeSeries s2 = new TimeSeries("L&G UK Index Trust", Month.class);
s2.add(new Month(2, 2001), 129.6);
s2.add(new Month(3, 2001), 123.2);
s2.add(new Month(4, 2001), 117.2);
s2.add(new Month(5, 2001), 124.1);
s2.add(new Month(6, 2001), 122.6);
s2.add(new Month(7, 2001), 119.2);
s2.add(new Month(8, 2001), 116.5);
s2.add(new Month(9, 2001), 112.7);
s2.add(new Month(10, 2001), 101.5);
s2.add(new Month(11, 2001), 106.1);
s2.add(new Month(12, 2001), 110.3);
s2.add(new Month(1, 2002), 111.7);
s2.add(new Month(2, 2002), 111.0);
s2.add(new Month(3, 2002), 109.6);
s2.add(new Month(4, 2002), 113.2);
s2.add(new Month(5, 2002), 111.6);
s2.add(new Month(6, 2002), 108.8);
s2.add(new Month(7, 2002), 101.6);

TimeSeriesCollection dataset = new TimeSeriesCollection();
dataset.addSeries(s1);
dataset.addSeries(s2);
```

In the example, the series contain monthly data. However, the `TimeSeries` class can be used to represent values observed at other intervals (annual, daily, hourly etc).

### 8.2.4 Constructing the Chart

The `createTimeSeriesChart()` method in the `ChartFactory` class provides a convenient way to create the chart:

```
JFreeChart chart = ChartFactory.createTimeSeriesChart(
    "Legal & General Unit Trust Prices", // title
    "Date", // x-axis label
    "Price Per Unit", // y-axis label
    dataset, // data
    true, // create legend?
    true, // generate tooltips?
    false // generate URLs?
);
```

This method constructs a `JFreeChart` object with a title, legend and plot with appropriate axes and renderer. The `dataset` is the one created in the previous section.

### 8.2.5 Customising the Chart

The chart will be initialised using default settings for most attributes. You are, of course, free to modify any of the settings to change the appearance of your chart. In this example, several attributes are modified:

- the renderer is changed to display series shapes at each data point, in addition to the lines between data points;
- a date format override is set for the domain axis;

Modifying the renderer requires a couple of steps to obtain a reference to the renderer and then cast it to a `XYLineAndShapeRenderer`:

```
XYItemRenderer r = plot.getRenderer();
if (r instanceof XYLineAndShapeRenderer) {
    XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) r;
    renderer.setBaseShapesVisible(true);
    renderer.setBaseShapesFilled(true);
}
```

In the final customisation, a date format override is set for the domain axis.

```
DateAxis axis = (DateAxis) plot.getDomainAxis();
axis.setDateFormatOverride(new SimpleDateFormat("MMM-yyyy"));
```

When this is set, the axis will continue to “auto-select” a `DateTickUnit` from the collection of standard tick units, but it will ignore the formatting from the tick unit and use the override format instead.

### 8.2.6 The Complete Program

The code for the demonstration application is presented in full, complete with the import statements:

```
/*
 * -----
 * TimeSeriesDemo.java
 * -----
 * (C) Copyright 2002-2005, by Object Refinery Limited.
 *
 */

package demo;

import java.awt.Color;
import java.text.SimpleDateFormat;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.DateAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.time.Month;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.data.xy.XYDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RectangleInsets;
import org.jfree.ui.RefineryUtilities;

/**
 * An example of a time series chart. For the most part, default settings are
```

```

* used, except that the renderer is modified to show filled shapes (as well as
* lines) at each data point.
* <p>
* IMPORTANT NOTE: THIS DEMO IS DOCUMENTED IN THE JFREECHART DEVELOPER GUIDE.
* DO NOT MAKE CHANGES WITHOUT UPDATING THE GUIDE ALSO!!
*/
public class TimeSeriesDemo1 extends ApplicationFrame {

    /**
     * A demonstration application showing how to create a simple time series
     * chart. This example uses monthly data.
     *
     * @param title the frame title.
     */
    public TimeSeriesDemo1(String title) {
        super(title);
        XYDataset dataset = createDataset();
        JFreeChart chart = createChart(dataset);
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(500, 270));
        chartPanel.setMouseZoomable(true, false);
        setContentPane(chartPanel);
    }

    /**
     * Creates a chart.
     *
     * @param dataset a dataset.
     *
     * @return A chart.
     */
    private static JFreeChart createChart(XYDataset dataset) {

        JFreeChart chart = ChartFactory.createTimeSeriesChart(
            "Legal & General Unit Trust Prices", // title
            "Date", // x-axis label
            "Price Per Unit", // y-axis label
            dataset, // data
            true, // create legend?
            true, // generate tooltips?
            false // generate URLs?
        );

        chart.setBackgroundPaint(Color.white);

        XYPlot plot = (XYPlot) chart.getPlot();
        plot.setBackgroundPaint(Color.lightGray);
        plot.setDomainGridlinePaint(Color.white);
        plot.setRangeGridlinePaint(Color.white);
        plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 5.0));
        plot.setDomainCrosshairVisible(true);
        plot.setRangeCrosshairVisible(true);

        XYItemRenderer r = plot.getRenderer();
        if (r instanceof XYLineAndShapeRenderer) {
            XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) r;
            renderer.setBaseShapesVisible(true);
            renderer.setBaseShapesFilled(true);
        }

        DateAxis axis = (DateAxis) plot.getDomainAxis();
        axis.setDateFormatOverride(new SimpleDateFormat("MMM-yyyy"));

        return chart;
    }
}

```

```

 * Creates a dataset, consisting of two series of monthly data.
 *
 * @return the dataset.
 */
private static XYDataset createDataset() {

    TimeSeries s1 = new TimeSeries("L&G European Index Trust", Month.class);
    s1.add(new Month(2, 2001), 181.8);
    s1.add(new Month(3, 2001), 167.3);
    s1.add(new Month(4, 2001), 153.8);
    s1.add(new Month(5, 2001), 167.6);
    s1.add(new Month(6, 2001), 158.8);
    s1.add(new Month(7, 2001), 148.3);
    s1.add(new Month(8, 2001), 153.9);
    s1.add(new Month(9, 2001), 142.7);
    s1.add(new Month(10, 2001), 123.2);
    s1.add(new Month(11, 2001), 131.8);
    s1.add(new Month(12, 2001), 139.6);
    s1.add(new Month(1, 2002), 142.9);
    s1.add(new Month(2, 2002), 138.7);
    s1.add(new Month(3, 2002), 137.3);
    s1.add(new Month(4, 2002), 143.9);
    s1.add(new Month(5, 2002), 139.8);
    s1.add(new Month(6, 2002), 137.0);
    s1.add(new Month(7, 2002), 132.8);

    TimeSeries s2 = new TimeSeries("L&G UK Index Trust", Month.class);
    s2.add(new Month(2, 2001), 129.6);
    s2.add(new Month(3, 2001), 123.2);
    s2.add(new Month(4, 2001), 117.2);
    s2.add(new Month(5, 2001), 124.1);
    s2.add(new Month(6, 2001), 122.6);
    s2.add(new Month(7, 2001), 119.2);
    s2.add(new Month(8, 2001), 116.5);
    s2.add(new Month(9, 2001), 112.7);
    s2.add(new Month(10, 2001), 101.5);
    s2.add(new Month(11, 2001), 106.1);
    s2.add(new Month(12, 2001), 110.3);
    s2.add(new Month(1, 2002), 111.7);
    s2.add(new Month(2, 2002), 111.0);
    s2.add(new Month(3, 2002), 109.6);
    s2.add(new Month(4, 2002), 113.2);
    s2.add(new Month(5, 2002), 111.6);
    s2.add(new Month(6, 2002), 108.8);
    s2.add(new Month(7, 2002), 101.6);

    TimeSeriesCollection dataset = new TimeSeriesCollection();
    dataset.addSeries(s1);
    dataset.addSeries(s2);

    dataset.setDomainIsPointsInTime(true);

    return dataset;
}

/**
 * Creates a panel for the demo (used by SuperDemo.java).
 *
 * @return A panel.
 */
public static JPanel createDemoPanel() {
    JFreeChart chart = createChart(createDataset());
    return new ChartPanel(chart);
}

/**
 * Starting point for the demonstration application.

```

```
*  
* @param args  ignored.  
*/  
public static void main(String[] args) {  
  
    TimeSeriesDemo1 demo = new TimeSeriesDemo1("Time Series Demo 1");  
    demo.pack();  
    RefineryUtilities.centerFrameOnScreen(demo);  
    demo.setVisible(true);  
  
}  
  
}
```

# Chapter 9

# Customising Charts

## 9.1 Introduction

JFreeChart has been designed to be highly customisable. There are many attributes that you can set to change the default appearance of your charts. In this section, some common techniques for customising charts are presented.

## 9.2 Chart Attributes

### 9.2.1 Overview

At the highest level, you can customise the appearance of your charts using methods in the `JFreeChart` class. This allows you to control:

- the chart border;
- the chart title and sub-titles;
- the background color and/or image;
- the rendering hints that are used to draw the chart, including whether or not *anti-aliasing* is used;

These items are described in the following sections.

### 9.2.2 The Chart Border

JFreeChart can draw a border around the outside of a chart. By default, no border is drawn, but you can change this using the `setBorderVisible()` method. The color and line-style for the border are controlled by the `setBorderPaint()` and `setBorderStroke()` methods.

Note: if you are displaying your chart inside a `ChartPanel`, then you might prefer to use the border facilities provided by Swing.

### 9.2.3 The Chart Title

A chart has one title that can appear at the top, bottom, left or right of the chart (you can also add subtitles—see the next section). The title is an instance of `TextTitle`. You can obtain a reference to the title using the `getTitle()` method:

```
TextTitle title = chart.getTitle();
```

To modify the title text (without changing the font or position):

```
chart.setTitle("A Chart Title");
```

The placement of the title at the top, bottom, left or right of the chart is controlled by a property of the title itself. To move the title to the bottom of the chart:

```
chart.getTitle().setPosition(RectangleEdge.BOTTOM);
```

If you prefer to have no title on your chart, you can set the title to `null`.

#### 9.2.4 Subtitles

A chart can have any number of subtitles. To add a sub-title to a chart, create a subtitle (any subclass of `Title`) and add it to the chart. For example:

```
TextTitle subtitle1 = new TextTitle("A Subtitle");
chart.addSubtitle(subtitle1);
```

You can add as many sub-titles as you like to a chart, but keep in mind that as you add more sub-titles there will be less and less space available for drawing the chart.

To modify an existing sub-title, you need to get a reference to the sub-title. For example:

```
Title subtitle = chart.getSubtitle(0);
```

You will need to cast the `Title` reference to an appropriate subclass before you can change its properties.

You can check the number of sub-titles using the `getSubtitleCount()` method.

#### 9.2.5 Setting the Background Color

You can use the `setBackgroundPaint()` method to set the background color for a chart.<sup>1</sup> For example:

```
chart.setBackgroundPaint(Color.blue);
```

You can use any implementation of the `Paint` interface, including the Java classes `Color`, `GradientPaint` and `TexturePaint`. For example:

```
Paint p = new GradientPaint(0, 0, Color.white, 1000, 0, Color.green));
chart.setBackgroundPaint(p);
```

You can also set the background paint to `null`, which is recommended if you have specified a background image for your chart.

#### 9.2.6 Using a Background Image

You can use the `setBackgroundImage()` method to set a background image for a chart.

```
chart.setBackgroundImage(JFreeChart.INFO.getLogo());
```

By default, the image will be scaled to fit the area that the chart is being drawn into, but you can change this using the `setBackgroundImageAlignment()` method.

```
chart.setBackgroundImageAlignment(Align.TOP_LEFT);
```

Using the `setBackgroundImageAlpha()` method, you can control the alpha-transparency for the image.

If you want an image to fill only the *data area* of your chart (that is, the area inside the axes), then you need to add a background image to the chart's `Plot` (described later).

---

<sup>1</sup>You can also set the background color for the chart's plot area, which has a slightly different effect—refer to the `Plot` class for details.

### 9.2.7 Rendering Hints

JFreeChart uses the Java2D API to draw charts. Within this API, you can specify *rendering hints* to fine tune aspects of the way that the rendering engine works.

JFreeChart allows you to specify the rendering hints to be passed to the Java2D API when charts are drawn—use the `setRenderingHints()` method.

As a convenience, a method is provided to turn anti-aliasing on or off. With anti-aliasing on, charts appear to be smoother but they take longer to draw:

```
// turn on antialiasing...
chart.setAntiAlias(true);
```

By default, charts are drawn with anti-aliasing turned on.

## 9.3 Plot Attributes

### 9.3.1 Overview

The `JFreeChart` class delegates a lot of the work in drawing a chart to the `Plot` class (or, rather, to a specific subclass of `Plot`). The `getPlot()` method in the `JFreeChart` class returns a reference to the plot being used by the chart.

```
Plot plot = chart.getPlot();
```

You may need to cast this reference to a specific subclass of `Plot`, for example:

```
CategoryPlot plot = chart.getCategoryPlot();
```

...or:

```
XYPlot plot = chart.getXYPlot();
```

Note that these methods will throw a `ClassCastException` if the plot is not an appropriate class.

### 9.3.2 Which Plot Subclass?

How do you know which subclass of `Plot` is being used by a chart? As you gain experience with JFreeChart, it will become clear which charts use `CategoryPlot` and which charts use `XYPlot`. If in doubt, take a look in the `ChartFactory` class source code to see how each chart type is put together.

### 9.3.3 Setting the Background Paint

You can use the `setBackgroundPaint()` method to set the background color for a plot. For example:

```
Plot plot = chart.getPlot();
plot.setBackgroundPaint(Color.white);
```

You can use any implementation of the `Paint` interface, including the Java classes `Color`, `GradientPaint` and `TexturePaint`. You can also set the background paint to `null`.

### 9.3.4 Using a Background Image

You can use the `setBackgroundImage()` method to set a background image for a plot:

```
Plot plot = chart.getPlot();
plot.setBackgroundImage(JFreeChart.INFO.getLogo());
```

By default, the image will be scaled to fit the area that the plot is being drawn into. You can change this using the `setBackgroundImageAlignment()` method:

```
plot.setBackgroundImageAlignment(Align.BOTTOM_RIGHT);
```

Use the `setBackgroundAlpha()` method to control the alpha-transparency used for the image.

If you prefer your image to fill the entire chart area, then you need to add a background image to the `JFreeChart` object (described previously).

## 9.4 Axis Attributes

### 9.4.1 Overview

The majority of charts created with JFreeChart have two axes, a *domain axis* and a *range axis*. Of course, there are some charts (for example, pie charts) that don't have axes at all. For charts where axes are used, the `Axis` objects are managed by the plot.

### 9.4.2 Obtaining an Axis Reference

Before you can change the properties of an axis, you need to obtain a reference to the axis. The plot classes `CategoryPlot` and `XYPlot` both have methods `getDomainAxis()` and `getRangeAxis()`.

These methods return a reference to a `ValueAxis`, except in the case of the `CategoryPlot`, where the *domain axis* is an instance of `CategoryAxis`.

```
// get an axis reference...
CategoryPlot plot = chart.getCategoryPlot();
CategoryAxis domainAxis = plot.getDomainAxis();

// change axis properties...
domainAxis.setLabel("Categories");
domainAxis.setLabelFont(someFont);
```

There are many different subclasses of the `CategoryAxis` and `ValueAxis` classes. Sometimes you will need to cast your axis reference to a more specific subclass, in order to access some of its attributes. For example, if you know that your range axis is a `NumberAxis` (and the range axis almost always is), then you can do the following:

```
XYPlot plot = chart.getXYPlot();
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setAutoRange(false);
```

### 9.4.3 Setting the Axis Label

You can use the `setLabel()` method to change the axis label. If you would prefer not to have a label for your axis, just set it to `null`.

You can change the font, color and insets (the space around the outside of the label) with the methods `setLabelFont()`, `setLabelPaint()`, and `setLabelInsets()`, defined in the `Axis` class.

### 9.4.4 Rotating Axis Labels

When an axis is drawn at the left or right of a plot (a "vertical" axis), the label is automatically rotated by 90 degrees to minimise the space required. If you prefer to have the label drawn horizontally, you can change the label angle:

```
XYPlot plot = chart.getXYPlot();
ValueAxis axis = plot.getRangeAxis();
axis.setLabelAngle(Math.PI / 2.0);
```

Note that the angle is specified in *radians* (`Math.PI` = 180 degrees).

### 9.4.5 Hiding Tick Labels

To hide the tick labels for an axis:

```
CategoryPlot plot = chart.getCategoryPlot();
ValueAxis axis = plot.getRangeAxis();
axis.setTickLabelsVisible(false);
```

For a `CategoryAxis`, `setTickLabelsVisible(false)` will hide the category labels.

#### 9.4.6 Hiding Tick Marks

To hide the tick marks for an axis:

```
XYPlot plot = chart.getXYPlot();
Axis axis = plot.getDomainAxis();
axis.setTickMarksVisible(false);
```

Category axes do not have tick marks.

#### 9.4.7 Setting the Tick Size

By default, numerical and date axes automatically select a tick size so that the tick labels will not overlap. You can override this by setting your own tick unit using the `setTickUnit()` method.

Alternatively, for a `NumberAxis` or a `DateAxis` you can specify your own set of tick units from which the axis will automatically select an appropriate tick size. This is described in the following sections.

#### 9.4.8 Specifying “Standard” Number Tick Units

In the `NumberAxis` class, there is a method `setStandardTickUnits()` that allows you to supply your own set of tick units for the “auto tick unit selection” mechanism.

One common application is where you have a number axis that should only display integers. In this case, you don’t want tick units of (say) 0.5 or 0.25. There is a (static) method in the `NumberAxis` class that returns a set of standard integer tick units:

```
XYPlot plot = chart.getXYPlot();
NumberAxis axis = (NumberAxis) plot.getRangeAxis();
TickUnitSource units = NumberAxis.createIntegerTickUnits();
axis.setStandardTickUnits(units);
```

You are free to create your own `TickUnits` collection, if you want greater control over the standard tick units.

#### 9.4.9 Specifying “Standard” Date Tick Units

Similar to the case in the previous section, the `DateAxis` class has a method `setStandardTickUnits()` that allows you to supply your own set of tick units for the “auto tick unit selection” mechanism.

The `createStandardDateTickUnits()` method returns the default collection for a `DateAxis`, but you are free to create your own `TickUnits` collection if you want greater control over the standard tick units.

# Chapter 10

## Dynamic Charts

### 10.1 Overview

To illustrate the use of JFreeChart for creating “dynamic” charts, this section presents a sample application that displays a frequently updating chart of JVM memory usage and availability.

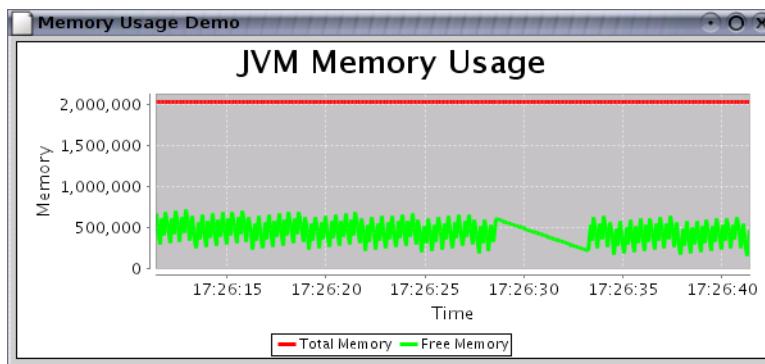


Figure 10.1: A dynamic chart demo

### 10.2 Background

#### 10.2.1 Event notification

JFreeChart uses an *event notification mechanism* that allows it to respond to changes to any component of the chart. For example, whenever a dataset is updated, a `DatasetChangeEvent` is sent to all listeners that are registered with the dataset. This triggers the following sequence of events:

- the plot (which registers itself with the dataset as a `DatasetChangeListener`) receives notification of the dataset change. It updates the axis ranges (if necessary) then passes on a `PlotChangeEvent` to all *its* registered listeners;
- the chart receives notification of the plot change event, and passes on a `ChartChangeEvent` to all *its* registered listeners;
- finally, for charts that are displayed in a `ChartPanel`, the panel will receive the chart change event. It responds by redrawing the chart—a complete redraw, not just the updated data.

A similar sequence of events happens for all changes to a chart or its subcomponents.

### 10.2.2 Performance

Regarding performance, you need to be aware that JFreeChart wasn't designed specifically for generating *real-time charts*. Each time a dataset is updated, the `ChartPanel` reacts by redrawing the entire chart. Optimisations, such as only drawing the most recently added data point, are difficult to implement in the general case, even more so given the `Graphics2D` abstraction (in the Java2D API) employed by JFreeChart. This limits the number of "frames per second" you will be able to achieve with JFreeChart. Whether this will be an issue for you depends on your data, the requirements of your application, and your operating environment.

## 10.3 The Demo Application

### 10.3.1 Overview

The `MemoryUsageDemo.java` demonstration is included in the JFreeChart demo collection (source code available to purchasers of this guide). You can obtain this from:

```
http://www.object-refinery.com/jfreechart/premium/index.html
```

You will need to enter the username and password supplied with your original purchase of the JFreeChart Developer Guide.

### 10.3.2 Creating the Dataset

The dataset is created using two `TimeSeries` objects (one for the *total memory* and the other for the *free memory*) that are added to a single time series collection:

```
// create two series that automatically discard data > 30 seconds old...
this.total = new TimeSeries("Total", Millisecond.class);
this.total.setMaximumItemAge(30000);
this.free = new TimeSeries("Free", Millisecond.class);
this.free.setMaximumItemAge(30000);
TimeSeriesCollection dataset = new TimeSeriesCollection();
dataset.addSeries(this.total);
dataset.addSeries(this.free);
```

The `maximumItemAge` attribute for each time series is set to 30,000 milliseconds (or 30 seconds) so that whenever new data is added to the series, any observations that are older than 30 seconds are automatically discarded.

### 10.3.3 Creating the Chart

The chart creation (and customisation) follows the standard pattern for all charts. No special steps are required to create a dynamic chart, except that you should ensure that the axes have their `autoRange` attribute set to `true`. It also helps to retain a reference to the dataset used in the chart.

### 10.3.4 Updating the Dataset

In the demo, the dataset is updated by adding data to the two time series from a separate thread, managed by the following timer:

```
class DataGenerator extends Timer implements ActionListener {

    DataGenerator(int interval) {
        super(interval, null);
        addActionListener(this);
    }

    public void actionPerformed(ActionEvent event) {
        long f = Runtime.getRuntime().freeMemory();
        long t = Runtime.getRuntime().totalMemory();
        addTotalObservation(t);
        addFreeObservation(f);
    }
}
```

```

    }
}

```

Note that JFreeChart does not yet use thread synchronisation between the chart drawing code and the dataset update code, so this approach is a little unsafe.

*One other point to note, at one point while investigating reports of a memory leak in JFreeChart, I left this demo running on a test machine for about six days. As the chart updates, you can see the effect of the garbage collector. Over the six day period, the total memory used remained constant while the free memory decreased as JFreeChart discarded temporary objects (garbage), and increased at the points where the garbage collector did its work.*

### 10.3.5 Source Code

For reference, here is the complete source code for the example:

```

/*
 * MemoryUsageDemo.java
 *
 * (C) Copyright 2002-2006, by Object Refinery Limited.
 */

package demo;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.Timer;

import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.DateAxis;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.time.Millisecond;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.ui.RectangleInsets;

/**
 * A demo application showing a dynamically updated chart that displays the
 * current JVM memory usage.
 * <p>
 * <b>IMPORTANT NOTE:> THIS DEMO IS DOCUMENTED IN THE JFREECHART DEVELOPER GUIDE.
 * DO NOT MAKE CHANGES WITHOUT UPDATING THE GUIDE ALSO!>
 */
public class MemoryUsageDemo extends JPanel {

    /** Time series for total memory used. */
    private TimeSeries total;

    /** Time series for free memory. */
    private TimeSeries free;

    /**
     * Creates a new application.
     *
     * @param maxAge  the maximum age (in milliseconds).
     */
    public MemoryUsageDemo(int maxAge) {

        super(new BorderLayout());

        // create two series that automatically discard data more than 30

```

```

// seconds old...
this.total = new TimeSeries("Total Memory", Millisecond.class);
this.total.setMaximumItemAge(maxAge);
this.free = new TimeSeries("Free Memory", Millisecond.class);
this.free.setMaximumItemAge(maxAge);
TimeSeriesCollection dataset = new TimeSeriesCollection();
dataset.addSeries(this.total);
dataset.addSeries(this.free);

DateAxis domain = new DateAxis("Time");
NumberAxis range = new NumberAxis("Memory");
domain.setTickLabelFont(new Font("SansSerif", Font.PLAIN, 12));
range.setTickLabelFont(new Font("SansSerif", Font.PLAIN, 12));
domain.setLabelFont(new Font("SansSerif", Font.PLAIN, 14));
range.setLabelFont(new Font("SansSerif", Font.PLAIN, 14));

XYItemRenderer renderer = new XYLineAndShapeRenderer(true, false);
renderer.setSeriesPaint(0, Color.red);
renderer.setSeriesPaint(1, Color.green);
renderer.setStroke(new BasicStroke(3f, BasicStroke.CAP_BUTT,
    BasicStroke.JOIN_BEVEL));
XYPlot plot = new XYPlot(dataset, domain, range, renderer);
plot.setBackgroundPaint(Color.lightGray);
plot.setDomainGridlinePaint(Color.white);
plot.setRangeGridlinePaint(Color.white);
plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 5.0));
domain.setAutoRange(true);
domain.setLowerMargin(0.0);
domain.setUpperMargin(0.0);
domain.setTickLabelsVisible(true);

range.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

JFreeChart chart = new JFreeChart("JVM Memory Usage",
    new Font("SansSerif", Font.BOLD, 24), plot, true);
chart.setBackgroundPaint(Color.white);
ChartPanel chartPanel = new ChartPanel(chart);
chartPanel.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createEmptyBorder(4, 4, 4, 4),
    BorderFactory.createLineBorder(Color.black)));
add(chartPanel);

}

/**
 * Adds an observation to the 'total memory' time series.
 *
 * @param y  the total memory used.
 */
private void addTotalObservation(double y) {
    this.total.add(new Millisecond(), y);
}

/**
 * Adds an observation to the 'free memory' time series.
 *
 * @param y  the free memory.
 */
private void addFreeObservation(double y) {
    this.free.add(new Millisecond(), y);
}

/**
 * The data generator.
 */
class DataGenerator extends Timer implements ActionListener {

    /**
     * Constructor.
     *
     * @param interval  the interval (in milliseconds)
     */
    DataGenerator(int interval) {
        super(interval, null);
        addActionListener(this);
    }

    /**
     * Adds a new free/total memory reading to the dataset.
     *
     * @param event  the action event.
     */
}

```

```
 */
public void actionPerformed(ActionEvent event) {
    long f = Runtime.getRuntime().freeMemory();
    long t = Runtime.getRuntime().totalMemory();
    addTotalObservation(t);
    addFreeObservation(f);
}

/**
 * Entry point for the sample application.
 *
 * @param args  ignored.
 */
public static void main(String[] args) {

    JFrame frame = new JFrame("Memory Usage Demo");
    MemoryUsageDemo panel = new MemoryUsageDemo(30000);
    frame.getContentPane().add(panel, BorderLayout.CENTER);
    frame.setBounds(200, 120, 600, 280);
    frame.setVisible(true);
    panel.new DataGenerator(100).start();

    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}
}
```

# Chapter 11

## Tooltips

### 11.1 Overview

JFreeChart includes mechanisms for generating, collecting and displaying tool tips for individual components of a chart.

In this section, I describe:

- how to generate tool tips (including customisation of tool tips);
- how tool tips are collected;
- how to display tool tips;
- how to disable tool tips if you don't need them;

### 11.2 Generating Tool Tips

If you want to use tool tips, you need to make sure they are generated as your chart is being drawn. You do this by setting a tool tip generator for your plot or, in many cases, the plot's item renderer.

In the sub-sections that follow, I describe how to set a tool tip generator for the common chart types.

#### 11.2.1 Pie Charts

The `PiePlot` class generates tool tips using the `PieToolTipGenerator` interface. A standard implementation (`StandardPieToolTipGenerator`) is provided, and you are free to create your own implementations.

To set the tool tip generator, use the following method in the `PiePlot` class:

```
↳ public void setToolTipGenerator(PieToolTipGenerator generator);  
Sets the tool tip generator for the pie chart. If you set this to null, no tool tips will be  
generated.
```

#### 11.2.2 Category Charts

Category charts—including most of the bar charts generated by JFreeChart—are based on the `CategoryPlot` class and use a `CategoryItemRenderer` to draw each data item. The `CategoryToolTipGenerator` interface specifies the method via which the renderer will obtain tool tips (if required).

To set the tool tip generator for a category plot's item renderer, use the following method (defined in the `AbstractCategoryItemRenderer` class):

```
► public void setToolTipGenerator(CategoryToolTipGenerator generator);
```

Sets the tool tip generator for the renderer. If you set this to `null`, no tool tips will be generated.

### 11.2.3 XY Charts

XY charts—including scatter plots and all the time series charts generated by JFreeChart—are based on the `XYPlot` class and use an `XYItemRenderer` to draw each data item. The renderer generates tool tips (if required) using an `XYToolTipGenerator`.

To set the tool tip generator for an XY plot's item renderer, use the following method (defined in the `AbstractXYItemRenderer` class):

```
► public void setToolTipGenerator(XYToolTipGenerator generator);
```

Sets the tool tip generator for the renderer. If you set this to `null`, no tool tips will be generated.

## 11.3 Collecting Tool Tips

Tool tips are collected, along with other chart entity information, using the `ChartRenderingInfo` class. You need to supply an instance of this class to `JFreeChart`'s `draw()` method, otherwise no tool tip information will be recorded (even if a generator has been registered with the plot or the plot's item renderer, as described in the previous sections).

Fortunately, the `ChartPanel` class takes care of this automatically, so if you are displaying your charts using the `ChartPanel` class you do not need to worry about how tool tips are collected—it is done for you.

## 11.4 Displaying Tool Tips

Tool tips are automatically displayed by the `ChartPanel` class, provided that you have set up a tool tip generator for the plot (or the plot's renderer).

You can also enable or disable the *display* of tool tips in the `ChartPanel` class, using this method:

```
► public void setDisplayToolTips(boolean flag);
```

Switches the display of tool tips on or off.

## 11.5 Disabling Tool Tips

The most effective way to disable tool tips is to set the tool tip generator to `null`. This ensures that no tool tip information is even generated, which can save memory and processing time (particularly for charts with large datasets).

You can also disable the *display* of tool tips in the `ChartPanel` class, using the method given in the previous section.

## 11.6 Customising Tool Tips

You can take full control of the text generated for each tool tip by providing your own implementation of the appropriate tool tip generator interface.

# Chapter 12

## Item Labels

### 12.1 Introduction

#### 12.1.1 Overview

For many chart types, JFreeChart will allow you to display *item labels* in, on or near to each data item in a chart. For example, you can display the actual value represented by the bars in a bar chart—see figure 12.1.

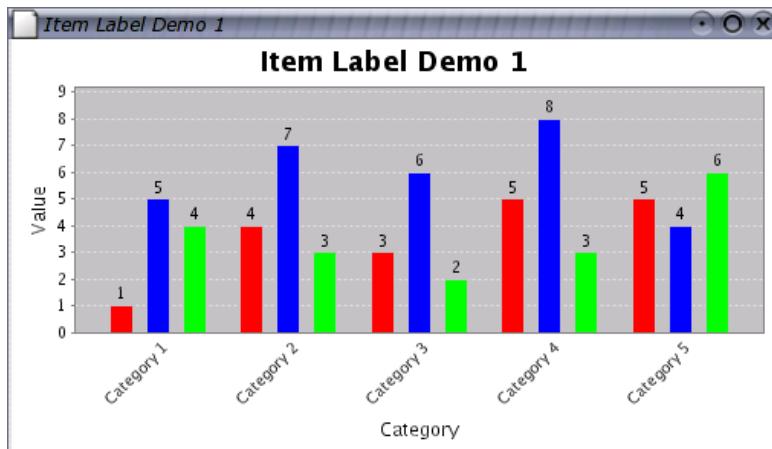


Figure 12.1: A bar chart with item labels

This chapter covers how to:

- make item labels visible (for the chart types that support item labels);
- change the appearance (font and color) of item labels;
- specify the location of item labels;
- customise the item label text.

A word of advice: *use this feature sparingly*. Charts are supposed to summarise your data—if you feel it is necessary to display the actual data values all over your chart, then perhaps your data is better presented in a table format.

### 12.1.2 Limitations

There are some limitations with respect to the item labels in the current release of JFreeChart:

- some renderers do not support item labels;
- axis ranges are not automatically adjusted to take into account the item labels—some labels may disappear off the chart if sufficient margins are not set (use the `setUpperMargin()` and/or `setLowerMargin()` methods in the relevant axis to adjust this).

In future releases of JFreeChart, some or all of these limitations will be addressed.

## 12.2 Displaying Item Labels

### 12.2.1 Overview

Item labels are not visible by default, so you need to configure the renderer to create and display them. This involves two steps:

- assign a `CategoryItemLabelGenerator` or `XYItemLabelGenerator` to the renderer—this is an object that assumes responsibility for creating the labels;
- set a flag in the renderer to make the labels visible, either for all series or, if you prefer, on a per series basis.

In addition, you have the option to customise the position, font and color of the item labels. These steps are detailed in the following sections.

### 12.2.2 Assigning a Label Generator

Item labels are created by a label generator that is assigned to a renderer (the same mechanism is also used for tooltips).

To assign a generator to a `CategoryItemRenderer`, use the following code:

```
CategoryItemRenderer renderer = plot.getRenderer();
CategoryItemLabelGenerator generator = new StandardCategoryItemLabelGenerator(
    "{2}", new DecimalFormat("0.00"));
renderer.setLabelGenerator(generator);
```

Similarly, to assign a generator to an `XYItemRenderer`, use the following code:

```
XYItemRenderer renderer = plot.getRenderer();
XYItemLabelGenerator generator = new StandardXYItemLabelGenerator(
    "{2}", new DecimalFormat("0.00"));
renderer.setLabelGenerator(generator);
```

You can customise the behaviour of the standard generator via settings that you can apply in the constructor, or you can create your own generator as described in section 12.5.2.

### 12.2.3 Making Labels Visible For All Series

The `setItemLabelsVisible()` method sets a flag that controls whether or not the item labels are displayed (note that a label generator must be assigned to the renderer, or there will be no labels to display). For a `CategoryItemRenderer`:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelsVisible(true);
```

Similarly, for a `XYItemRenderer`:

```
XYItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelsVisible(true);
```

Once set, this flag takes precedence over any *per series* settings you may have made elsewhere. In order for the per series settings to apply, you need to set this flag to `null` (see section 12.2.4).

### 12.2.4 Making Labels Visible For Selected Series

If you prefer, you can set flags that control the visibility of the item labels on a per series basis. For example, item labels are displayed only for the first series in figure 12.2.

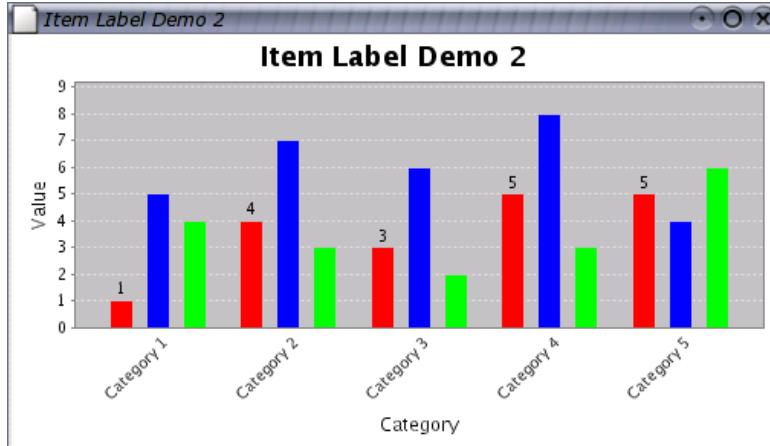


Figure 12.2: Item labels for selected series only

You can use code similar to the following:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelsVisible(null); // clears the ALL series flag
renderer.setSeriesItemLabelsVisible(0, true);
renderer.setSeriesItemLabelsVisible(1, false);
```

Notice that the flag for “all series” has been set to `null`—this is important, because the “all series” flag takes precedence over the “per series” flags.

### 12.2.5 Troubleshooting

If, after following the steps outlined in the previous sections, you still can’t see any labels on your chart, there are a couple of things to consider:

- the renderer must have a label generator assigned to it—this is an object that creates the text items that are used for each label.
- some renderers don’t yet support the display of item labels (refer to the documentation for the renderer you are using).

## 12.3 Item Label Appearance

### 12.3.1 Overview

You can change the appearance of the item labels by changing the font and/or the color used to display the labels. As for most other renderer attributes, the settings can be made once for *all series*, or on a *per series* basis.

*In the current release of JFreeChart, labels are drawn with a transparent background. You cannot set a background color for the labels, nor can you specify that a border be drawn around the labels. This may change in the future.*

### 12.3.2 Changing the Label Font

To change the font for the item labels in all series, you can use code similar to the following:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelFont(new Font("SansSerif", Font.PLAIN, 10));
```

Similarly, to set the font for individual series:

```
CategoryItemRenderer renderer = plot.getRenderer();

// clear the settings for ALL series...
renderer.setItemLabelFont(null);

// add settings for individual series...
renderer.setSeriesItemLabelFont(0, new Font("SansSerif", Font.PLAIN, 10));
renderer.setSeriesItemLabelFont(1, new Font("SansSerif", Font.BOLD, 10));
```

Notice how the font for all series has been set to `null` to prevent it from overriding the per series settings.

### 12.3.3 Changing the Label Color

To change the color for the item labels in all series, you can use code similar to the following:

```
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setItemLabelPaint(Color.red);
```

Similarly, to set the color for individual series:

```
CategoryItemRenderer renderer = plot.getRenderer();

// clear the settings for ALL series...
renderer.setItemLabelPaint(null);

// add settings for individual series...
renderer.setSeriesItemLabelPaint(0, Color.red);
renderer.setSeriesItemLabelPaint(1, Color.blue);
```

Once again, notice how the paint for all series has been set to `null` to prevent it from overriding the per series settings.

## 12.4 Item Label Positioning

### 12.4.1 Overview

The positioning of item labels is controlled by four attributes that are combined into an `ItemLabelPosition` object. You can define label positions for items with positive and negative values independently, via the following methods in the `CategoryItemRenderer` interface:

```
public void setPositiveItemLabelPosition(ItemLabelPosition position);
public void setNegativeItemLabelPosition(ItemLabelPosition position);
```

Understanding how these attributes impact the final position of individual labels is key to getting good results from the item label features in JFreeChart.

There are four attributes:

- the *item label anchor* - determines the base location for the item label;
- the *text anchor* - determines the point on the label that is aligned to the base location;
- the *rotation anchor* - this is the point on the label text about which the rotation (if any) is applied;
- the *rotation angle* - the angle through which the label is rotated.

These are described in the following sections.

### 12.4.2 The Item Label Anchor

The purpose of the item label anchor setting is to determine an  $(x, y)$  location on the chart that is near to the data item that is being labelled. The label is then aligned to this anchor point when it is being drawn. Refer to the [ItemLabelAnchor](#) documentation for more information.

### 12.4.3 The Text Anchor

The text anchor determines which point on the label should be aligned with the anchor point described in the previous section. It is possible to align the center of the label with the anchor point, or the top-right of the label, or the bottom-left, and so on...refer to the [TextAnchor](#) documentation for all the options.

Running the [DrawStringDemo](#) application in the `org.jfree.demo` package (included in the JCommon distribution) is a good way to gain an understanding of how the text anchor is used to align labels to a point on the screen.

### 12.4.4 The Rotation Anchor

The rotation anchor defines a point on the label about which the rotation (if any) will be applied to the label. The [DrawStringDemo](#) class also demonstrates this feature.

### 12.4.5 The Rotation Angle

The rotation angle defines the angle through which the label is rotated. The angle is specified in radians, and the rotation point is defined by the rotation anchor described in the previous section.

## 12.5 Customising the Item Label Text

### 12.5.1 Overview

Up to this point, we've relied on the label generator built in to JFreeChart to create the text for the item labels. If you want to have complete control over the label text, you can write your own class that implements the [CategoryItemLabelGenerator](#) interface.

In this section I provide a brief overview of the technique for implementing a custom label generator, then present two examples to illustrate the type of results you can achieve with this technique.

### 12.5.2 Implementing a Custom Item Label Generator

To develop a custom label generator, you simply need to write a class that implements the method defined in the [CategoryItemLabelGenerator](#) interface:

```
public String generateLabel(CategoryDataset dataset, int series, int category);
```

The renderer will call this method at the point that it requires a `String` use for a label, and will pass in the `CategoryDataset` and the `series` and `category` indices for the current item. This means that you have full access to the entire dataset (not just the current item) for the creation of the label.

The method can return an arbitrary `String` value, so you can apply any formatting you want to the result. It is also valid to return `null` if you prefer no label to be displayed.

All this is best illustrated by way of examples, which are provided in the following sections.

## 12.6 Example 1 - Values Above a Threshold

### 12.6.1 Overview

In this first example, the goal is to display labels for the items that have a value greater than some predefined threshold value (see figure 12.3).

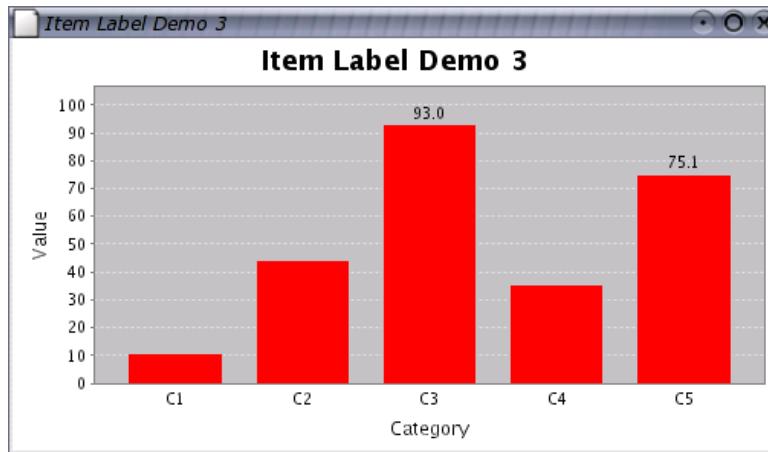


Figure 12.3: Item labels above a threshold

It isn't all that difficult to achieve, we simply need to:

- write a class that implements the `CategoryItemLabelGenerator` interface, and implement the `generateItemLabel()` method in such a way that it returns `null` for any item where the value is less than the threshold;
- create an instance of this new class, and assign it to the renderer using the `setLabelGenerator()` method.

### 12.6.2 Source Code

The complete source code is presented below.

```
/*
 * -----
 * ItemLabelDemo1.java
 * -----
 * (C) Copyright 2004, 2005, by Object Refinery Limited.
 *
 */
package demo;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.text.NumberFormat;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.labels.AbstractCategoryItemLabelGenerator;
import org.jfree.chart.labels.CategoryItemLabelGenerator;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.CategoryItemRenderer;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
```

```

import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RainbowUtilities;

/**
 * A simple demo showing a label generator that only displays labels for items
 * with a value that is greater than some threshold.
 */
public class ItemLabelDemo1 extends ApplicationFrame {

    /**
     * A custom label generator.
     */
    static class LabelGenerator extends AbstractCategoryItemLabelGenerator
        implements CategoryItemLabelGenerator {

        /** The threshold. */
        private double threshold;

        /**
         * Creates a new generator that only displays labels that are greater
         * than or equal to the threshold value.
         *
         * @param threshold the threshold value.
         */
        public LabelGenerator(double threshold) {
            super("", NumberFormat.getInstance());
            this.threshold = threshold;
        }

        /**
         * Generates a label for the specified item. The label is typically a
         * formatted version of the data value, but any text can be used.
         *
         * @param dataset the dataset (<code>null</code> not permitted).
         * @param series the series index (zero-based).
         * @param category the category index (zero-based).
         *
         * @return the label (possibly <code>null</code>).
         */
        public String generateLabel(CategoryDataset dataset,
                                    int series,
                                    int category) {

            String result = null;
            Number value = dataset.getValue(series, category);
            if (value != null) {
                double v = value.doubleValue();
                if (v > this.threshold) {
                    result = value.toString(); // could apply formatting here
                }
            }
            return result;
        }
    }

    /**
     * Creates a new demo instance.
     *
     * @param title the frame title.
     */
    public ItemLabelDemo1(String title) {

        super(title);
        CategoryDataset dataset = createDataset();
        JFreeChart chart = createChart(dataset);
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new Dimension(500, 270));
        setContentPane(chartPanel);
    }

    /**
     * Returns a sample dataset.
     *
     * @return The dataset.
     */
    private static CategoryDataset createDataset() {

        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

```

```

dataset.addValue(11.0, "S1", "C1");
dataset.addValue(44.3, "S1", "C2");
dataset.addValue(93.0, "S1", "C3");
dataset.addValue(35.6, "S1", "C4");
dataset.addValue(75.1, "S1", "C5");
return dataset;
}

/**
 * Creates a sample chart.
 *
 * @param dataset the dataset.
 *
 * @return the chart.
 */
private static JFreeChart createChart(CategoryDataset dataset) {

    // create the chart...
    JFreeChart chart = ChartFactory.createBarChart(
        "Item Label Demo 1",           // chart title
        "Category",                   // domain axis label
        "Value",                      // range axis label
        dataset,                      // data
        PlotOrientation.VERTICAL,     // orientation
        false,                        // include legend
        true,                         // tooltips?
        false                         // URLs?
    );
    chart.setBackgroundPaint(Color.white);

    CategoryPlot plot = chart.getCategoryPlot();
    plot.setBackgroundPaint(Color.lightGray);
    plot.setDomainGridlinePaint(Color.white);
    plot.setRangeGridlinePaint(Color.white);

    NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
    rangeAxis.setUpperMargin(0.15);

    CategoryItemRenderer renderer = plot.getRenderer();
    renderer.setItemLabelGenerator(new LabelGenerator(50.0));
    renderer.setItemLabelFont(new Font("Serif", Font.PLAIN, 20));
    renderer.setItemLabelsVisible(true);

    return chart;
}

/**
 * Creates a panel for the demo (used by SuperDemo.java).
 *
 * @return A panel.
 */
public static JPanel createDemoPanel() {
    JFreeChart chart = createChart(createDataset());
    return new ChartPanel(chart);
}

/**
 * Starting point for the demonstration application.
 *
 * @param args ignored.
 */
public static void main(String[] args) {

    ItemLabelDemo1 demo = new ItemLabelDemo1("Item Label Demo 1");
    demo.pack();
    RefineryUtilities.centerFrameOnScreen(demo);
    demo.setVisible(true);
}
}

```

## 12.7 Example 2 - Displaying Percentages

### 12.7.1 Overview

In this example, the requirement is to display a bar chart where each bar is labelled with the value represented by the bar and also a percentage (where the percentage is calculated relative to a particular bar within the series OR the total of all the values in the series)—see figure 12.4.

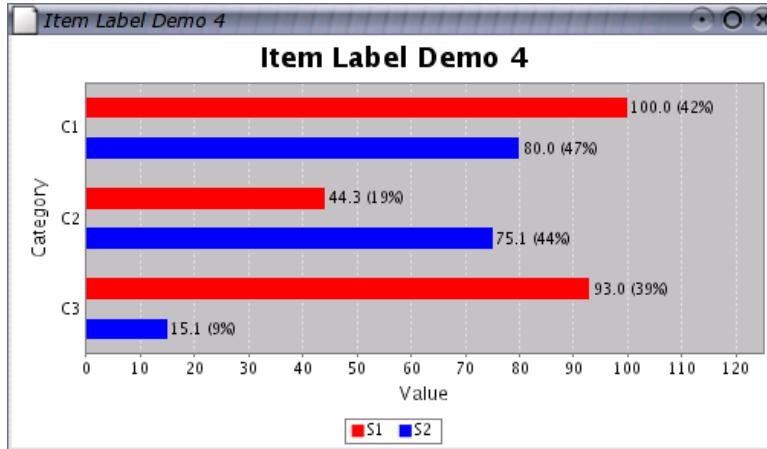


Figure 12.4: Percentage item labels

In this implementation, the label generator calculates the percentage value on-the-fly. If a category index is supplied in the constructor, the base value used to calculate the percentage is taken from the specified category within the current series. If no category index is available, then the total of all the values in the current series is used as the base.

A default percentage formatter is created within the label generator—a more sophisticated implementation would provide the ability for the formatter to be customised via the generator's constructor.

### 12.7.2 Source Code

The complete source code follows.

```
/*
 * -----
 * ItemLabelDemo2.java
 * -----
 * (C) Copyright 2005, by Object Refinery Limited.
 *
 */
package demo;

import java.awt.Color;

/**
 * A simple demo showing a label generator that displays labels that include
 * a percentage calculation.
 */
public class ItemLabelDemo2 extends ApplicationFrame {

    /**
     * A custom label generator.
     */
    static class LabelGenerator extends AbstractCategoryItemLabelGenerator
        implements CategoryItemLabelGenerator {

        /**
         * The index of the category on which to base the percentage

```

```

    * (null = use series total).
    */
private Integer category;

/** A percent formatter. */
private NumberFormat formatter = NumberFormat.getPercentInstance();

/**
 * Creates a new label generator that displays the item value and a
 * percentage relative to the value in the same series for the
 * specified category.
 *
 * @param category the category index (zero-based).
 */
public LabelGenerator(int category) {
    this(new Integer(category));
}

/**
 * Creates a new label generator that displays the item value and
 * a percentage relative to the value in the same series for the
 * specified category. If the category index is <code>null</code>,
 * the total of all items in the series is used.
 *
 * @param category the category index (<code>null</code> permitted).
 */
public LabelGenerator(Integer category) {
    super("", NumberFormat.getInstance());
    this.category = category;
}

/**
 * Generates a label for the specified item. The label is typically
 * a formatted version of the data value, but any text can be used.
 *
 * @param dataset the dataset (<code>null</code> not permitted).
 * @param series the series index (zero-based).
 * @param category the category index (zero-based).
 *
 * @return the label (possibly <code>null</code>).
 */
public String generateLabel(CategoryDataset dataset,
                            int series,
                            int category) {

    String result = null;
    double base = 0.0;
    if (this.category != null) {
        final Number b = dataset.getValue(series, this.category.intValue());
        base = b.doubleValue();
    } else {
        base = calculateSeriesTotal(dataset, series);
    }
    Number value = dataset.getValue(series, category);
    if (value != null) {
        final double v = value.doubleValue();
        // you could apply some formatting here
        result = value.toString()
                    + "(" + this.formatter.format(v / base) + ")";
    }
    return result;
}

/**
 * Calculates a series total.
 *
 * @param dataset the dataset.
 * @param series the series index.
 *
 * @return The total.
 */
private double calculateSeriesTotal(CategoryDataset dataset, int series) {
    double result = 0.0;
    for (int i = 0; i < dataset.getColumnCount(); i++) {
        Number value = dataset.getValue(series, i);
        if (value != null) {
            result = result + value.doubleValue();
        }
    }
}

```

```

        return result;
    }

}

/**
 * Creates a new demo instance.
 *
 * @param title the frame title.
 */
public ItemLabelDemo2(String title) {

    super(title);
    CategoryDataset dataset = createDataset();
    JFreeChart chart = createChart(dataset);
    ChartPanel chartPanel = new ChartPanel(chart);
    chartPanel.setPreferredSize(new Dimension(500, 270));
    setContentPane(chartPanel);

}

/**
 * Returns a sample dataset.
 *
 * @return the dataset.
 */
private static CategoryDataset createDataset() {

    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    dataset.addValue(100.0, "S1", "C1");
    dataset.addValue(44.3, "S1", "C2");
    dataset.addValue(93.0, "S1", "C3");
    dataset.addValue(80.0, "S2", "C1");
    dataset.addValue(75.1, "S2", "C2");
    dataset.addValue(15.1, "S2", "C3");
    return dataset;
}

/**
 * Creates a sample chart.
 *
 * @param dataset the dataset.
 *
 * @return the chart.
 */
private static JFreeChart createChart(CategoryDataset dataset) {

    // create the chart...
    JFreeChart chart = ChartFactory.createBarChart(
        "Item Label Demo 2",           // chart title
        "Category",                  // domain axis label
        "Value",                     // range axis label
        dataset,                      // data
        PlotOrientation.HORIZONTAL,   // orientation
        true,                        // include legend
        true,                        // tooltips?
        false                         // URLs?
    );

    chart.setBackgroundPaint(Color.white);

    CategoryPlot plot = chart.getCategoryPlot();
    plot.setBackgroundPaint(Color.lightGray);
    plot.setDomainGridlinePaint(Color.white);
    plot.setRangeGridlinePaint(Color.white);
    plot.setRangeAxisLocation(AxisLocation.BOTTOM_OR_LEFT);

    NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
    rangeAxis.setUpperMargin(0.25);

    CategoryItemRenderer renderer = plot.getRenderer();
    renderer.setItemLabelsVisible(true);

    // use one or the other of the following lines to see the
    // different modes for the label generator...
    renderer.setItemLabelGenerator(new LabelGenerator(null));
    //renderer.setLabelGenerator(new LabelGenerator(0));

    return chart;
}

```

```
}

/**
 * Creates a panel for the demo (used by SuperDemo.java).
 *
 * @return A panel.
 */
public static JPanel createDemoPanel() {
    JFreeChart chart = createChart(createDataset());
    return new ChartPanel(chart);
}

/**
 * Starting point for the demonstration application.
 *
 * @param args  ignored.
 */
public static void main(String[] args) {

    ItemLabelDemo2 demo = new ItemLabelDemo2("Item Label Demo 2");
    demo.pack();
    RefineryUtilities.centerFrameOnScreen(demo);
    demo.setVisible(true);
}

}
```

# Chapter 13

## Multiple Axes and Datasets

### 13.1 Introduction

JFreeChart supports the use of multiple axes and datasets in the `CategoryPlot` and `XYPlot` classes. You can use this feature to display two or more datasets on a single chart, while making allowance for the fact that the datasets may contain data of vastly different magnitudes—see figure 13.1 for an example.

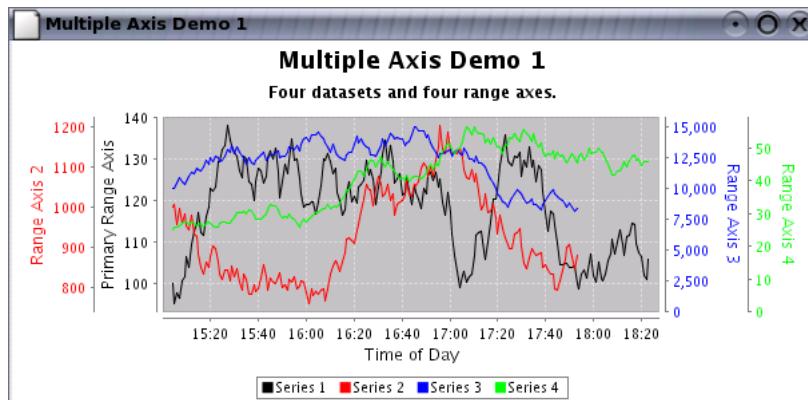


Figure 13.1: A chart with multiple axes

Typical charts constructed with JFreeChart use a plot that has a single *dataset*, a single *renderer*, a single *domain axis* and a single *range axis*. However, it is possible to add multiple datasets, renderers and axes to a plot. In this section, an example is presented showing how to use these additional datasets, renderers and axes.

### 13.2 An Example

#### 13.2.1 Introduction

The `MultipleAxisDemo1.java` application (included in the JFreeChart Demo distribution) provides a good example of how to create a chart with multiple axes. This section provides some notes on the steps taken within that code.

### 13.2.2 Create a Chart

To create a chart with multiple axes, datasets, and renderers, you should first create a regular chart (for example, using the `ChartFactory` class). You can use any chart that is constructed using a `CategoryPlot` or an `XYPlot`. In the example, a time series chart is created as follows:

```
XYDataset dataset1 = createDataset("Series 1", 100.0, new Minute(), 200);
JFreeChart chart = ChartFactory.createTimeSeriesChart(
    "Multiple Axis Demo 1",
    "Time of Day",
    "Primary Range Axis",
    dataset1,
    true,
    true,
    false
);
```

### 13.2.3 Adding an Additional Axis

To add an additional axis to a plot, you can use the `setRangeAxis()` method:

```
NumberAxis axis2 = new NumberAxis("Range Axis 2");
plot.setRangeAxis(1, axis2);
plot.setRangeAxisLocation(1, AxisLocation.BOTTOM_OR_RIGHT);
```

The `setRangeAxis()` method is used to add the axis to the plot. Note that an index of 1 (one) has been used—you can add as many additional axes as you require, by incrementing the index each time you add a new axis.

The `setRangeAxisLocation()` method allows you to specify where the axis will appear on the chart, using the `AxisLocation` class. You can have the axis on the same side as the primary axis, or on the opposite side—the choice is yours. In the example, `BOTTOM_OR_RIGHT` is specified, which means (for a range axis) on the right if the plot has a vertical orientation, or at the bottom if the plot has a horizontal orientation.

At this point, no additional dataset has been added to the chart, so if you were to display the chart you would see the additional axis, but it would have no data plotted against it.

### 13.2.4 Adding an Additional Dataset

To add an additional dataset to a plot, use the `setDataset()` method:

```
XYDataset dataset2 = ... // up to you
plot.setDataset(1, dataset2);
```

By default, the dataset will be plotted *against the primary range axis*. To have the dataset plotted against a different axis, use the `mapDatasetToDomainAxis()` and `mapDatasetToRangeAxis()` methods. These methods accept two arguments, the first is the index of the dataset, and the second is the index of the axis.

### 13.2.5 Adding an Additional Renderer

When you add an additional dataset, usually it makes sense to add an additional renderer to go with the dataset. Use the `setRenderer()` method:

```
XYItemRenderer renderer2 = ... // up to you
plot.setRenderer(1, renderer2);
```

The index (1 in this case) should correspond to the index of the dataset added previously.

Note: if you don't specify an additional renderer, the primary renderer will be used instead. In that case, the series colors will be shared between the primary dataset and the additional dataset.

### 13.3 Hints and Tips

When using multiple axes, you need to provide some visual cue to readers to indicate which axis applies to a particular series. In the `MultipleAxisDemo1.java` application, the color of the axis label text has been changed to match the series color.

Additional demos available for download with the JFreeChart Developer Guide include:

- `DualAxisDemo1.java`
- `DualAxisDemo2.java`
- `DualAxisDemo3.java`
- `DualAxisDemo4.java`
- `MultipleAxisDemo1.java`
- `MultipleAxisDemo2.java`
- `MultipleAxisDemo3.java`

# Chapter 14

## Combined Charts

### 14.1 Introduction

JFreeChart supports *combined charts* via several plot classes that can manage any number of *subplots*:

- `CombinedDomainCategoryPlot` / `CombinedRangeCategoryPlot`;
- `CombinedDomainXYPlot` / `CombinedRangeXYPlot`;

This section presents a few examples that use the combined chart facilities provided by JFreeChart. All the examples are included in the JFreeChart demo collection.

### 14.2 Combined Domain Category Plot

#### 14.2.1 Overview

A *combined domain category plot* is a plot that displays two or more subplots (instances of `CategoryPlot`) that share a common *domain axis*. Each subplot maintains its own *range axis*. An example is shown in figure 14.1.

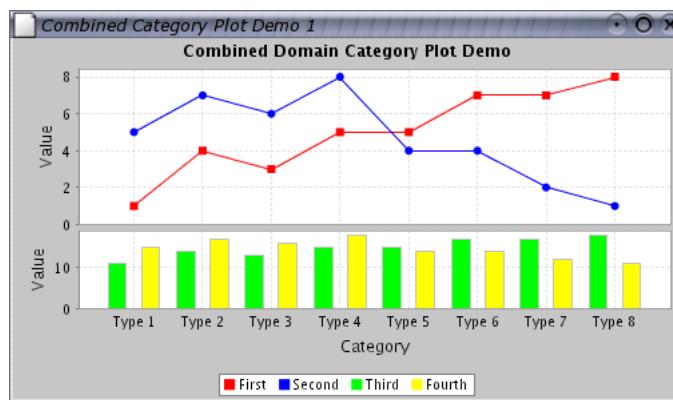


Figure 14.1: A combined domain category plot

It is possible to display this chart with a horizontal or vertical orientation—the example shown has a vertical orientation.

### 14.2.2 Constructing the Chart

A demo application (`CombinedCategoryPlotDemo1.java`, available for download with the JFreeChart Developer Guide) provides an example of how to create this type of chart. The key step is the creation of a `CombinedDomainCategoryPlot` instance, to which subplots are added:

```
CategoryAxis domainAxis = new CategoryAxis("Category");
CombinedDomainCategoryPlot plot = new CombinedDomainCategoryPlot(domainAxis);
plot.add(subplot1, 2);
plot.add(subplot2, 1);

JFreeChart result = new JFreeChart(
    "Combined Domain Category Plot Demo",
    new Font("SansSerif", Font.BOLD, 12),
    plot,
    true
);
```

Notice how `subplot1` has been added with a weight of 2 (the second argument in the `add()` method), while `subplot2` has been added with a weight of 1. This controls the amount of space allocated to each plot.

The subplots are regular `CategoryPlot` instances that have had their domain axis set to `null`. For example, in the demo application the following code is used (it includes some customisation of the subplots):

```
CategoryDataset dataset1 = createDataset1();
NumberAxis rangeAxis1 = new NumberAxis("Value");
rangeAxis1.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
LineAndShapeRenderer renderer1 = new LineAndShapeRenderer();
renderer1.setBaseToolTipGenerator(new StandardCategoryToolTipGenerator());
CategoryPlot subplot1 = new CategoryPlot(dataset1, null, rangeAxis1, renderer1);
subplot1.setDomainGridlinesVisible(true);

CategoryDataset dataset2 = createDataset2();
NumberAxis rangeAxis2 = new NumberAxis("Value");
rangeAxis2.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
BarRenderer renderer2 = new BarRenderer();
renderer2.setBaseToolTipGenerator(new StandardCategoryToolTipGenerator());
CategoryPlot subplot2 = new CategoryPlot(dataset2, null, rangeAxis2, renderer2);
subplot2.setDomainGridlinesVisible(true);
```

## 14.3 Combined Range Category Plot

### 14.3.1 Overview

A *combined range category plot* is a plot that displays two or more subplots (instances of `CategoryPlot`) that share a common *range axis*. Each subplot maintains its own *domain axis*. An example is shown in figure 14.2.

It is possible to display this chart with a horizontal or vertical orientation (the example above has a vertical orientation).

### 14.3.2 Constructing the Chart

A demo application (`CombinedCategoryPlotDemo2.java`, available for download with the JFreeChart Developer Guide) provides an example of how to create this type of chart. The key step is the creation of a `CombinedRangeCategoryPlot` instance, to which subplots are added:

```
ValueAxis rangeAxis = new NumberAxis("Value");
CombinedRangeCategoryPlot plot = new CombinedRangeCategoryPlot(rangeAxis);
plot.add(subplot1, 3);
plot.add(subplot2, 2);

JFreeChart result = new JFreeChart(
    "Combined Range Category Plot Demo",
    new Font("SansSerif", Font.BOLD, 12),
    plot,
    true
);
```

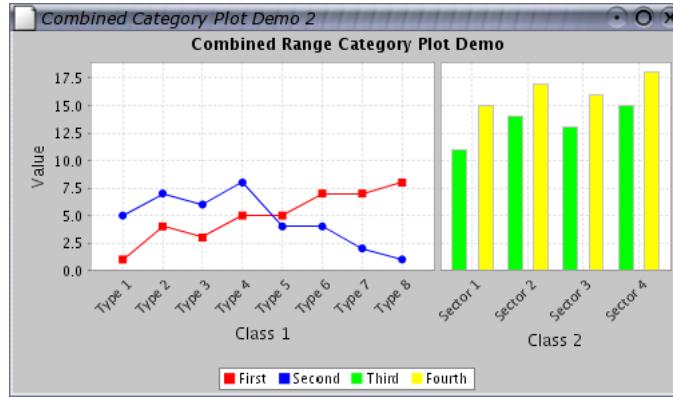


Figure 14.2: A combined range category plot.

Notice how `subplot1` has been added with a weight of 3 (the second argument in the `add()` method), while `subplot2` has been added with a weight of 2. This controls the amount of space allocated to each plot.

The subplots are regular `CategoryPlot` instances that have had their range axis set to `null`. For example, in the demo application the following code is used (it includes some customisation of the subplots):

```
CategoryDataset dataset1 = createDataset1();
CategoryAxis domainAxis1 = new CategoryAxis("Class 1");
domainAxis1.setCategoryLabelPositions(CategoryLabelPositions.UP_45);
domainAxis1.setMaxCategoryLabelWidthRatio(5.0f);
LineAndShapeRenderer renderer1 = new LineAndShapeRenderer();
renderer1.setBaseToolTipGenerator(new StandardCategoryToolTipGenerator());
CategoryPlot subplot1 = new CategoryPlot(dataset1, domainAxis1, null, renderer1);
subplot1.setDomainGridlinesVisible(true);

CategoryDataset dataset2 = createDataset2();
CategoryAxis domainAxis2 = new CategoryAxis("Class 2");
domainAxis2.setCategoryLabelPositions(CategoryLabelPositions.UP_45);
domainAxis2.setMaxCategoryLabelWidthRatio(5.0f);
BarRenderer renderer2 = new BarRenderer();
renderer2.setBaseToolTipGenerator(new StandardCategoryToolTipGenerator());
CategoryPlot subplot2 = new CategoryPlot(dataset2, domainAxis2, null, renderer2);
subplot2.setDomainGridlinesVisible(true);
```

## 14.4 Combined Domain XY Plot

### 14.4.1 Overview

A *combined domain XY plot* is a plot that displays two or more subplots (instances of `XYPlot`) that share a common *domain axis*. Each subplot maintains its own *range axis*. An example is shown in figure 14.3.

It is possible to display this chart with a horizontal or vertical orientation (the example shown has a vertical orientation).

### 14.4.2 Constructing the Chart

A demo application (`CombinedXYPlotDemo1.java`, available for download with the JFreeChart Developer Guide) provides an example of how to create this type of chart. The key step is the creation of a `CombinedDomainXYPlot` instance, to which subplots are added:

```
CombinedDomainXYPlot plot = new CombinedDomainXYPlot(new NumberAxis("Domain"));
plot.setGap(10.0);
```

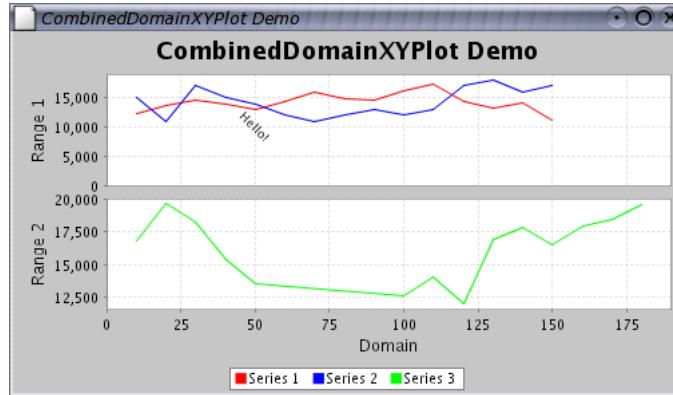


Figure 14.3: A combined domain XY plot

```

plot.add(subplot1, 1);
plot.add(subplot2, 1);
plot.setOrientation(PlotOrientation.VERTICAL);

return new JFreeChart(
    "CombinedDomainXYPlot Demo",
    JFreeChart.DEFAULT_TITLE_FONT, plot, true
);

```

Notice how the subplots are added with weights (both 1 in this case). This controls the amount of space allocated to each plot.

The subplots are regular `XYPlot` instances that have had their domain axis set to `null`. For example, in the demo application the following code is used (it includes some customisation of the subplots):

```

XYDataset data1 = createDataset1();
XYItemRenderer renderer1 = new StandardXYItemRenderer();
NumberAxis rangeAxis1 = new NumberAxis("Range 1");
XYPlot subplot1 = new XYPlot(data1, null, rangeAxis1, renderer1);
subplot1.setRangeAxisLocation(AxisLocation.BOTTOM_OR_LEFT);

XYTextAnnotation annotation = new XYTextAnnotation("Hello!", 50.0, 10000.0);
annotation.setFont(new Font("SansSerif", Font.PLAIN, 9));
annotation.setRotationAngle(Math.PI / 4.0);
subplot1.addAnnotation(annotation);

// create subplot 2...
XYDataset data2 = createDataset2();
XYItemRenderer renderer2 = new StandardXYItemRenderer();
NumberAxis rangeAxis2 = new NumberAxis("Range 2");
rangeAxis2.setAutoRangeIncludesZero(false);
XYPlot subplot2 = new XYPlot(data2, null, rangeAxis2, renderer2);
subplot2.setRangeAxisLocation(AxisLocation.TOP_OR_LEFT);

```

## 14.5 Combined Range XY Plot

### 14.5.1 Overview

A *combined range XY plot* is a plot that displays two or more subplots (instances of `XYPlot`) that share a common *range axis*. Each subplot maintains its own *domain axis*. An example is shown in figure 14.4.

It is possible to display this chart with a horizontal or vertical orientation (the example shown has a vertical orientation).

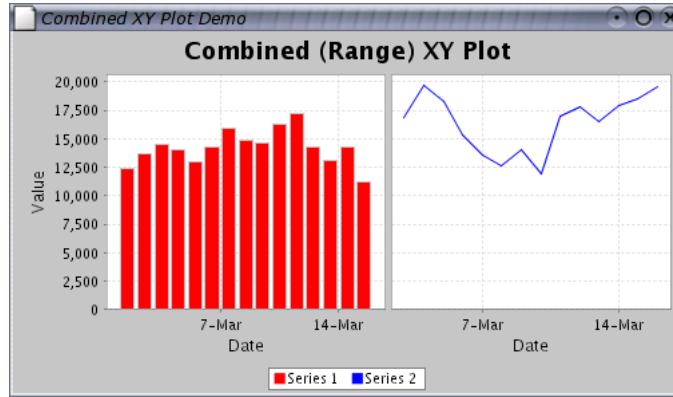


Figure 14.4: A combined range XY plot

### 14.5.2 Constructing the Chart

A demo application (`CombinedXYPlotDemo2.java`, available for download with the JFreeChart Developer Guide) provides an example of how to create this type of chart. The key step is the creation of a `CombinedRangeXYPlot` instance, to which subplots are added:

```
// create the plot...
CombinedRangeXYPlot plot = new CombinedRangeXYPlot(new NumberAxis("Value"));
plot.add(subplot1, 1);
plot.add(subplot2, 1);

return new JFreeChart(
    "Combined (Range) XY Plot",
    JFreeChart.DEFAULT_TITLE_FONT, plot, true
);
```

Notice how the subplots are added with weights (both 1 in this case). This controls the amount of space allocated to each plot.

The subplots are regular `XYPlot` instances that have had their range axis set to `null`. For example, in the demo application the following code is used (it includes some customisation of the subplots):

```
// create subplot 1...
IntervalXYDataset data1 = createDataset1();
XYItemRenderer renderer1 = new XYBarRenderer(0.20);
renderer1.setToolTipGenerator(
    new StandardXYToolTipGenerator(
        new SimpleDateFormat("d-MMM-yyyy"), new DecimalFormat("0,000.0")
    )
);
XYPlot subplot1 = new XYPlot(data1, new DateAxis("Date"), null, renderer1);

// create subplot 2...
XYDataset data2 = createDataset2();
XYItemRenderer renderer2 = new StandardXYItemRenderer();
renderer2.setToolTipGenerator(
    new StandardXYToolTipGenerator(
        new SimpleDateFormat("d-MMM-yyyy"), new DecimalFormat("0,000.0")
    )
);
XYPlot subplot2 = new XYPlot(data2, new DateAxis("Date"), null, renderer2);
```

# Chapter 15

## Datasets and JDBC

### 15.1 Introduction

In this section, I describe the use of several *datasets* that are designed to work with JDBC to obtain data from database tables:

- [JDBCPieDataset](#)
- [JDBCCategoryDataset](#)
- [JDBCXYDataset](#)

These datasets have been developed by Bryan Scott of the Australian Antarctic Division.

### 15.2 About JDBC

JDBC is a high-level Java API for working with relational databases. JDBC does a good job of furthering Java's *platform independence*, making it possible to write portable code that will work with many different database systems.

JDBC provides a mechanism for loading a *JDBC driver* specific to the database system actually being used. JDBC drivers are available for many databases, on many different platforms.

### 15.3 Sample Data

To see the JDBC datasets in action, you need to create some sample data in a test database.

Here is listed some sample data that will be used to create a pie chart, a bar chart and a time series chart.

A pie chart will be created using this data (in a table called `piedata1`):

CATEGORY	VALUE
London	54.3
New York	43.4
Paris	17.9

Similarly, a bar chart will be created using this data (in a table called `categorydata1`):

CATEGORY	SERIES1	SERIES2	SERIES3
London	54.3	32.1	53.4
New York	43.4	54.3	75.2
Paris	17.9	34.8	37.1

Finally, a time series chart will be generated using this data (in a table called `xydata1`):

X	SERIES1	SERIES2	SERIES3
1-Aug-2002	54.3	32.1	53.4
2-Aug-2002	43.4	54.3	75.2
3-Aug-2002	39.6	55.9	37.1
4-Aug-2002	35.4	55.2	27.5
5-Aug-2002	33.9	49.8	22.3
6-Aug-2002	35.2	48.4	17.7
7-Aug-2002	38.9	49.7	15.3
8-Aug-2002	36.3	44.4	12.1
9-Aug-2002	31.0	46.3	11.0

You should set up a test database containing these tables...ask your database administrator to help you if necessary. I've called my test database `jfreechartdb`, but you can change the name if you want to.

In the next section I document the steps I used to set up this sample data using *PostgreSQL*, the database system that I have available for testing purposes. If you are using a different system, you may need to perform a slightly different procedure—refer to your database documentation for information.

## 15.4 PostgreSQL

### 15.4.1 About PostgreSQL

*PostgreSQL* is a powerful object-relational database server, distributed under an open-source licence. You can find out more about PostgreSQL at:

<http://www.postgresql.org>

Note: although PostgreSQL is free, it has most of the features of large commercial relational database systems. I encourage you to install it and try it out.

### 15.4.2 Creating a New Database

First, while logged in as the database administrator, I create a test database called `jfreechartdb`:

```
CREATE DATABASE jfreechartdb;
```

Next, I create a user `jfreechart`:

```
CREATE USER jfreechart WITH PASSWORD 'password';
```

This username and password will be used to connect to the database via JDBC.

### 15.4.3 Creating the Pie Chart Data

To create the table for the pie dataset:

```
CREATE TABLE piedata1 (
    category VARCHAR(32),
    value      FLOAT
);
```

...and to populate it:

```
INSERT INTO piedata1 VALUES ('London', 54.3);
INSERT INTO piedata1 VALUES ('New York', 43.4);
INSERT INTO piedata1 VALUES ('Paris', 17.9);
```

#### 15.4.4 Creating the Category Chart Data

To create the table for the category dataset:

```
CREATE TABLE categorydata1 (
    category VARCHAR(32),
    series1 FLOAT,
    series2 FLOAT,
    series3 FLOAT
);
```

...and to populate it:

```
INSERT INTO categorydata1 VALUES ('London', 54.3, 32.1, 53.4);
INSERT INTO categorydata1 VALUES ('New York', 43.4, 54.3, 75.2);
INSERT INTO categorydata1 VALUES ('Paris', 17.9, 34.8, 37.1);
```

#### 15.4.5 Creating the XY Chart Data

To create the table for the XY dataset:

```
CREATE TABLE xydata1 (
    date DATE,
    series1 FLOAT,
    series2 FLOAT,
    series3 FLOAT
);
```

...and to populate it:

```
INSERT INTO xydata1 VALUES ('1-Aug-2002', 54.3, 32.1, 53.4);
INSERT INTO xydata1 VALUES ('2-Aug-2002', 43.4, 54.3, 75.2);
INSERT INTO xydata1 VALUES ('3-Aug-2002', 39.6, 55.9, 37.1);
INSERT INTO xydata1 VALUES ('4-Aug-2002', 35.4, 55.2, 27.5);
INSERT INTO xydata1 VALUES ('5-Aug-2002', 33.9, 49.8, 22.3);
INSERT INTO xydata1 VALUES ('6-Aug-2002', 35.2, 48.4, 17.7);
INSERT INTO xydata1 VALUES ('7-Aug-2002', 38.9, 49.7, 15.3);
INSERT INTO xydata1 VALUES ('8-Aug-2002', 36.3, 44.4, 12.1);
INSERT INTO xydata1 VALUES ('9-Aug-2002', 31.0, 46.3, 11.0);
```

#### Granting Table Permissions

The last step in setting up the sample database is to grant read access to the new tables to the user `jfreechart`:

```
GRANT SELECT ON piedata1 TO jfreechart;
GRANT SELECT ON categorydata1 TO jfreechart;
GRANT SELECT ON xydata1 TO jfreechart;
```

### 15.5 The JDBC Driver

To access the sample data via JDBC, you need to obtain a JDBC driver for your database. For PostgreSQL, I downloaded a free driver from:

<http://jdbc.postgresql.org>

In order to use this driver, I need to ensure that the jar file containing the driver is on the classpath.

## 15.6 The Demo Applications

### 15.6.1 JDBC Pie Chart Demo

The `JDBC Pie Chart Demo` application will generate a pie chart using the data in the `piedata1` table, providing that you have configured your database correctly.

The code for reading the data is in the `readData()` method:

```
private PieDataset readData() {
    JDBC PieDataset data = null;
    String url = "jdbc:postgresql://nomad/jfreechartdb";
    Connection con;

    try {
        Class.forName("org.postgresql.Driver");
    }
    catch (ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }

    try {
        con = DriverManager.getConnection(url, "jfreechart", "password");

        data = new JDBC PieDataset(con);
        String sql = "SELECT * FROM PIEDATA1;";
        data.executeQuery(sql);
        con.close();
    }

    catch (SQLException e) {
        System.err.print("SQLException: ");
        System.err.println(e.getMessage());
    }

    catch (Exception e) {
        System.err.print("Exception: ");
        System.err.println(e.getMessage());
    }

    return data;
}
```

Important things to note in the code are:

- the `url` used to reference the test database includes the name of my test server (`nomad`), you will need to modify this;
- a connection is made to the database using the username/password combination `jfreechart/password`;
- the query used to pull the data from the database is a standard `SELECT` query, but you can use any SQL query as long as it returns columns in the required format (refer to the `JDBC Pie Dataset` class documentation for details).

### 15.6.2 JDBC Category Chart Demo

The `JDBC Category Chart Demo` application generates a bar chart using the data in the `categorydata1` table. The code is almost identical to the `JDBC Pie Chart Demo`. Once again, you can use any SQL query as long as it returns columns in the required format (refer to the `JDBC Category Dataset` class documentation for details).

### 15.6.3 JDBC XY Chart Demo

The `JDBC XY Chart Demo` application generates a time series chart using the data in the `xydata1` table. The code is almost identical to the `JDBC Pie Chart Demo`. Once again, you can use any SQL query as long as it returns columns in the required format (refer to the `JDBC XY Dataset` class documentation for details).

# Chapter 16

## Exporting Charts to Acrobat PDF

### 16.1 Introduction

In this section, I describe how to export a chart to an Acrobat PDF file using JFreeChart and iText. Along with the description, I provide a small demonstration application that creates a PDF file containing a basic chart. The resulting file can be viewed using Acrobat Reader, or any other software that is capable of reading and displaying PDF files.

### 16.2 What is Acrobat PDF?

Acrobat PDF is a widely used electronic document format. Its popularity is due, at least in part, to its ability to reproduce high quality output on a variety of different platforms.

PDF was created by Adobe Systems Incorporated. Adobe provide a free (but closed source) application called *Acrobat Reader* for reading PDF documents. Acrobat Reader is available on most end-user computing platforms, including GNU/Linux, Windows, Unix, Macintosh and others.

If your system doesn't have Acrobat Reader installed, you can download a copy from:

<http://www.adobe.com/products/acrobat/readstep.html>

On some platforms, there are free (in the GNU sense) software packages available for viewing PDF files. Ghostview on Linux is one example.

### 16.3 iText

iText is a popular free Java class library for creating documents in PDF format. It is developed by Bruno Lowagie, Paulo Soares and others. The home page for iText is:

<http://www.lowagie.com/iText>

At the time of writing, the latest version of iText is 2.0.1.

### 16.4 Graphics2D

JFreeChart can work easily with iText because iText provides a `Graphics2D` implementation. Before I proceed to the demonstration application, I will briefly review the `Graphics2D` class.

The `java.awt.Graphics2D` class, part of the standard Java 2D API, defines a range of methods for drawing text and graphics in a two dimensional space. Particular subclasses of `Graphics2D` handle all the details of mapping the output (text and graphics) to specific devices.

JFreeChart has been designed to draw charts using only the methods defined by the `Graphics2D` class. This means that JFreeChart can generate output to any target that can provide a `Graphics2D` subclass.

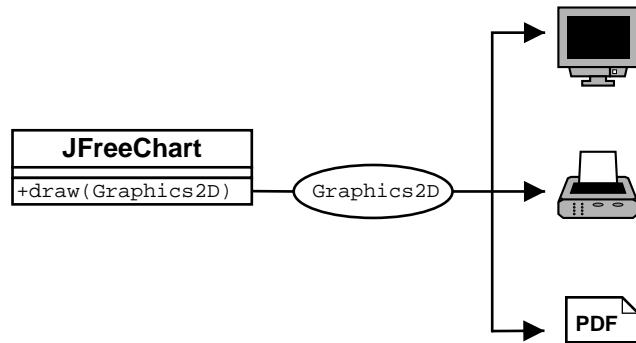


Figure 16.1: The JFreeChart `draw()` method

iText incorporates a `PdfGraphics2D` class, which means that iText is capable of generating PDF content based on calls to the methods defined by the `Graphics2D` class...and this makes it easy to produce charts in PDF format, as you will see in the following sections.

## 16.5 Getting Started

To compile and run the demonstration application, you will need the following jar files:

File:	Description:
<code>jfreechart-1.0.9.jar</code>	The JFreeChart class library.
<code>jcommon-1.0.12.jar</code>	The JCommon class library (used by JFreeChart).
<code>itext-2.0.1.jar</code>	The iText class library.

The first two files are included with JFreeChart, and the third is the iText runtime.

## 16.6 The Application

The first thing the sample application needs to do is create a chart. Here we create a time series chart:

```

// create a chart...
XYDataset dataset = createDataset();
JFreeChart chart = ChartFactory.createTimeSeriesChart(
    "Legal & General Unit Trust Prices",
    "Date",
    "Price Per Unit",
    dataset,
    true,
    true,
    false
);
// some additional chart customisation here...
  
```

There is nothing special here—in fact you could replace the code above with any other code that creates a `JFreeChart` object. You are encouraged to experiment.

Next, I will save a copy of the chart in a PDF file:

```

// write the chart to a PDF file...
File fileName = new File(System.getProperty("user.home") + "/jfreechart1.pdf");
saveChartAsPDF(fileName, chart, 400, 300, new DefaultFontMapper());
  
```

There are a couple of things to note here.

First, I have hard-coded the filename used for the PDF file. I've done this to keep the sample code short. In a real application, you would provide some other means for the user to specify the filename, perhaps by presenting a file chooser dialog.

Second, the `saveChartAsPDF()` method hasn't been implemented yet! To create that method, I'll first write another more general method, `writeChartAsPDF()`. This method performs most of the work that will be required by the `saveChartAsPDF()` method, but it writes data to an *output stream* rather than a file.

```
public static void writeChartAsPDF(OutputStream out,
                                  JFreeChart chart,
                                  int width,
                                  int height,
                                  FontMapper mapper) throws IOException {

    Rectangle pagesize = new Rectangle(width, height);
    Document document = new Document(pagesize, 50, 50, 50, 50);
    try {
        PdfWriter writer = PdfWriter.getInstance(document, out);
        document.addAuthor("JFreeChart");
        document.addSubject("Demonstration");
        document.open();
        PdfContentByte cb = writer.getDirectContent();
        PdfTemplate tp = cb.createTemplate(width, height);
        Graphics2D g2 = tp.createGraphics(width, height, mapper);
        Rectangle2D r2D = new Rectangle2D.Double(0, 0, width, height);
        chart.draw(g2, r2D);
        g2.dispose();
        cb.addTemplate(tp, 0, 0);
    }
    catch (DocumentException de) {
        System.err.println(de.getMessage());
    }
    document.close();
}
```

Inside this method, you will see some code that sets up and opens an iText document, obtains a `Graphics2D` instance from the document, draws the chart using the `Graphics2D` object, and closes the document.

You will also notice that one of the parameters for this method is a `FontMapper` object. The `FontMapper` interface maps Java `Font` objects to the `BaseFont` objects used by iText.

The `DefaultFontMapper` class is predefined with default mappings for the Java *logical fonts*. If you use only these fonts, then it is enough to create a `DefaultFontMapper` using the default constructor. If you want to use other fonts (for example, a font that supports a particular character set) then you need to do more work. I'll give an example of this later.

In the implementation of the `writeChartAsPDF()` method, I've chosen to create a PDF document with a custom page size (matching the requested size of the chart). You can easily adapt the code to use a different page size, alter the size and position of the chart and even draw multiple charts inside one PDF document.

Now that I have a method to send PDF data to an output stream, it is straightforward to implement the `saveChartAsPDF()` method. Simply create a `FileOutputStream` and pass it on to the `writeChartAsPDF()` method:

```
public static void saveChartAsPDF(File file,
                                  JFreeChart chart,
                                  int width,
                                  int height,
                                  FontMapper mapper) throws IOException {

    OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
    writeChartAsPDF(out, chart, width, height, mapper);
    out.close();
}
```

This is all the code that is required. The pieces can be assembled into the following program (reproduced in full here so that you can see all the required import statements and the context in which the code is run):

```

        int width,
        int height,
        FontMapper mapper) throws IOException {

    Rectangle pagesize = new Rectangle(width, height);
    Document document = new Document(pagesize, 50, 50, 50, 50);
    try {
        PdfWriter writer = PdfWriter.getInstance(document, out);
        document.addAuthor("JFreeChart");
        document.addSubject("Demonstration");
        document.open();
        PdfContentByte cb = writer.getDirectContent();
        PdfTemplate tp = cb.createTemplate(width, height);
        Graphics2D g2 = tp.createGraphics(width, height, mapper);
        Rectangle2D r2D = new Rectangle2D.Double(0, 0, width, height);
        chart.draw(g2, r2D);
        g2.dispose();
        cb.addTemplate(tp, 0, 0);
    }
    catch (DocumentException de) {
        System.err.println(de.getMessage());
    }
    document.close();
}

/**
 * Creates a dataset, consisting of two series of monthly data. *
 */
* @return the dataset.
*/
public static XYDataset createDataset() {

    TimeSeries s1 = new TimeSeries("L&G European Index Trust", Month.class);
    s1.add(new Month(2, 2001), 181.8);
    s1.add(new Month(3, 2001), 167.3);
    s1.add(new Month(4, 2001), 153.8);
    s1.add(new Month(5, 2001), 167.6);
    s1.add(new Month(6, 2001), 158.8);
    s1.add(new Month(7, 2001), 148.3);
    s1.add(new Month(8, 2001), 153.9);
    s1.add(new Month(9, 2001), 142.7);
    s1.add(new Month(10, 2001), 123.2);
    s1.add(new Month(11, 2001), 131.8);
    s1.add(new Month(12, 2001), 139.6);
    s1.add(new Month(1, 2002), 142.9);
    s1.add(new Month(2, 2002), 138.7);
    s1.add(new Month(3, 2002), 137.3);
    s1.add(new Month(4, 2002), 143.9);
    s1.add(new Month(5, 2002), 139.8);
    s1.add(new Month(6, 2002), 137.0);
    s1.add(new Month(7, 2002), 132.8);

    TimeSeries s2 = new TimeSeries("L&G UK Index Trust", Month.class);
    s2.add(new Month(2, 2001), 129.6);
    s2.add(new Month(3, 2001), 123.2);
    s2.add(new Month(4, 2001), 117.2);
    s2.add(new Month(5, 2001), 124.1);
    s2.add(new Month(6, 2001), 122.6);
    s2.add(new Month(7, 2001), 119.2);
    s2.add(new Month(8, 2001), 116.5);
    s2.add(new Month(9, 2001), 112.7);
    s2.add(new Month(10, 2001), 101.5);
    s2.add(new Month(11, 2001), 106.1);
    s2.add(new Month(12, 2001), 110.3);
    s2.add(new Month(1, 2002), 111.7);
    s2.add(new Month(2, 2002), 111.0);
    s2.add(new Month(3, 2002), 109.6);
    s2.add(new Month(4, 2002), 113.2);
    s2.add(new Month(5, 2002), 111.6);
    s2.add(new Month(6, 2002), 108.8);
    s2.add(new Month(7, 2002), 101.6);

    TimeSeriesCollection dataset = new TimeSeriesCollection();
    dataset.addSeries(s1);
    dataset.addSeries(s2);

    return dataset;
}

public static void main(String[] args) {
    try {

```

```

// create a chart...
XYDataset dataset = createDataset();
JFreeChart chart = ChartFactory.createTimeSeriesChart(
    "Legal & General Unit Trust Prices",
    "Date",
    "Price Per Unit",
    dataset,
    true,
    true,
    false
);

// some additional chart customisation here...
XYPlot plot = chart.getXYPlot();
XYLineAndShapeRenderer renderer
    = (XYLineAndShapeRenderer) plot.getRenderer();
renderer.setShapesVisible(true);
DateAxis axis = (DateAxis) plot.getDomainAxis();
axis.setDateFormatOverride(new SimpleDateFormat("MMM-yyyy"));

// write the chart to a PDF file...
File fileName = new File(System.getProperty("user.home")
    + "/jfreetechart1.pdf");
saveChartAsPDF(fileName, chart, 400, 300, new DefaultFontMapper());
}

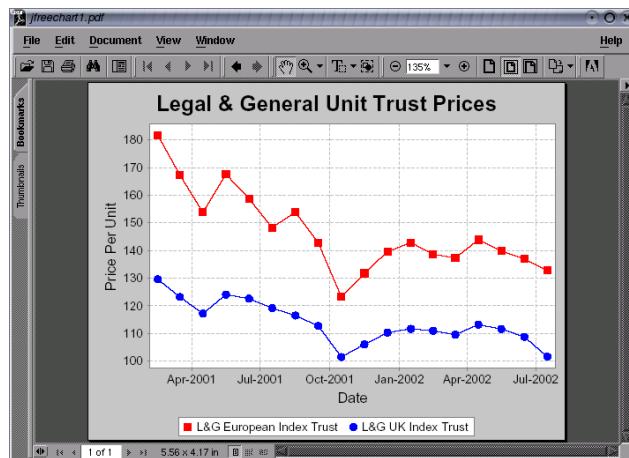
catch (IOException e) {
    System.out.println(e.getMessage());
}
}
}

```

Before you compile and run the application, remember to change the file name used for the PDF file to something appropriate for your system! And include the jar files listed in section [16.5](#) on your classpath.

## 16.7 Viewing the PDF File

After compiling and running the sample application, you can view the resulting PDF file using a PDF viewer like Acrobat Reader (or, in my case, Gnome PDF Viewer):



Most PDF viewer applications provide zooming features that allow you to get a close up view of your charts.

## 16.8 Unicode Characters

It is possible to use the full range of Unicode characters in JFreeChart and iText, as long as you are careful about which fonts you use. In this section, I present some modifications to the previous example to show how to do this.

### 16.8.1 Background

Internally, Java uses the Unicode character encoding to represent text strings. This encoding uses sixteen bits per character, which means there are potentially 65,536 different characters available (the Unicode standard defines something like 38,000 characters).

You can use any of these characters in both JFreeChart and iText, subject to one proviso: *the font you use to display the text must define the characters used or you will not be able to see them.*

Many fonts are not designed to display the entire Unicode character set. The following website contains useful information about fonts that do support Unicode (at least to some extent):

```
http://www.slovo.info/unifonts.htm
```

I have tried out the `tahoma.ttf` font with success. In fact, I will use this font in the example that follows. The Tahoma font doesn't support every character defined in Unicode, so if you have specific requirements then you need to choose an appropriate font. At one point I had the Arial Unicode MS font (`arialuni.ttf`) installed on my system—this has support for the full Unicode character set, although this means that the font definition file is quite large (around 24 megabytes!)

### 16.8.2 Fonts, iText and Java

iText has to handle fonts according to the PDF specification. This deals with document portability by allowing fonts to be (optionally) embedded in a PDF file. This requires access to the font definition file.

Java, on the other hand, abstracts away some of the details of particular font formats with the use of the `Font` class.

To support the `Graphics2D` implementation in iText, it is necessary to map `Font` objects from Java to `BaseFont` objects in iText. This is the role of the `FontMapper` interface.

If you create a new `DefaultFontMapper` instance using the default constructor, it will already contain sensible mappings for the logical fonts defined by the Java specification. But if you want to use additional fonts—and you must if you want to use a wide range of Unicode characters—then you need to add extra mappings to the `DefaultFontMapper` object.

### 16.8.3 Mapping Additional Fonts

I've decided to use the Tahoma font to display a chart title that incorporates some Unicode characters. The font definition file (`tahoma.ttf`) is located, on my system, in the directory:

```
/opt/sun-jdk-1.4.2.08/jre/lib/fonts
```

Here's the code used to create the `FontMapper` for use by iText—I've based this on an example written by Paulo Soares:

```
DefaultFontMapper mapper = new DefaultFontMapper();
mapper.insertDirectory("/opt/sun-jdk-1.4.2.08/jre/lib/fonts");
DefaultFontMapper.BaseFontParameters pp =
    mapper.getBaseFontParameters("Tahoma");
if (pp!=null) {
    pp.encoding = BaseFont.IDENTITY_H;
}
```

Now I can modify the code that creates the chart, in order to add a custom title to the chart (I've changed the data and chart type also):

```
// create a chart...
TimeSeries series = new TimeSeries("Random Data");
Day current = new Day(1, 1, 2000);
double value = 100.0;
for (int i = 0; i < 1000; i++) {
    try {
        value = value + Math.random() - 0.5;
```

```

        series.add(current, new Double(value));
        current = (Day) current.next();
    }
    catch (SeriesException e) {
        System.err.println("Error adding to series");
    }
}
XYDataset data = new TimeSeriesCollection(series);
JFreeChart chart = ChartFactory.createTimeSeriesChart(
    "Test",
    "Date",
    "Value",
    data,
    true,
    false,
    false
);

// Unicode test...
String text = "\u278A\u20A0\u20A1\u20A2\u20A3\u20A4\u20A5\u20A6\u20A7\u20A8\u20A9";
//String text = "hi";
Font font = new Font("Tahoma", Font.PLAIN, 12);
TextTitle subtitle = new TextTitle(text, font);
chart.addSubtitle(subtitle);

```

Notice that the subtitle (a random collection of currency symbols) is defined using escape sequences to specify each Unicode character. This avoids any problems with encoding conversions when I save the Java source file.

The output from the modified sample program is shown in figure 16.2. The example has been embedded in this document in PDF format, so it is a good example of the type of output you can expect by following the instructions in this document.

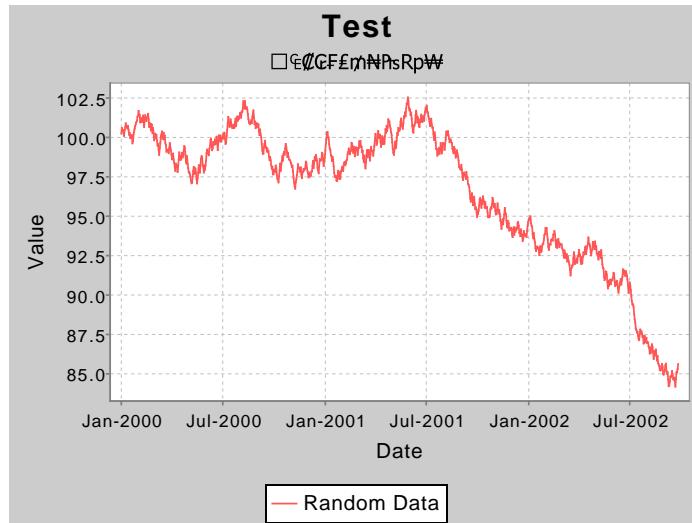


Figure 16.2: A Unicode subtitle

# Chapter 17

## Exporting Charts to SVG Format

### 17.1 Introduction

In this section, I present an example that shows how to export charts to SVG format, using JFreeChart and Batik (an open source library for working with SVG).

### 17.2 Background

#### 17.2.1 What is SVG?

Scalable Vector Graphics (SVG) is a standard language for describing two-dimensional graphics in XML format. It is a *Recommendation* of the World Wide Web Consortium (W3C).

#### 17.2.2 Batik

Batik is an open source toolkit, written in Java, that allows you to generate SVG content. Batik is available from:

<http://xml.apache.org/batik>

At the time of writing, the latest *stable* version of Batik is 1.6.

### 17.3 A Sample Application

#### 17.3.1 JFreeChart and Batik

JFreeChart and Batik can work together relatively easily because:

- JFreeChart draws all chart output using Java's `Graphics2D` abstraction; and
- Batik provides a concrete implementation of `Graphics2D` that generates SVG output (`SVGGraphics2D`).

In this section, a simple example is presented to get you started using JFreeChart and Batik. The example is based on the technique described here:

<http://xml.apache.org/batik/svggen.html>

### 17.3.2 Getting Started

First, you should download Batik and install it according to the instructions provided on the Batik web page.

To compile and run the sample program presented in the next section, you need to ensure that the following jar files are on your classpath:

File:	Description:
jcommon-1.0.12.jar	Common classes from JFree.
jfreechart-1.0.9.jar	The JFreeChart class library.
batik-awt-util.jar	Batik runtime files.
batik-dom.jar	Batik runtime files.
batik-svggen.jar	Batik runtime files.
batik-util.jar	Batik runtime files.

### 17.3.3 The Application

Create a project in your favourite Java development environment, add the libraries listed in the previous section, and type in the following program (or easier, grab a copy of the source from the JFreeChart demo collection):

```

/*
 * -----
 * SVGExportDemo.java
 * -----
 * (C) Copyright 2002-2005, by Object Refinery Limited.
 *
 */

package demo.svg;

import java.awt.geom.Rectangle2D;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;

import org.apache.batik.dom.GenericDOMImplementation;
import org.apache.batik.svggen.SVGGraphics2D;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;
import org.w3c.dom.DOMImplementation;
import org.w3c.dom.Document;

/**
 * A demonstration showing the export of a chart to SVG format.
 */
public class SVGExportDemo {

    /**
     * Starting point for the demo.
     *
     * @param args ignored.
     */
    public static void main(String[] args) throws IOException {

        // create a dataset...
        DefaultPieDataset data = new DefaultPieDataset();
        data.setValue("Category 1", new Double(43.2));
        data.setValue("Category 2", new Double(27.9));
        data.setValue("Category 3", new Double(79.5));

        // create a chart
        JFreeChart chart = ChartFactory.createPieChart(
            "Sample Pie Chart",
            data,
            true,
            false,
            false
        );

        // THE FOLLOWING CODE BASED ON THE EXAMPLE IN THE BATIK DOCUMENTATION...
        // Get a DOMImplementation
    }
}

```

```

DOMImplementation domImpl
    = GenericDOMImplementation.getDOMImplementation();

// Create an instance of org.w3c.dom.Document
Document document = domImpl.createDocument(null, "svg", null);

// Create an instance of the SVG Generator
SVGGraphics2D svgGenerator = new SVGGraphics2D(document);

// set the precision to avoid a null pointer exception in Batik 1.5
svgGenerator.getGeneratorContext().setPrecision(6);

// Ask the chart to render into the SVG Graphics2D implementation
chart.draw(svgGenerator, new Rectangle2D.Double(0, 0, 400, 300), null);

// Finally, stream out SVG to a file using UTF-8 character to
// byte encoding
boolean useCSS = true;
Writer out = new OutputStreamWriter(
    new FileOutputStream(new File("test.svg")), "UTF-8");
svgGenerator.stream(out, useCSS);

}
}

```

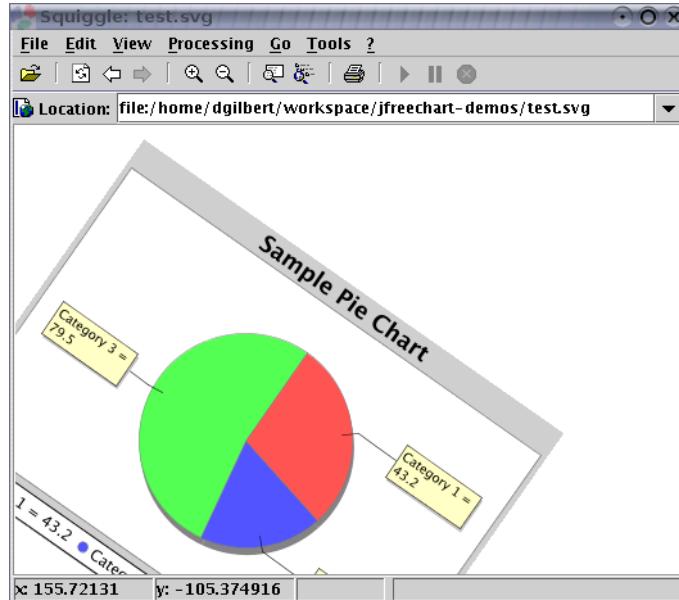
Running this program creates a file `test.svg` in SVG format.

#### 17.3.4 Viewing the SVG

Batik includes a viewer application (“Squiggle”) which you can use to open and view the SVG file. The Batik download includes instructions for running the viewer, effectively all you require is:

```
java -jar batik-squiggle.jar
```

The following screen shot shows the pie chart that we created earlier, displayed using the browser application. A transformation (rotation) has been applied to the chart from within the browser:



If you play about with the viewer, zooming in and out and applying various transformations to the chart, you will begin to appreciate the power of the SVG format.

# Chapter 18

## Applets

### 18.1 Introduction

Subject to a couple of provisos, using JFreeChart in an applet is relatively straightforward. This section provides a brief overview of the important issues and describes a working example that should be sufficient to get you started.

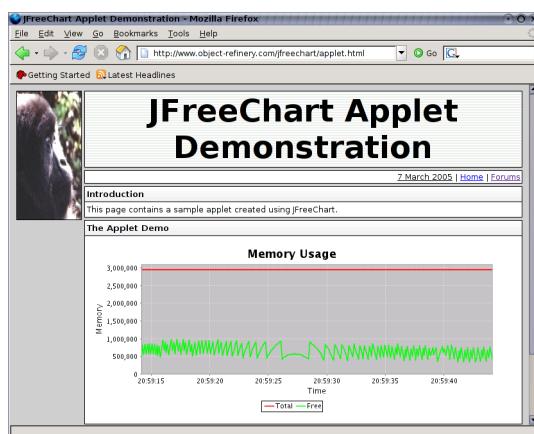


Figure 18.1: An applet using JFreeChart

Figure 18.1 shows a sample applet that uses JFreeChart. This applet is available online at:

<http://www.object-refinery.com/jfreechart/applet.html>

The source code for this applet appears later in this section.

### 18.2 Issues

The main issues to consider when developing applets (whether with or without JFreeChart) are:

- browser support;
- security restrictions;
- code size.

Be sure that you understand these issues *before* you commit significant resources to writing applets.

### 18.2.1 Browser Support

The *vast majority* of web browsers provide support for the latest version of Java (JDK 1.5.0) and will therefore have no problems running applets that use JFreeChart (recall that JFreeChart will run on any version of the JDK from 1.3.1 onwards).

However, the *vast majority* of users on the web use (by default in most cases) the one web browser—Microsoft Internet Explorer (MSIE)—that only supports a version of Java (JDK 1.1) that is now hopelessly out-of-date. This is a problem, because applets that use JFreeChart will not work on a default installation of MSIE. There is a workaround—users can download and install Sun’s Java plugin—but, like many workarounds, it is too much effort and inconvenience for many people. The end result is a deployment problem for developers who choose to write applets.

This single issue has caused many developers to abandon their plans to develop applets<sup>1</sup> and instead choose an easier-to-deploy technology such as *Java Servlets* (see the next chapter).

### 18.2.2 Security

Applets (and Java more generally) have been designed with security in mind. When an applet runs in your web browser, it is restricted in the operations that it is permitted to perform. For example, an applet typically will not be allowed to read or write to the local filesystem. Describing the details of Java’s security mechanism is beyond the scope of this text, but you should be aware that some functions provided by JFreeChart (for example, the option to save charts to PNG format via the pop-up menu) will not work in applets that are subject to the default security policy. If you need these functions to work, then you will need to study Java’s security mechanism in more detail.

### 18.2.3 Code Size

A final issue to consider is the size of the “runtime” code required for your applet. Before an applet can run, the code (typically packed into jar files) has to be downloaded to the end user’s computer. Clearly, for users with limited bandwidth connections, the size of the code can be an issue.

The JFreeChart code is distributed in a jar file that is around 1,000KB in size. That isn’t large—especially when you consider the number and variety of charts that JFreeChart supports—but, at the same time, it isn’t exactly optimal for a user on a dial-up modem connection. And you need to add to that the JCommon jar file (around 290KB) plus whatever code you have for your applet.

As always with JFreeChart, you have the source code so you could improve this by repackaging the JFreeChart jar file to include only those classes that are used by your applet (directly or indirectly).

## 18.3 A Sample Applet

As mentioned in the introduction, a sample applet that uses JFreeChart can be seen at the following URL:<sup>2</sup>

<http://www.object-refinery.com/jfreechart/applet.html>

Two aspects of the sample applet are interesting, the source code that is used to create the applet and the HTML file that is used to invoke the applet.

---

<sup>1</sup>For some people this issue won’t be a concern. For example, you may be developing applets for internal corporate use, and your standard desktop configuration includes a browser that supports JDK 1.5.0. Alternatively, you may be providing an applet for public use via the World Wide Web, but it is not critical that *every* user be able to run the applet.

<sup>2</sup>If the applet does not work for you, please check that your web browser is configured correctly and supports JDK 1.3.1 or later.

### 18.3.1 The HTML

The HTML used to invoke the applet is important, since it needs to reference the necessary jar files. The HTML applet tag used is:

```
<APPLET ARCHIVE="jfreechart-1.0.9-applet-demo.jar,
jfreechart-1.0.9.jar,jcommon-1.0.12.jar"
CODE="demo.applet.Applet1" width=640 height=260
ALT="You should see an applet, not this text.">
</APPLET>
```

Notice that three jar files are referenced. The first contains the applet class (source code in the next section) only, while the remaining two jar files are the standard JFreeChart and JCommon class libraries (the version numbers reflect the age of the demo rather than the current releases).

You can place the applet tag anywhere in your HTML file that you might place some other element (such as an image).

### 18.3.2 The Source Code

The sample applet is created using the following source code (which is included in the “support demos” package). There is very little applet-specific code here—we just extend `JApplet`:

```
/*
 * -----
 * Applet1.java
 * -----
 * (C) Copyright 2002-2005, by Object Refinery Limited.
 */

package demo.applet;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JApplet;
import javax.swing.Timer;

import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.DateAxis;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.time.Millisecond;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;

/**
 * A simple applet demo.
 */
public class Applet1 extends JApplet {

    /** Time series for total memory used. */
    private TimeSeries total;

    /** Time series for free memory. */
    private TimeSeries free;

    /**
     * Creates a new instance.
     */
    public Applet1() {

        // create two series that automatically discard data more than
        // 30 seconds old...
        this.total = new TimeSeries("Total", Millisecond.class);
        this.total.setMaximumItemAge(30000);
        this.free = new TimeSeries("Free", Millisecond.class);
        this.free.setMaximumItemAge(30000);
        TimeSeriesCollection dataset = new TimeSeriesCollection();
        dataset.addSeries(total);
        dataset.addSeries(free);
    }
}
```

```

DateAxis domain = new DateAxis("Time");
NumberAxis range = new NumberAxis("Memory");

XYItemRenderer renderer = new XYLineAndShapeRenderer(true, false);

XYPlot plot = new XYPlot(dataset, domain, range, renderer);
plot.setBackgroundPaint(Color.lightGray);
plot.setDomainGridlinePaint(Color.white);
plot.setRangeGridlinePaint(Color.white);
renderer.setSeriesPaint(0, Color.red);
renderer.setSeriesPaint(1, Color.green);
renderer.setSeriesStroke(0, new BasicStroke(1.5f));
renderer.setSeriesStroke(1, new BasicStroke(1.5f));

domain.setAutoRange(true);
domain.setLowerMargin(0.0);
domain.setUpperMargin(0.0);
domain.setTickLabelsVisible(true);

range.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

JFreeChart chart = new JFreeChart(
    "Memory Usage", JFreeChart.DEFAULT_TITLE_FONT, plot, true
);
chart.setBackgroundPaint(Color.white);
ChartPanel chartPanel = new ChartPanel(chart);
chartPanel.setPopupMenu(null);

getContentPane().add(chartPanel);
new Applet1.DataGenerator().start();

}

/**
 * Adds an observation to the 'total memory' time series.
 *
 * @param y  the total memory used.
 */
private void addTotalObservation(double y) {
    total.add(new Millisecond(), y);
}

/**
 * Adds an observation to the 'free memory' time series.
 *
 * @param y  the free memory.
 */
private void addFreeObservation(double y) {
    free.add(new Millisecond(), y);
}

/**
 * The data generator.
 */
class DataGenerator extends Timer implements ActionListener {

    /**
     * Constructor.
     */
    DataGenerator() {
        super(100, null);
        addActionListener(this);
    }

    /**
     * Adds a new free/total memory reading to the dataset.
     *
     * @param event  the action event.
     */
    public void actionPerformed(ActionEvent event) {
        long f = Runtime.getRuntime().freeMemory();
        long t = Runtime.getRuntime().totalMemory();
        addTotalObservation(t);
        addFreeObservation(f);
    }
}
}

```

# Chapter 19

## Servlets

### 19.1 Introduction

The *Java Servlets API* is a popular technology for creating web applications. JFreeChart is well suited for use in a servlet environment and, in this section, some examples are presented to help those developers that are interested in using JFreeChart for web applications.

All the sample code in this section is available for download from:

<http://www.object-refinery.com/jfreechart/premium/index.html>

The file to download is `jfreechart-1.0.9-demo.zip`.<sup>1</sup>

### 19.2 A Simple Servlet

The `ServletDemo1` class implements a very simple servlet that returns a PNG image of a bar chart generated using JFreeChart. When it is run, the servlet will return a raw image to the client (web browser) which will display the image without any surrounding HTML—see figure 19.1. Typically,

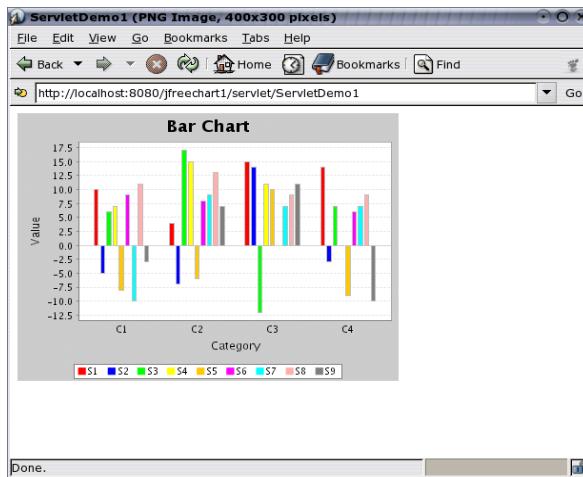


Figure 19.1: `ServletDemo1` in a browser

you will not present raw output in this way, so this servlet is not especially useful on its own, but the example is:

<sup>1</sup>To access this page you need to enter the username and password provided to you in the confirmation e-mail you received when you purchased the JFreeChart Developer Guide.

- a good illustration of the *request-response* nature of servlets;
- useful as a test case if you are configuring a server environment and want to check that everything is working.

We will move on to a more complex example later, showing how to request different charts using HTML forms, and embedding the generated charts within HTML output.

Here is the code for the basic servlet:

```
/*
 * -----
 * ServletDemo1.java
 * -----
 * (C) Copyright 2002-2004, by Object Refinery Limited.
 *
 */

package demo;

import java.io.IOException;
import java.io.OutputStream;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

/**
 * A basic servlet that returns a PNG image file generated by JFreeChart.
 * This class is described in the JFreeChart Developer Guide in the
 * "Servlets" chapter.
 */
public class ServletDemo1 extends HttpServlet {

    /**
     * Creates a new demo.
     */
    public ServletDemo1() {
        // nothing required
    }

    /**
     * Processes a GET request.
     *
     * @param request the request.
     * @param response the response.
     *
     * @throws ServletException if there is a servlet related problem.
     * @throws IOException if there is an I/O problem.
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        OutputStream out = response.getOutputStream();
        try {
            DefaultCategoryDataset dataset = new DefaultCategoryDataset();
            dataset.addValue(10.0, "S1", "C1");
            dataset.addValue(4.0, "S1", "C2");
            dataset.addValue(15.0, "S1", "C3");
            dataset.addValue(14.0, "S1", "C4");
            dataset.addValue(-5.0, "S2", "C1");
            dataset.addValue(-7.0, "S2", "C2");
            dataset.addValue(14.0, "S2", "C3");
            dataset.addValue(-3.0, "S2", "C4");
            dataset.addValue(6.0, "S3", "C1");
            dataset.addValue(17.0, "S3", "C2");
            dataset.addValue(-12.0, "S3", "C3");
            dataset.addValue(7.0, "S3", "C4");
            dataset.addValue(7.0, "S4", "C1");
            dataset.addValue(15.0, "S4", "C2");
            dataset.addValue(11.0, "S4", "C3");
            dataset.addValue(0.0, "S4", "C4");
            dataset.addValue(-8.0, "S5", "C1");
        }
    }
}
```

```

dataset.addValue(-6.0, "S5", "C2");
dataset.addValue(10.0, "S5", "C3");
dataset.addValue(-9.0, "S5", "C4");
dataset.addValue(9.0, "S6", "C1");
dataset.addValue(8.0, "S6", "C2");
dataset.addValue(null, "S6", "C3");
dataset.addValue(6.0, "S6", "C4");
dataset.addValue(-10.0, "S7", "C1");
dataset.addValue(9.0, "S7", "C2");
dataset.addValue(7.0, "S7", "C3");
dataset.addValue(7.0, "S7", "C4");
dataset.addValue(11.0, "S8", "C1");
dataset.addValue(13.0, "S8", "C2");
dataset.addValue(9.0, "S8", "C3");
dataset.addValue(9.0, "S8", "C4");
dataset.addValue(-3.0, "S9", "C1");
dataset.addValue(7.0, "S9", "C2");
dataset.addValue(11.0, "S9", "C3");
dataset.addValue(-10.0, "S9", "C4");

JFreeChart chart = ChartFactory.createBarChart(
    "Bar Chart",
    "Category",
    "Value",
    dataset,
    PlotOrientation.VERTICAL,
    true, true, false
);
response.setContentType("image/png");
ChartUtilities.writeChartAsPNG(out, chart, 400, 300);
}
catch (Exception e) {
    System.err.println(e.toString());
}
finally {
    out.close();
}
}
}

```

The `doGet()` method is called by the servlet engine when a request is made by a client (usually a web browser). In response to the request, the servlet performs several steps:

- an `OutputStream` reference is obtained for returning output to the client;
- a chart is created;
- the *content type* for the response is set to `image/png`. This tells the client what type of data it is receiving;
- a PNG image of the chart is written to the output stream;
- the output stream is closed.

### 19.3 Compiling the Servlet

Note that the classes in the `javax.servlet.*` package (and sub-packages), used by the demo servlet, are not part of the *Java 2 Standard Edition (J2SE)*. In order to compile the above code using J2SE, you will need to obtain a `servlet.jar` file. I've used the one that is redistributed with Tomcat (an open source servlet engine written using Java). You can find out more about Tomcat at:

<http://tomcat.apache.org/>

You will also require the JFreeChart and JCommon jar files to compile the above servlet. Change your working directory to `jfreechart-1.0.9-demo`, then enter the following command (on Windows, you need to change the colons to semi-colons, and the forward slashes to backward slashes):

```
javac -classpath jfreechart-1.0.9.jar:lib/jcommon-1.0.12.jar:lib/servlet.jar
source/demo/ServletDemo1.java
```

This should create a `ServletDemo1.class` file. The next section describes how to deploy this servlet using Tomcat.

## 19.4 Deploying the Servlet

Servlets are deployed in the `webapps` directory provided by your servlet engine. In my case, I am using Tomcat 5.5.15 on Ubuntu Linux 5.10, and the directory is:<sup>2</sup>

```
/home/dgilbert/apache-tomcat-5.5.15/webapps
```

Within the `webapps` directory, create a `jfreechart1` directory to hold the first servlet demo, then create the following structure within the directory:

```
.../jfreechart1/WEB-INF/web.xml
.../jfreechart1/WEB-INF/lib/jfreechart-1.0.9.jar
.../jfreechart1/WEB-INF/lib/jcommon-1.0.12.jar
.../jfreechart1/WEB-INF/classes/demo/ServletDemo1.class
```

You need to create the `web.xml` file—it provides information about the servlet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
  <servlet>
    <servlet-name>
      ServletDemo1
    </servlet-name>
    <servlet-class>
      demo.ServletDemo1
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
    <url-pattern>/servlet/ServletDemo1</url-pattern>
  </servlet-mapping>
</web-app>
```

Once you have all these files in place, restart your servlet engine and type in the following URL using your favourite web browser:

```
http://localhost:8080/jfreechart1/servlet/ServletDemo1
```

If all is well, you will see the chart image displayed in your browser, as shown in figure 19.1.

## 19.5 Embedding Charts in HTML Pages

It is possible to embed a chart image generated by a servlet inside an HTML page (that is generated by another servlet). This is demonstrated by `ServletDemo2`, which is also available in the `jfreechart-1.0.9-demo.zip` file.

`ServletDemo2` processes a request by returning a page of HTML that, in turn, references another servlet (`ServletDemo2ChartGenerator`) that returns a PNG image of a chart. The end result is a chart embedded in an HTML page, as shown in figure 19.2.

Here is the code for `ServletDemo2`:

---

<sup>2</sup>Servlets are portable between different servlet engines, so if you are using a different servlet engine, consult the documentation to find the location of the `webapps` folder.

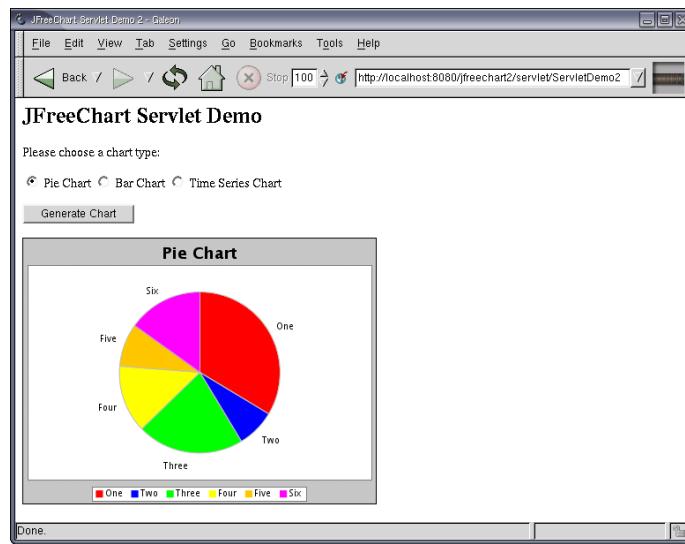


Figure 19.2: ServletDemo2 in a browser

```

/*
 * -----
 * ServletDemo2.java
 * -----
 * (C) Copyright 2002-2004, by Object Refinery Limited.
 *
 */

package demo;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * A basic servlet that generates an HTML page that displays a chart generated by
 * JFreeChart.
 * <P>
 * This servlet uses another servlet (ServletDemo2ChartGenerator) to create a PNG image
 * for the embedded chart.
 * <P>
 * This class is described in the JFreeChart Developer Guide.
 */
public class ServletDemo2 extends HttpServlet {

    /**
     * Creates a new servlet demo.
     */
    public ServletDemo2() {
        // nothing required
    }

    /**
     * Processes a POST request.
     * <P>
     * The chart.html page contains a form for generating the first request, after that
     * the HTML returned by this servlet contains the same form for generating subsequent
     * requests.
     *
     * @param request the request.
     * @param response the response.
     *
     * @throws ServletException if there is a servlet related problem.
     * @throws IOException if there is an I/O problem.
     */
    public void doPost(HttpServletRequest request, HttpServletResponse response)

```

```

throws ServletException, IOException {
    PrintWriter out = new PrintWriter(response.getWriter());
    try {
        String param = request.getParameter("chart");

        response.setContentType("text/html");
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>JFreeChart Servlet Demo 2</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<H2>JFreeChart Servlet Demo</H2>");
        out.println("<P>");
        out.println("Please choose a chart type:");

        out.println("<FORM ACTION=\"ServletDemo2\" METHOD=POST>");
        String pieChecked = (param.equals("pie") ? " CHECKED" : "");
        String barChecked = (param.equals("bar") ? " CHECKED" : "");
        String timeChecked = (param.equals("time") ? " CHECKED" : "");
        out.println("<INPUT TYPE=\"radio\" NAME=\"chart\" VALUE=\"pie\" " + pieChecked
            + "> Pie Chart");
        out.println("<INPUT TYPE=\"radio\" NAME=\"chart\" VALUE=\"bar\" " + barChecked
            + "> Bar Chart");
        out.println("<INPUT TYPE=\"radio\" NAME=\"chart\" VALUE=\"time\" " + timeChecked
            + "> Time Series Chart");
        out.println("<P>");
        out.println("<INPUT TYPE=\"submit\" VALUE=\"Generate Chart\">");
        out.println("</FORM>");

        out.println("<P>");
        out.println("<IMG SRC=\"ServletDemo2ChartGenerator?type=" + param
            + "\" BORDER=1 WIDTH=400 HEIGHT=300/>");
        out.println("</BODY>");
        out.flush();
        out.close();
    }
    catch (Exception e) {
        System.err.println(e.toString());
    }
    finally {
        out.close();
    }
}
}

```

Notice how this code gets a reference to a `Writer` from the `response` parameter, rather than an `OutputStream` as in the previous example. The reason for this is because this servlet will be returning text (HTML), compared to the previous servlet which returned binary data (a PNG image).<sup>3</sup>

The response type is set to `text/html` since this servlet returns HTML text. An important point to note is that the `<IMG>` tag in the HTML references another servlet (`ServletDemo2ChartGenerator`), and this other servlet creates the required chart image. The actual chart returned is controlled by the `chart` parameter, which is set up in the HTML using a `<FORM>` element.

Here is the source code for `ServletDemo2ChartGenerator`:

```

/*
 * -----
 * ServletDemo2ChartGenerator.java
 * -----
 * (C) Copyright 2002-2004, by Object Refinery Limited.
 *
 */

package demo;

import java.io.IOException;
import java.io.OutputStream;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

```

<sup>3</sup>The `Writer` is wrapped in a `PrintWriter` in order to use the more convenient methods available in the latter class.

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.data.time.Day;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.data.xy.XYDataset;
import org.jfree.date.SerialDate;

/**
 * A servlet that returns one of three charts as a PNG image file. This servlet is
 * referenced in the HTML generated by ServletDemo2.
 * <P>
 * Three different charts can be generated, controlled by the 'type' parameter. The possible
 * values are 'pie', 'bar' and 'time' (for time series).
 * <P>
 * This class is described in the JFreeChart Developer Guide.
 */
public class ServletDemo2ChartGenerator extends HttpServlet {

    /**
     * Default constructor.
     */
    public ServletDemo2ChartGenerator() {
        // nothing required
    }

    /**
     * Process a GET request.
     *
     * @param request the request.
     * @param response the response.
     *
     * @throws ServletException if there is a servlet related problem.
     * @throws IOException if there is an I/O problem.
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        OutputStream out = response.getOutputStream();
        try {
            String type = request.getParameter("type");
            JFreeChart chart = null;
            if (type.equals("pie")) {
                chart = createPieChart();
            }
            else if (type.equals("bar")) {
                chart = createBarChart();
            }
            else if (type.equals("time")) {
                chart = createTimeSeriesChart();
            }
            if (chart != null) {
                response.setContentType("image/png");
                ChartUtilities.writeChartAsPNG(out, chart, 400, 300);
            }
        }
        catch (Exception e) {
            System.err.println(e.toString());
        }
        finally {
            out.close();
        }
    }

    /**
     * Creates a sample pie chart.
     *
     * @return a pie chart.
     */
    private JFreeChart createPieChart() {

        // create a dataset...
        DefaultPieDataset data = new DefaultPieDataset();

```

```

        data.setValue("One", new Double(43.2));
        data.setValue("Two", new Double(10.0));
        data.setValue("Three", new Double(27.5));
        data.setValue("Four", new Double(17.5));
        data.setValue("Five", new Double(11.0));
        data.setValue("Six", new Double(19.4));

        JFreeChart chart = ChartFactory.createPieChart(
            "Pie Chart", data, true, true, false
        );
        return chart;
    }

    /**
     * Creates a sample bar chart.
     *
     * @return a bar chart.
     */
    private JFreeChart createBarChart() {

        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.addValue(10.0, "S1", "C1");
        dataset.addValue(4.0, "S1", "C2");
        dataset.addValue(15.0, "S1", "C3");
        dataset.addValue(14.0, "S1", "C4");
        dataset.addValue(-5.0, "S2", "C1");
        dataset.addValue(-7.0, "S2", "C2");
        dataset.addValue(14.0, "S2", "C3");
        dataset.addValue(-3.0, "S2", "C4");
        dataset.addValue(6.0, "S3", "C1");
        dataset.addValue(17.0, "S3", "C2");
        dataset.addValue(-12.0, "S3", "C3");
        dataset.addValue(7.0, "S3", "C4");
        dataset.addValue(7.0, "S4", "C1");
        dataset.addValue(15.0, "S4", "C2");
        dataset.addValue(11.0, "S4", "C3");
        dataset.addValue(0.0, "S4", "C4");
        dataset.addValue(-8.0, "S5", "C1");
        dataset.addValue(-6.0, "S5", "C2");
        dataset.addValue(10.0, "S5", "C3");
        dataset.addValue(-9.0, "S5", "C4");
        dataset.addValue(9.0, "S6", "C1");
        dataset.addValue(8.0, "S6", "C2");
        dataset.addValue(null, "S6", "C3");
        dataset.addValue(6.0, "S6", "C4");
        dataset.addValue(-10.0, "S7", "C1");
        dataset.addValue(9.0, "S7", "C2");
        dataset.addValue(7.0, "S7", "C3");
        dataset.addValue(7.0, "S7", "C4");
        dataset.addValue(11.0, "S8", "C1");
        dataset.addValue(13.0, "S8", "C2");
        dataset.addValue(9.0, "S8", "C3");
        dataset.addValue(9.0, "S8", "C4");
        dataset.addValue(-3.0, "S9", "C1");
        dataset.addValue(7.0, "S9", "C2");
        dataset.addValue(11.0, "S9", "C3");
        dataset.addValue(-10.0, "S9", "C4");

        JFreeChart chart = ChartFactory.createBarChart3D(
            "Bar Chart",
            "Category",
            "Value",
            dataset,
            PlotOrientation.VERTICAL,
            true,
            true,
            false
        );
        return chart;
    }

    /**
     * Creates a sample time series chart.
     *
     * @return a time series chart.
     */
    private JFreeChart createTimeSeriesChart() {
        // here we just populate a series with random data...
    }
}

```

```

TimeSeries series = new TimeSeries("Random Data");
Day current = new Day(1, SerialDate.JANUARY, 2001);
for (int i = 0; i < 100; i++) {
    series.add(current, Math.random() * 100);
    current = (Day) current.next();
}
XYDataset data = new TimeSeriesCollection(series);

JFreeChart chart = ChartFactory.createTimeSeriesChart(
    "Time Series Chart", "Date", "Rate",
    data, true, true, false
);
return chart;
}
}

```

To compile these two servlets, you can enter the following command at the command line:

```

javac -classpath jfreechart-1.0.9.jar:lib/jcommon-1.0.12.jar:lib/servlet.jar
source/demo/ServletDemo2.java source/demo/ServletDemo2ChartGenerator.java

```

The following sections describe the supporting files required for the servlet, and how to deploy them.

## 19.6 Supporting Files

Servlets typically generate output for clients that access the web application via a web browser. Most web applications will include at least one HTML page that is used as the starting point for the application.

For the demo servlets above, the following `index.html` page<sup>4</sup> is used:

```

<HTML>
<HEADER>
<TITLE>JFreeChart : Basic Servlet Demo</TITLE>
</HEADER>

<BODY>
<H2>JFreeChart: Basic Servlet Demo</H2>
<P>
There are two sample servlets available:
<ul>
<li>a very basic servlet to generate a <a href="servlet/ServletDemo1">bar chart</a>;</li>
<li>another servlet that allow you to select one of <a href="chart.html">three sample charts</a>. The selected chart is displayed in an HTML page.</li>
</ul>
</BODY>
</HTML>

```

There are two hyperlinks in this page, the first references the first demo servlet (`ServletDemo1`) and the second references another HTML page, `chart.html`:

```

<HTML>
<HEADER>
<TITLE>JFreeChart Servlet Demo 2</TITLE>
</HEADER>

<BODY>
<H2>JFreeChart Servlet Demo</H2>
<P>
Please choose a chart type:
<FORM ACTION="servlet/ServletDemo2" METHOD=POST>
<INPUT TYPE="radio" NAME="chart" VALUE="pie" CHECKED> Pie Chart
<INPUT TYPE="radio" NAME="chart" VALUE="bar"> Bar Chart

```

---

<sup>4</sup>You'll find this file in the `servlets` directory of the demo distribution, along with the other servlet support files.

```

<INPUT TYPE="radio" NAME="chart" VALUE="time"> Time Series Chart
<P>
<INPUT TYPE="submit" VALUE="Generate Chart">
</FORM>
</BODY>

</HTML>

```

This second HTML page contains a <FORM> element used to specify a parameter for the second servlet (`ServletDemo2`). When this servlet runs, it returns its own HTML that is almost identical to the above but also includes an <IMG> element with a reference to the `ServletDemo2ChartGenerator` servlet.

## 19.7 Deploying Servlets

After compiling the demo servlets, they need to be deployed to a servlet engine, along with the supporting files, so that they can be accessed by clients. Fortunately, this is relatively straightforward.

The first requirement is a `web.xml` file to describe the web application being deployed:

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
  <servlet>
    <servlet-name>
      ServletDemo1
    </servlet-name>
    <servlet-class>
      demo.ServletDemo1
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>
      ServletDemo2
    </servlet-name>
    <servlet-class>
      demo.ServletDemo2
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>
      ServletDemo2ChartGenerator
    </servlet-name>
    <servlet-class>
      demo.ServletDemo2ChartGenerator
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
    <url-pattern>/servlet/ServletDemo1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ServletDemo2</servlet-name>
    <url-pattern>/servlet/ServletDemo2</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ServletDemo2ChartGenerator</servlet-name>
    <url-pattern>/servlet/ServletDemo2ChartGenerator</url-pattern>
  </servlet-mapping>
</web-app>

```

This file lists the servlets by name, and specifies the class file that implements the servlet. The actual class files will be placed in a directory where the servlet engine will know to find them (the `classes` sub-directory within a directory specific to the application).

The final step is copying all the files to the appropriate directory for the servlet engine. In testing with Tomcat, I created a `jfreechart2` directory within Tomcat's `webapps` directory. The `index.html` and `chart.html` files are copied to this directory.

```
webapps/jfreechart2/index.html  
webapps/jfreechart2/chart.html
```

Next, a subdirectory WEB-INF is created within the `jfreechart2` directory, and the `web.xml` file is copied to here.

```
webapps/jfreechart2/WEB-INF/web.xml
```

A `classes` subdirectory is created within WEB-INF to hold the `.class` files for the three demo servlets. These need to be saved in a directory hierarchy matching the package hierarchy:

```
webapps/jfreechart2/WEB-INF/classes/demo/ServletDemo1.class  
webapps/jfreechart2/WEB-INF/classes/demo/ServletDemo2.class  
webapps/jfreechart2/WEB-INF/classes/demo/ServletDemo2ChartGenerator.class
```

Finally, the servlets make use of classes in the JFreeChart and JCommon class libraries. The jar files for these libraries need to be added to a `lib` directory within WEB-INF. You will need:

```
webapps/jfreechart2/WEB-INF/lib/jcommon-1.0.12.jar  
webapps/jfreechart2/WEB-INF/lib/jfreechart-1.0.9.jar
```

Now restart your servlet engine, and point your browser to:

```
http://localhost:8080/jfreechart2/index.html
```

If all the files have been put in the correct places, you should see the running servlet demonstration (this has been tested using Tomcat 5.5.15 running on Ubuntu Linux 5.10 for AMD64).

# Chapter 20

## Miscellaneous

### 20.1 Introduction

This section contains miscellaneous information about JFreeChart.

### 20.2 X11 / Headless Java

If you are using JFreeChart in a server environment running Unix / Linux, you may encounter the problem that JFreeChart won't run without X11. This is a common problem for Java code that relies on AWT, see the following web page for further information:

<http://java.sun.com/products/java-media/2D/forDevelopers/java2dfa.html#xvfb>

There is also a thread in the JFreeChart forum with lots of info:

<http://www.jfree.org/phpBB2/viewtopic.php?t=1012>

### 20.3 Java Server Pages

Developers that are interested in using JFreeChart with JSP will want to check out the Cewolf project:

<http://cewolf.sourceforge.net/>

Thanks to Guido Laures for leading this effort.

### 20.4 Loading Images

Images in Java are represented by the `Image` class. You can load an image using the `createImage()` method in the `Toolkit` class, but you need to be aware that this method loads the image asynchronously—in other words, the method returns immediately (before the image is loaded) and the image loading continues in a separate thread. This can cause problems if you use the image without first waiting for it to complete loading.

You can use the `MediaTracker` class to check the progress of an image as it loads. But in the case where you just want to ensure that you have a fully loaded image, a useful technique is to use Swing's `ImageIcon` class to do the image loading for you:

```
ImageIcon icon = new ImageIcon("/home/dgilbert/temp/daylight.png");
Image image = icon.getImage();
```

In this case, the constructor doesn't return until the image is fully loaded, so by the time you call the `getImage()` method, you know that the image loading is complete.

# Chapter 21

## Packages

### 21.1 Overview

The following sections contain reference information for the classes, arranged by package, that make up the JFreeChart class library.

Package:	Description:
<code>org.jfree.chart</code>	The main chart classes.
<code>org.jfree.chart.annotations</code>	A simple framework for annotating charts.
<code>org.jfree.chart.axis</code>	Axis classes and related interfaces.
<code>org.jfree.chart.editor</code>	A framework (incomplete) for providing property editors for charts.
<code>org.jfree.chart.encoders</code>	Classes for writing image files.
<code>org.jfree.chart.entity</code>	Classes representing chart entities.
<code>org.jfree.chart.event</code>	The event classes.
<code>org.jfree.chart.imagemap</code>	HTML image map utility classes.
<code>org.jfree.chart.labels</code>	The item label and tooltip classes.
<code>org.jfree.chart.needle</code>	Needle classes for the compass plot.
<code>org.jfree.chart.plot</code>	Plot classes and interfaces.
<code>org.jfree.chart.plot.dial</code>	Dial plot classes and interfaces.
<code>org.jfree.chart.renderer</code>	The base package for renderers.
<code>org.jfree.chart.renderer.category</code>	Plug-in renderers for use with the <code>CategoryPlot</code> class.
<code>org.jfree.chart.renderer.xy</code>	Plug-in renderers for use with the <code>XYPlot</code> class.
<code>org.jfree.chart.servlet</code>	Servlet utility classes.
<code>org.jfree.chart.title</code>	Chart title classes.
<code>org.jfree.chart.urls</code>	Interfaces and classes for generating URLs in image maps.
<code>org.jfree.chart.util</code>	Utility classes.
<code>org.jfree.data</code>	Dataset interfaces and classes.
<code>org.jfree.data.category</code>	The <code>CategoryDataset</code> interface and related classes.
<code>org.jfree.data.contour</code>	The <code>ContourDataset</code> interface and related classes.
<code>org.jfree.data.function</code>	The <code>Function2D</code> interface and related classes.
<code>org.jfree.data.gantt</code>	Dataset interfaces and classes for Gantt charts.
<code>org.jfree.data.general</code>	General dataset classes.
<code>org.jfree.data.io</code>	General I/O classes for datasets.
<code>org.jfree.data.jdbc</code>	Some JDBC dataset classes.
<code>org.jfree.data.statistics</code>	Classes that are used for generating statistics.
<code>org.jfree.data.time</code>	Time-based dataset interfaces and classes.
<code>org.jfree.data.time.ohlc</code>	Classes to represent an open-high-low-close dataset.
<code>org.jfree.data.xml</code>	Classes for reading datasets from XML.
<code>org.jfree.data.xy</code>	The <code>XYDataset</code> interface and related classes.

Additional information can be found in the HTML format API documentation that is generated from the JFreeChart source files.

# Chapter 22

## Package: org.jfree.chart

### 22.1 Overview

This package contains the major classes and interfaces in the *JFreeChart Class Library*, including the all important `JFreeChart` class.

### 22.2 ChartColor

#### 22.2.1 Overview

This class defines some standard colors.

#### 22.2.2 Methods

This class defines the following methods:

► `public static Paint[] createDefaultPaintArray();`  
Returns an array of `Paint` instances. This array is used by the `DefaultDrawingSupplier` class as the default series colors for most plots.

### 22.3 ChartFactory

#### 22.3.1 Overview

This class contains a range of convenient methods for creating standard types of charts.

*HINT: The use of these methods is optional. Take a look at the source code for the method you are using to see if it might be a better option to cut-and-paste the code into your application, and then customise it to meet your requirements.*

#### 22.3.2 Pie Charts

To create a regular pie chart, you can use either of the following methods:

► `public static JFreeChart createPieChart(String title, PieDataset dataset, boolean legend, boolean tooltips, Locale locale); [1.0.7]`  
Creates a pie chart based on the specified dataset (`null` permitted), with tooltips formatted according to the specified locale. The resulting chart is constructed using a `PiePlot`.  
► `public static JFreeChart createPieChart(String title, PieDataset dataset, boolean legend, boolean tooltips, boolean urls);`  
Creates a pie chart based on the specified dataset (`null` permitted). The chart is constructed

using a `PiePlot`. Note that URL support is only applicable to the creation of HTML image maps.

To create a pie chart with a “3D effect”:

```
➔ public static JFreeChart createPieChart3D(String title, PieDataset dataset,
boolean legend, boolean tooltips, Locale locale); [1.0.7]
Creates a pie chart with a 3D perspective, using the specified dataset (which may be null).
The chart is constructed using a PiePlot3D. The locale is used to create default formatters for
the tool tip generator.

➔ public static JFreeChart createPieChart3D(String title, PieDataset dataset,
boolean legend, boolean tooltips, boolean urls)
Creates a 3D pie chart for the specified PieDataset (null permitted). The chart is constructed
using a PiePlot3D.
```

To create a single chart containing multiple pie charts:

```
➔ public static JFreeChart createMultiplePieChart(String title, CategoryDataset dataset,
TableOrder order, boolean legend, boolean tooltips, boolean urls);
Creates a multiple pie chart for the specified CategoryDataset. This chart is constructed using a
MultiplePiePlot. The order argument can be either TableOrder.BY_ROW or TableOrder.BY_COLUMN.
```

To create a single chart containing multiple pie charts with a “3D effect”:

```
➔ public static JFreeChart createMultiplePieChart3D(String title, CategoryDataset dataset,
TableOrder order, boolean legend, boolean tooltips, boolean urls);
Creates a multiple pie chart for the specified CategoryDataset. This chart is constructed using a
MultiplePiePlot. The order argument can be either TableOrder.BY_ROW or TableOrder.BY_COLUMN.
```

A special case of pie chart can be created to display the “difference” between two datasets:

```
➔ public static JFreeChart createPieChart(String title, PieDataset dataset,
PieDataset previousDataset, int percentDiffForMaxScale, boolean greenForIncrease,
boolean legend, boolean tooltips, Locale locale, boolean subTitle, boolean showDifference);
[1.0.7]
Returns a pie chart that displays the difference between the two supplied datasets.

➔ public static JFreeChart createPieChart(String title, PieDataset dataset,
PieDataset previousDataset, int percentDiffForMaxScale, boolean greenForIncrease,
boolean legend, boolean tooltips, boolean urls, boolean subTitle, boolean showDifference);
As above.
```

A ring chart is a customised form of pie chart:

```
➔ public static JFreeChart createRingChart(String title, PieDataset dataset, boolean legend,
boolean tooltips, Locale locale); [1.0.7]
Creates a ring chart based on the supplied dataset (which may be null). Numerical values in
the default tool tip generator will be formatted according to the specified locale.

➔ public static JFreeChart createRingChart(String title, PieDataset dataset, boolean legend,
boolean tooltips, boolean urls);
Creates a ring chart based on the supplied dataset (which may be null). Note that URL
support is only applicable for charts that are used to create HTML image maps.
```

### 22.3.3 Bar Charts

To create a bar chart:

```
➔ public static JFreeChart createBarChart(String title, String categoryAxisLabel,
String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates a horizontal or vertical bar chart for the given CategoryDataset (see the BarRenderer
class documentation for an example).
```

To create a bar chart with a “3D effect”:

```
→ public static JFreeChart createBarChart3D(String title, String categoryAxisLabel,
String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates a bar chart with 3D effect for the given CategoryDataset (see the BarRenderer3D class documentation for an example).
```

To create a stacked bar chart:

```
→ public static JFreeChart createStackedBarChart(String title, String categoryAxisLabel,
String valueAxisLabel, CategoryDataset data, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates a stacked bar chart for the given CategoryDataset.
```

To create a stacked bar chart with a “3D effect”:

```
→ public static JFreeChart createStackedBarChart3D(String title, String categoryAxisLabel,
String valueAxisLabel, CategoryDataset data, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates a stacked bar chart with 3D effect for the given CategoryDataset.
```

To create a bar chart using an IntervalXYDataset (bearing in mind that you can use the XYBarDataset wrapper to convert any XYDataset to the required type):

```
→ public static JFreeChart createXYBarChart(String title, String xAxisLabel,
boolean dateAxis, String yAxisLabel, IntervalXYDataset dataset, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates an XY bar chart for the given IntervalXYDataset. The dateAxis argument allows you to select whether the chart is created with a DateAxis or a NumberAxis for the domain axis. The chart created with this method uses a XYPlot and XYBarRenderer.
```

#### 22.3.4 Line Charts

To create a line chart based on a CategoryDataset:

```
→ public static JFreeChart createLineChart(String title, String categoryAxisLabel,
String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates a line chart for the given CategoryDataset. The chart will be constructed with a CategoryPlot and a LineAndShapeRenderer.

→ public static JFreeChart createLineChart3D(String title, String categoryAxisLabel,
String valueAxisLabel, CategoryDataset dataset, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates a line chart for the given CategoryDataset, with a pseudo 3D effect. The chart will be constructed with a CategoryPlot and a LineRenderer3D.
```

To create a line chart based on a XYDataset:

```
→ public static JFreeChart createXYLineChart(String title, String xAxisLabel,
String yAxisLabel, XYDataset dataset, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls)
Creates a XY line chart for the given XYDataset.
```

#### 22.3.5 Other Chart Types

To create a scatter plot:

```
→ public static JFreeChart createScatterPlot(String title, String xAxisLabel, String yAxisLabel,
XYDataset data, PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);
Creates a scatter plot for the given XYDataset.
```

To create a time series chart:

```
→ public static JFreeChart createTimeSeriesChart(String title, String timeAxisLabel,
String valueAxisLabel, XYDataset data, boolean legend, boolean tooltips, boolean urls);
Creates a time series chart for the given XYDataset.
```

To create a high-low-open-close chart:

```
→ public static JFreeChart createHighLowChart(String title, String timeAxisLabel,
String valueAxisLabel, OHLCdataset dataset, Timeline timeline, boolean legend);
Creates a high-low-open-close chart for the given OHLCdataset.
```

To create a candlestick chart:

```
→ public static JFreeChart createCandlestickChart(String title, String timeAxisLabel,
String valueAxisLabel, OHLCdataset data, boolean legend);
Creates a candlestick chart for the given OHLCdataset.
```

To create an area chart using data from a XYDataset:

```
→ public static JFreeChart createXYAreaChart(String title, String xLabel,
String yLabel, XYDataset dataset, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates an area chart for the specified dataset. The chart that is created uses a XYPlot and a
XYAreaRenderer.
```

To create a stacked area chart using data from a TableXYDataset:

```
→ public static JFreeChart createStackedXYAreaChart(String title, String xLabel,
String yLabel, TableXYDataset dataset, PlotOrientation orientation,
boolean legend, boolean tooltips, boolean urls);
Creates a stacked area chart for the specified dataset (notice that the dataset must be a
TableXYDataset for stacking). The chart that is created uses a XYPlot and a StackedXYAreaRenderer.
```

To create a box-and-whisker chart where the domain values are categories:

```
→ public static JFreeChart createBoxAndWhiskerChart(String title, String categoryAxisLabel,
String valueAxisLabel, BoxAndWhiskerCategoryDataset dataset, boolean legend); [1.0.4]
Creates a box-and-whisker chart from the specified dataset. This chart will use a CategoryPlot
and a BoxAndWhiskerRenderer.
```

To create a box-and-whisker chart where the domain values are numbers or dates:

```
→ public static JFreeChart createBoxAndWhiskerChart(String title, String timeAxisLabel,
String valueAxisLabel, BoxAndWhiskerXYDataset dataset, boolean legend);
Creates a box-and-whisker chart from the specified dataset. This chart will use a XYPlot and a
BoxAndWhiskerRenderer.
```

### 22.3.6 Notes

This class contains methods for common chart types only. There are many other chart types that you can create by mixing and matching plots and renderers. It is worthwhile to review the source code for the methods in this class, then consider how you could substitute different renderers or axes to create different types of charts. Don't be afraid to copy, paste and modify the code from this class!

## 22.4 ChartFrame

### 22.4.1 Overview

A frame containing a chart within a ChartPanel.

## 22.4.2 Constructors

There are two constructors:

```
► public ChartFrame(String title, JFreeChart chart);
Creates a new ChartFrame containing the specified chart.
```

The second constructor gives you the opportunity to request that the chart is contained within a `JScrollPane`:

```
► public ChartFrame(String title, JFreeChart chart, boolean scrollPane);
Creates a new ChartFrame containing the specified chart. The scrollPane flag indicates whether or not the chart should be displayed within a ScrollPane.
```

## 22.4.3 Methods

To access the chart's panel:

```
► public ChartPanel getChartPanel();
Returns the panel that contains the chart.
```

### Notes

This class is used in a few demo applications, but you won't generally need to use it yourself—instead, you'll most likely create a `ChartPanel` and add that directly to your own forms.

## 22.5 ChartMouseEvent

### 22.5.1 Overview

An event generated by the `ChartPanel` class to represent a mouse click or a mouse movement over a chart. These events are passed to listeners via the `ChartMouseListener` interface.

### 22.5.2 Constructor

To create a new event:

```
► public ChartMouseEvent(JFreeChart chart, MouseEvent trigger, ChartEntity entity);
Creates a new event for the specified chart. The event also records the underlying trigger event and the entity underneath the mouse pointer (possibly null).
```

Event objects will usually be created by the `ChartPanel` class and sent to all registered listeners—you won't normally need to create an instance of this class yourself.

### 22.5.3 Methods

Use the following methods to access the attributes for the event:

```
► public JFreeChart getChart();
Returns the chart (never null) that the event relates to.

► public MouseEvent getTrigger();
Returns the underlying mouse event (never null) that triggered the generation of this event. This contains information about the mouse location, among other things. Note that the mouse location here is given in coordinates relative to the source component (which is the ChartPanel)—to convert to the corresponding (x, y) values in data space, you need to take into account the axis ranges and the current data area (see MouseListenerDemo4.java for an example).

► public ChartEntity getEntity();
Returns the chart entity underneath the mouse pointer (this may be null). There are many subclasses of the ChartEntity class, and by determining which subclass is used you can find additional information about the entity "underneath" the mouse pointer.
```

### 22.5.4 Notes

Some points to note:

- to receive notification of these events, an object first needs to implement the `ChartMouseListener` interface and then register itself with a `ChartPanel` object, via the `addChartMouseListener()` method (see section 22.7.6);<sup>1</sup>
- some demos (`MouseListenerDemo1-4.java`) are included in the JFreeChart demo collection.

## 22.6 ChartMouseListener

### 22.6.1 Overview

An interface that defines the callback methods for a *chart mouse listener*. Any class that implements this interface can be registered with a `ChartPanel` and receive notification of mouse events. This mechanism can be used to implement interactive charts in Swing applications (information about the chart element at the current mouse location is contained in the `ChartMouseEvent` object). This is a low-level mechanism, but very flexible.

### 22.6.2 Methods

This receives notification of mouse click events:

```
► void chartMouseClicked(ChartMouseEvent event);
A callback method for receiving notification of a mouse click on a chart.
```

This method receives notification of mouse movement events:

```
► void chartMouseMoved(ChartMouseEvent event);
A callback method for receiving notification of a mouse movement event on a chart.
```

### 22.6.3 Notes

Some points to note:

- some demo applications (`MouseListenerDemo1-4.java`) are included in the JFreeChart demo collection.

## 22.7 ChartPanel

### 22.7.1 Overview

A panel that provides a convenient means to display a `JFreeChart` instance in a Swing-based user-interface (extends `javax.swing.JPanel`).

The panel can be set up to include a popup menu providing access to:

- chart properties – the property editors are incomplete, but allow you to customise many chart properties;
- printing – print a chart via the standard Java printing facilities;
- saving – write the chart to a PNG format file;

---

<sup>1</sup>It should be obvious, but apparently needs stating in some cases, that the mouse events relate to JFreeChart usage in a Swing-based application—if you are developing web-applications (and don’t want to employ applets or Java Web Start) then you’ll need to rely on chart images plus HTML image maps (all of which JFreeChart can help with, but none of which have anything to do with `ChartMouseEvents`).

- zooming – zoom in or out by adjusting the axis ranges;

In addition, the panel can:

- provide offscreen buffering to improve performance when redrawing overlapping frames;
- display tool tips;

All of these features are used in the demonstration applications included with the JFreeChart Developer Guide.

### 22.7.2 Constructors

The standard constructor accepts a `JFreeChart` as the only parameter, and creates a panel that displays the chart:

```
► public ChartPanel(JFreeChart chart);
Creates a new panel for displaying the specified chart.
```

By default, the panel is automatically updated whenever the chart changes (for example, if you modify the range for an axis, the chart will be redrawn automatically).

```
► public ChartPanel(JFreeChart chart, boolean useBuffer);
Creates a new panel for displaying the specified chart. If useBuffer is true, the panel draws the chart to an off-screen buffered image, then copies the image to the screen as required. For charts that draw slowly (for example, those with a large number of data points), this will improve performance by ensuring that the chart is only redrawn when necessary.
```

### 22.7.3 The Chart

The chart that is displayed by the panel is accessible via the following methods:

```
► public JFreeChart getChart();
Returns the chart that is displayed in the panel.

► public void setChart(JFreeChart chart);
Sets the chart that is displayed in the panel. The panel registers with the chart as a change listener, so that it can repaint the chart whenever it changes.
```

### 22.7.4 Chart Scaling

JFreeChart is designed to draw charts at arbitrary sizes. In the case of the `ChartPanel` class, the chart is drawn to fit the current size of the panel (which is usually determined externally by a layout manager). When the panel gets very small (or very large) the layout procedure used by JFreeChart may not produce good results. To counteract this, the `ChartPanel` class specifies minimum and maximum drawing thresholds. When the panel dimensions fall below the minimum threshold (or above the maximum threshold) the chart is drawn at the maximum (minimum) size then scaled down (up) to fit the actual panel size.

You can control the threshold values with the following methods:

```
► public int getMinimumDrawWidth();
Returns the lower threshold for the chart drawing width. The default is 300 pixels.

► public void setMinimumDrawWidth(double width);
Sets the lower threshold for the chart drawing width. If the panel is narrower than this, the chart is drawn at the specified width then scaled down to fit the panel.

► public int getMaximumDrawHeight();
Returns the lower threshold for the chart drawing height. The default is 200 pixels.
```

```
► public void setMinimumDrawHeight(double height);
```

Sets the lower threshold for the chart drawing height. If the panel is shorter than this, the chart is drawn at the specified height then scaled down to fit the panel.

For the maximum drawing size threshold, you can use the following methods:

```
► public int getMaximumDrawWidth();
```

Returns the upper threshold for the chart drawing width. The default value is 800 pixels.

```
► public void setMaximumDrawWidth(double width);
```

Sets the upper threshold for the chart drawing width. If the panel is wider than this, the chart is drawn at the specified width then scaled up to fit the panel.

```
► public int getMaximumDrawHeight();
```

Returns the upper threshold for the chart drawing height. The default value is 600 pixels.

```
► public void setMaximumDrawHeight(double height);
```

Sets the upper threshold for the chart drawing height. If the panel is taller than this, the chart is drawn at the specified height then scaled up to fit the panel.

When chart scaling is being applied, the `getScreenDataArea()` can be used to determine the data area in the coordinate space of the panel.

### 22.7.5 Tooltips

The panel includes support for displaying tool tips (assuming that tool tips have been generated by the plot or renderer). To disable (or re-enable) the display of tool tips, use the following method:

```
► public void setDisplayToolTips(boolean flag);
```

Switches the display of tool tips on or off for this panel.

The panel uses the standard Swing tool tip mechanism, which means that the tool tip timings (initial delay, dismiss delay and reshown delay) can be controlled application-wide using the usual Swing API calls. In addition, the panel has a facility to temporarily override the application wide settings while the mouse pointer is within the bounds of the panel:

```
► public void setInitialDelay(int delay);
```

Sets the initial delay (in milliseconds) before tool tips are displayed.

```
► public void setDismissDelay(int delay);
```

Sets the delay (in milliseconds) before tool tips are dismissed.

```
► public void setReshowDelay(int delay);
```

Sets the delay (in milliseconds) before tool tips are reshown.

### 22.7.6 Chart Mouse Events

Any object that implements the `ChartMouseListener` interface can register with the panel to receive notification of any mouse events that relate to the chart.

```
► public void addChartMouseListener(ChartMouseListener listener)
```

Adds an object to the list of objects that should receive notification of any `ChartMouseEvents` that occur.

```
► public void removeChartMouseListener(ChartMouseListener listener);
```

Removes an object from the list of objects that should receive notification of chart mouse events.

### 22.7.7 The Popup Menu

The chart panel has a popup menu that provides menu items for property editing, saving charts to PNG, printing charts, and some zooming options. The constructors provide options for including/excluding any of these options.

You can access the popup menu with the following methods:

- `public JPopupMenu getPopupMenu();`  
Returns the popup menu for the panel.
- `public void setPopupMenu(JPopupMenu popup);`  
Sets the popup menu for the panel. Set this to `null` if you don't want a popup menu at all.

A couple of the functions that can be accessed via the popup menu can also be called via the API:

- `public void doEditChartProperties(); [1.0.3]`  
Presents a chart property editor that allows some chart properties to be updated. As mentioned elsewhere, the property editors are incomplete.
- `public void doSaveAs() throws IOException;`  
Presents a file chooser component that allows the user to save the chart to a file in PNG format.
- `public void createChartPrintJob();`  
Presents a print dialog and prints the chart to a single page.

A default directory can be specified for the “save as” option:

- `public File getDefaultDirectoryForSaveAs(); [1.0.7]`  
Returns the default directory presented in the file chooser when saving a chart via the “Save As...” menu item. The default value is `null`, which means the user's home directory is selected.
- `public void setDefaultDirectoryForSaveAs(File directory); [1.0.7]`  
Sets the default directory that is presented in the file chooser when saving a chart via the “Save As...” menu item. If you set this to `null`, the user's home directory will be used as the default. If the `directory` argument is not in fact a directory, this method throws an `IllegalArgumentException`.

### 22.7.8 Zooming

The chart panel supports chart zooming for many types of charts. From the user perspective, zooming is initiated either via the popup menu or via a mouse drag on the displayed chart. The following methods are used to switch zooming on or off, for one or both axes:

- `public boolean isDomainZoomable();`  
Returns `true` if the panel will update the domain axis bounds in response to zoom requests, and `false` otherwise.
- `public void setDomainZoomable(boolean flag);`  
Sets the flag that controls whether or not the panel will update the domain axis bounds (assuming the axis supports this) in response to zoom requested.
- `public boolean isRangeZoomable();`  
Returns `true` if the panel will update the range axis bounds in response to zoom requests, and `false` otherwise.
- `public void setRangeZoomable(boolean flag);`  
Sets the flag that controls whether or not the panel will update the range axis bounds (assuming the axis supports this) in response to zoom requested.
- `public void setMouseZoomable(boolean flag);`  
A convenience method that sets the `domainZoomable` and `rangeZoomable` flags simultaneously.
- `public void setMouseZoomable(boolean flag, boolean fillRectangle)`  
A convenience method that sets the `domainZoomable` and `rangeZoomable` flags simultaneously.

The `fillZoomRectangle` flag controls the appearance of the rectangle drawn on the panel to show the zoom area while the user drags the mouse over the chart panel:

► `public boolean getFillZoomRectangle();`  
 Returns `true` if the zoom rectangle should be filled, and `false` if it should be drawn as an outline only.

► `public void setFillZoomRectangle(boolean flag);`  
 Sets the flag that controls whether or not the zoom rectangle is filled or drawn as an outline. The zoom rectangle is displayed while the mouse is dragged within the chart panel, to highlight the area that will be displayed in the chart once the zoom is completed.

The zoom trigger distance specifies the minimum distance that the mouse must be dragged before a zoom operation is triggered:

► `public int getZoomTriggerDistance();`  
 Returns the minimum distance (in Java2D units) that the mouse must be dragged in order to trigger a zoom operation. The default value is 10.

► `public void setZoomTriggerDistance(int distance);`  
 Sets the minimum distance (in Java2D units) that the mouse must be dragged in order to trigger a zoom operation.

The zooming operations interact with the plot via the `Zoomable` interface—plots that do not implement this interface will not be zoomable.

### 22.7.9 Other Methods

To get information about the entities in the chart drawn within the panel:

► `public ChartRenderingInfo getChartRenderingInfo();`  
 Returns a structure containing information about the chart drawn within the panel. Note that any dimensions in this structure do not take into account the scaling that may be applied by the panel.

Some convenience methods can return information about the chart. To find the data area for a chart (that is, the area inside the axes where the data is plotted) in the coordinate space of the panel, you can use the following methods:

► `public Rectangle2D getScreenDataArea();`  
 Returns the area within which the data is plotted, in screen coordinates. This takes into account any scaling applied by the panel—see section 22.7.4.

► `public Rectangle2D getScreenDataArea(int x, int y);`  
 Returns the area within which the data is plotted on the screen, for the subplot at the *screen* coordinate (*x*, *y*). The returned data area is also specified in screen coordinates—this takes into account any scaling applied by the panel. If the chart doesn't have any subplots, this method is equivalent to the `getScreenDataArea()` method (that is, it returns the data area for the main plot).

### 22.7.10 Notes

The size of the `ChartPanel` is determined by the layout manager used to arrange components in your user interface. In some cases, the layout manager will respect the *preferred size* of the panel, which you can set like this:

```
chartPanel.setPreferredSize(new Dimension(500, 270));
```

This class implements the `Printable` interface, to provide a simple mechanism for printing a chart. An option in the panel's popup menu calls the `createPrintJob()` method. The print job ends up calling the `print()` method to draw the chart on a single piece of paper.

If you need greater control over the printing process—for example, you want to display several charts on one page—you can write your own implementation of the `Printable` interface (in any class that has access to the chart(s) you want to print). The implementation incorporated with the `ChartPanel` class is a basic example, provided for convenience only.

**See Also**[JFreeChart](#).

## 22.8 ChartRenderingInfo

### 22.8.1 Overview

This class can be used to collect information about a chart as it is rendered, particularly information concerning the dimensions of various sub-components of the chart.

In the current implementation, four pieces of information are recorded for most chart types:

- the chart area;
- the plot area (including the axes);
- the data area (“inside” the axes);
- the dimensions and other information (including tool tips) for the entities within a chart;

You have some control over the information that is generated. For instance, tool tips will not be generated unless you set up a generator in the renderer.

### 22.8.2 Constructors

The default constructor:

► `public ChartRenderingInfo();`

Creates a `ChartRenderingInfo` object. Entity information will be collected using an instance of `StandardEntityCollection`.

An alternative constructor allows you to supply a specific entity collection:

► `public ChartRenderingInfo(EntityCollection entities);`  
Creates a `ChartRenderingInfo` object.

### 22.8.3 Methods

To get the area in which the chart is drawn:

► `public Rectangle2D getChartArea();`

Returns the area in which the chart has been drawn.

► `public void setChartArea(Rectangle2D area);`

Sets the area (in Java2D space) into which the chart has been drawn. This method is called by JFreeChart, you won’t normally call it yourself. You should note that this method records (after the fact) where a chart has been drawn—setting this attribute has no impact on the chart itself.

To access the entity collection:

► `public EntityCollection getEntityCollection();`  
Returns the entity collection (which may be `null`).

► `public void setEntityCollection(EntityCollection entities);`

Sets the entity collection. If you set this to `null`, no entity information is retained as the chart is rendered (which saves a lot of resources, but means that tooltips and HTML image maps cannot be generated).

► `public void clear();`

Clears all the information from this instance.

► `public PlotRenderingInfo getPlotInfo();`

Returns the `PlotRenderingInfo` state object for this instance. You can use this to obtain rendering information for the chart’s plot.

## 22.8.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this instance for equality with an arbitrary object.

► public Object clone() throws CloneNotSupportedException;
Returns a deep clone of this instance.
```

## 22.8.5 Notes

The `ChartPanel` class automatically collects entity information using this class, because it needs it to generate tool tips.

# 22.9 ChartUtilities

## 22.9.1 Overview

This class contains utility methods for:

- creating images from charts—supported formats are PNG and JPEG;
- generating HTML image maps.

All of the methods in this class are `static`.

## 22.9.2 Generating PNG Images

The *Portable Network Graphics* (PNG) format is a good choice for creating chart images. The format offers:

- a free and open specification;
- fast and effective compression;
- no loss of quality when images are reconstructed from the compressed binary format;
- excellent support in most web clients;

JFreeChart provides support for writing charts in PNG format via either:

- an encoder developed by J. David Eisenberg (published as free software under the terms of the GNU LGPL). You can find this encoder at:

<http://www.catcode.com>

- Java's `ImageIO` library;

The former option is used on JDK 1.3.1, while the latter option is used with JDK 1.4.2 or later.

The most general method allows you to write the image data directly to an output stream:

```
► public static void writeChartAsPNG(OutputStream out, JFreeChart chart,
int width, int height) throws IOException
Writes a chart image of the specified size directly to the output stream in PNG format.
```

If you need to retain information about the chart dimensions and content (to create an HTML image map, for example) you can pass in a newly created `ChartRenderingInfo` object using this method:

```
→ public static void writeChartAsPNG(OutputStream out, JFreeChart chart,
int width, int height, ChartRenderingInfo info)
Writes a chart image of the specified size directly to the output stream, and collects chart
information in the supplied info object. If info is null, no chart info is collected.
```

The above methods have counterparts that write image data directly to a file:

```
→ public static void saveChartAsPNG(File file, JFreeChart chart, int width, int height);
Saves a chart image of the specified size into the specified file, using the PNG format.

→ public static void saveChartAsPNG(File file, JFreeChart chart, int width, int height,
ChartRenderingInfo info);
Saves a chart to a PNG format image file. If an info object is supplied, it will be populated
with information about the structure of the chart.
```

### 22.9.3 Generating JPEG Images

The *Joint Photographic Experts Group* (JPEG) image format is supported using methods that are almost identical to those listed for PNG in the previous section.

*NOTE: JPEG is not an ideal format for charts. Images lose some definition after decompression from this format. This is most noticeable in high color contrast areas, which are common in charts. It is recommended that you use PNG format instead of JPEG, if at all possible.*

Since JFreeChart must rely on Java's ImageIO API to write images in JPEG format, these methods can only be used on Java 1.4.2 or later.

To write a chart to a file in JPEG format:

```
→ public static void saveChartAsJPEG(File file, JFreeChart chart, int width, int height);
Equivalent to saveChartAsJPEG(file, chart, width, height, null)—see the next method.
```

As with the PNG methods, if you need to know more information about the structure of the chart within the generated image, you will need to pass in a `ChartRenderingInfo` object:

```
→ public static void saveChartAsJPEG(File file, JFreeChart chart, int width, int height,
ChartRenderingInfo info);
Saves a chart to a JPEG format image file with the specified dimensions. If info is not null,
it will be populated with information about the structure of the chart. If file or chart is null,
this method throws an IllegalArgumentException.
```

Alternative methods allow you to specify the quality setting for the JPEG encoding:

```
→ public static void saveChartAsJPEG(File file, float quality, JFreeChart chart, int width,
int height) throws IOException;
Equivalent to saveChartAsJPEG(file, quality, chart, width, height, null)—see the next method.

→ public static void saveChartAsJPEG(File file, float quality, JFreeChart chart, int width,
int height, ChartRenderingInfo info) throws IOException;
Saves a chart to a JPEG format image file with the specified dimensions. The quality setting
should be in the range 0.0 (low quality) to 1.0 (high quality). If file or chart is null, this
method throws an IllegalArgumentException.
```

### 22.9.4 HTML Image Maps

An *HTML image map* is an HTML fragment used to describe the characteristics of an image file. The image map can define regions within the image, and associate these with URLs and tooltip information.

*NOTE: Most methods supporting HTML image map creation have been relocated in the `ImageMapUtilities` class.*

To generate a simple HTML image map for a `JFreeChart` instance, first generate an image for the chart and be sure to retain the `ChartRenderingInfo` object from the image drawing. Then, generate the image map using the following method:

```
► public static void writeImageMap(PrintWriter writer, String name,
    ChartRenderingInfo info, boolean useOverLibForToolTips);
    Writes a <MAP> element containing the region definitions for a chart that has been converted to
    an image. The info object should be the structure returned from the method call that wrote
    the chart to an image file.
```

There are two demonstration applications in the JFreeChart download that illustrate how this works: `ImageMapDemo1` and `ImageMapDemo2`.

## 22.9.5 Notes

Some points to note:

- when writing charts to image files, PNG tends to be a better format for charts than JPEG since the compression is “lossless” for PNG.

## 22.10 ClipPath

### 22.10.1 Overview

This class is used by the `ContourPlot` class. *This class is deprecated as of version 1.0.4.*

## 22.11 DrawableLegendItem

### 22.11.1 Overview

Used to represent a `LegendItem` plus it’s physical drawing characteristics (position, label location etc.) as it is being laid out on the chart.

*This class is deprecated (as of version 1.0.2) as it is no longer used by JFreeChart.*

## 22.12 Effect3D

### 22.12.1 Overview

An interface that should be implemented by renderers that use a “3D effect”. This allows the 3D axis classes to synchronise their own “3D effect” with that of the renderer and plot.

### 22.12.2 Methods

This interface defines two methods:

```
► public double getXOffset();
    Returns the x-offset (in Java2D units) for the 3D effect.

    ► public double getYOffset();
    Returns the y-offset (in Java2D units) for the 3D effect.
```

### See Also

`BarRenderer3D`, `CategoryAxis3D`, `NumberAxis3D`.

## 22.13 HashUtilities

### 22.13.1 Overview

A utility class to assist with the generation of hash codes. This class was first introduced in JFreeChart 1.0.3.

### 22.13.2 Methods

The following methods compute hash codes for the specified input:

```
→ public static int hashCodeForPaint(Paint p); [1.0.3]

→ public static int hashCodeForDoubleArray(double[] a); [1.0.3]

→ public static int hashCode(int pre, boolean b); [1.0.7]

→ public static int hashCode(int pre, double d); [1.0.7]

→ public static int hashCode(int pre, Paint p); [1.0.7]

→ public static int hashCode(int pre, Stroke s); [1.0.7]

→ public static int hashCode(int pre, String s); [1.0.7]

→ public static int hashCode(int pre, Comparable c); [1.0.7]

→ public static int hashCode(int pre, int i); [1.0.8]

→ public static int hashCode(int pre, Object obj); [1.0.8]

→ public static int hashCode(int pre, BooleanList list); [1.0.9]

→ public static int hashCode(int pre, PaintList list); [1.0.9]

→ public static int hashCode(int pre, StrokeList list); [1.0.9]
```

## 22.14 JFreeChart

### 22.14.1 Overview

The `JFreeChart` class coordinates the entire process of drawing charts. One method:

```
public void draw(Graphics2D g2, Rectangle2D area);
```

...instructs the `JFreeChart` object to draw a chart onto a specific area on some *graphics device*.

Java supports several graphics devices—including the screen, the printer, and buffered images—via different implementations of the abstract class `java.awt.Graphics2D`. Thanks to this abstraction, JFreeChart can generate charts on any of these target devices, as well as others implemented by third parties (for example, the SVG Generator implemented by the Batik Project).

In broad terms, the `JFreeChart` class sets up a context for drawing a `Plot`. The plot obtains data from a `Dataset`, and may delegate the drawing of individual data items to a `CategoryItemRenderer` or an `XYItemRenderer`, depending on the plot type (not all plot types use renderers).

The `JFreeChart` class can work with many different `Plot` subclasses. Depending on the type of plot, a specific dataset will be required. Table 22.1 summarises the combinations that are currently available:

Dataset:	Compatible Plot Types:
<code>BoxAndWhiskerCategoryDataset</code>	<code>CategoryPlot</code> with a <code>BoxAndWhiskerRenderer</code> .
<code>BoxAndWhiskerXYDataset</code>	<code>XYPlot</code> with a <code>XYBoxAndWhiskerRenderer</code> .
<code>CategoryDataset</code>	<code>CategoryPlot</code> subclasses with various renderers, or a <code>SpiderWebPlot</code> .
<code>ContourDataset</code>	<code>ContourPlot</code> .
<code>GanttCategoryDataset</code>	<code>CategoryPlot</code> with a <code>GanttRenderer</code> .
<code>IntervalCategoryDataset</code>	<code>CategoryPlot</code> with an <code>IntervalBarRenderer</code> .
<code>IntervalXYDataset</code>	<code>XYPlot</code> with an <code>XYBarRenderer</code> .
<code>OHLCDataset</code>	<code>XYPlot</code> with a <code>HighLowRenderer</code> or a <code>CandlestickRenderer</code> .
<code>PieDataset</code>	<code>PiePlot</code> .
<code>StatisticalCategoryDataset</code>	<code>CategoryPlot</code> with a <code>StatisticalBarRenderer</code> .
<code>ValueDataset</code>	<code>CompassPlot</code> , <code>MeterPlot</code> and <code>ThermometerPlot</code> .
<code>WaferMapDataset</code>	<code>WaferMapPlot</code> .
<code>WindDataset</code>	<code>XYPlot</code> with a <code>WindItemRenderer</code> .
<code>XYDataset</code>	<code>XYPlot</code> with various renderers.
<code>XYZDataset</code>	<code>XYPlot</code> with an <code>XYZBubbleRenderer</code> .

Table 22.1: Compatible plot and dataset types

## 22.14.2 Constructors

All constructors require you to supply a `Plot` instance (the `Plot` maintains a reference to the dataset used for the chart).

The simplest constructor is:

```
➔ public JFreeChart(Plot plot);
Creates a new JFreeChart instance. The chart will have no title, and no legend.
```

For greater control, a more complete constructor is available:

```
➔ public JFreeChart(Plot plot, String title, Font titleFont, boolean createLegend);
Creates a new JFreeChart instance. This constructor allows you to specify a single title (you can add additional titles, later, if necessary).
```

The `ChartFactory` class provides some utility methods that can make the process of constructing charts simpler.

## 22.14.3 Attributes

The attributes maintained by the `JFreeChart` class are listed in Table 22.2.

## 22.14.4 Anti-Aliasing

When drawing to pixel-based displays, the use of a technique called anti-aliasing can improve the appearance of the output by “smoothing” the edges of lines and shapes. Using anti-aliasing for drawing operations is usually slower, but the results often look better. You can control whether or not `JFreeChart` uses anti-aliasing with the following methods:

```
➔ public boolean getAntiAlias();
Returns true if this chart is drawn with anti-aliased graphics, and false otherwise.
```

Attribute:	Description:
<i>borderVisible</i>	A flag that controls whether or not a border is drawn around the outside of the chart.
<i>borderStroke</i>	The <b>Stroke</b> used to draw the chart's border.
<i>borderPaint</i>	The <b>Paint</b> used to paint the chart's border.
<i>title</i>	The chart title (an instance of <a href="#">TextTitle</a> ).
<i>subTitles</i>	A list of subtitles.
<i>legend</i>	The chart legend.
<i>plot</i>	The plot.
<i>antialias</i>	A flag that indicates whether or not the chart should be drawn with anti-aliasing.
<i>backgroundPaint</i>	The background paint for the chart.
<i>backgroundImage</i>	An optional background image for the chart.
<i>backgroundImageAlignment</i>	The alignment of the background image (if there is one).
<i>backgroundImageAlpha</i>	The alpha transparency for the background image.
<i>notify</i>	A flag that controls whether or not change events are passed on to the chart's registered listeners;
<i>renderingHints</i>	The Java2D rendering hints that will be applied when the chart is drawn.

Table 22.2: Attributes for the *JFreeChart* class

► `public void setAntiAlias(boolean flag);`

Sets a flag controlling whether or not anti-aliasing is used when drawing the chart, and sends a [ChartChangeEvent](#) to all registered listeners.

While people generally agree that anti-aliased shapes and lines look better, opinion is divided when it comes to text. Fortunately, the anti-aliasing setting can be controlled independently for text items, using the following methods:

► `public Object getTextAntiAlias(); [1.0.5]`

Returns the current hint for text anti-aliasing—see the `java.awt.RenderingHints` class for valid values. The default value is `null`, which generally means that the text follows the general anti-aliasing hint (see `getAntiAlias()`).

► `public void setTextAntiAlias(boolean flag); [1.0.5]`

A convenience method that switches text anti-aliasing on or off—see the following method.

► `public void setTextAntiAlias(Object val); [1.0.5]`

Sets the text anti-aliasing hint value to `val` and sends a [ChartChangeEvent](#) to all registered listeners. Valid arguments include:

- `null` – clears the setting, in which case the text will generally follow the hint that applies to general graphics (see `getAntiAlias()`);
- `RenderingHints.VALUE_TEXT_ANTIALIAS_ON` – text anti-aliasing on;
- `RenderingHints.VALUE_TEXT_ANTIALIAS_OFF` – text anti-aliasing off;
- `RenderingHints.VALUE_TEXT_ANTIALIAS_GASP` – introduced in Java 1.6.0, this setting turns anti-aliasing off for certain font sizes where hinting is optimal, and on for other sizes.

## 22.14.5 Methods

The most important method for a chart is the `draw()` method:

► `public void draw(Graphics2D g2, Rectangle2D chartArea);`

Draws the chart on the `Graphics2D` device, within the specified area.

The chart does not retain any information about the location or dimensions of the items it draws. Callers that require such information should use the alternative method:

```
► public void draw(Graphics2D g2, Rectangle2D chartArea, ChartRenderingInfo info);
```

Draws the chart on the `Graphics2D` device, within the specified area. If `info` is not `null`, it will be populated with information about the items drawn within the chart (to be returned to the caller).

To set the title for a chart:

```
► public void setTitle(String title);
```

Sets the title for a chart and sends a `ChartChangeEvent` to all registered listeners.

An alternative method for setting the chart title is:

```
► public void setTitle(TextTitle title);
```

Sets the title for a chart and sends a `ChartChangeEvent` to all registered listeners.

Although a chart can have only one title, it can have any number of subtitles:

```
► public void addSubtitle(Title title);
```

Adds a title to the chart.

The legend shows the names of the series (or sometimes categories) in a chart, next to a small color indicator. To add a legend to the chart:

```
► public void addLegend(LegendTitle legend);
```

Adds a legend to the chart and triggers a `ChartChangeEvent`. An `IllegalArgumentException` is thrown if `legend` is `null`. Note that legends are implemented as chart titles, so they can be positioned in the same way as any subtitle (at the top, bottom, left or right of the chart).

```
► public void removeLegend();
```

Removes the first legend from the chart and triggers a `ChartChangeEvent`.

To set the background paint for the chart:

```
► public void setBackgroundPaint(Paint paint);
```

Sets the background paint for the chart and sends a `ChartChangeEvent` to all registered listeners.

If this is set to `null`, the chart background will be transparent.

## 22.14.6 Background Image

A chart can have a background image (optional)—for an example, see `TimeSeriesDemo4.java` in the JFreeChart demo collection.

```
► public Image getBackgroundImage();
```

Returns the background image for the chart (possibly `null`).

```
► public void setBackgroundImage(Image image);
```

Sets the background image for the chart (`null` permitted) and sends a `ChartChangeEvent` to all registered listeners. You must ensure that the image is fully loaded before passing it to this method—see section 20.4 for more information.

To control the alignment of the background image:

```
► public int getBackgroundImageAlignment();
```

Returns a code that specifies the alignment of the background image.

```
► public void setBackgroundImageAlignment(int alignment);
```

Sets the alignment for the background image and sends a `ChartChangeEvent` to all registered listeners. Standard alignment codes are defined by the `Align` class.

To control the alpha transparency of the background image:

```
► public float getBackgroundImageAlpha();
```

Returns the alpha transparency for the background image.

```
► public void setBackgroundImageAlpha(float alpha);
```

Sets the alpha transparency for the background image then sends a `ChartChangeEvent` to all registered listeners. The `alpha` should be a value between `0.0` (fully transparent) and `1.0` (opaque).

An alternative option is to set a background image for the chart's `Plot`—this image will be positioned within the plot area only rather than the entire chart area.

### 22.14.7 The Chart Border

A border can be drawn around the outside of a chart, if required. By default, no border is drawn, since in many cases a border can be added externally (for example, in an HTML page). If you do require a border, use the following methods:

► `public boolean isBorderVisible();`

Returns the flag that controls whether or not a border is drawn around the outside of the chart. The default value is `false`.

► `public void setBorderVisible(boolean visible);`

Sets the flag that controls whether or not a border is drawn around the outside of the chart, and sends a `ChartChangeEvent` to all registered listeners.

To control the appearance of the border:

► `public Stroke getBorderStroke();`

Returns the `Stroke` used to draw the chart border, if there is one.

► `public void setBorderStroke(Stroke stroke);`

Sets the `Stroke` used to draw the chart border, if there is one, and sends a `ChartChangeEvent` to all registered listeners.

► `public Paint getBorderPaint();`

Returns the `Paint` used to draw the chart border, if there is one.

► `public void setBorderPaint(Paint paint);`

Sets the `Paint` used to paint the chart border, if there is one, and sends a `ChartChangeEvent` to all registered listeners.

### 22.14.8 Chart Change Listeners

If an object wants to “listen” for changes that are made to a chart, it needs to implement the `ChartChangeListener` interface so that it can register with the chart instance to receive `ChartChangeEvent` notifications.

For example, a `ChartPanel` instance automatically registers itself with the chart that it displays—any change to the chart results in the panel being repainted.

To receive notification of any change to a chart, a listener object should register via this method:

► `public void addChangeListener(ChartChangeListener listener);`

Register to receive chart change events.

To stop receiving change notifications, a listener object should deregister via this method:

► `public void removeChangeListener(ChartChangeListener listener);`

Deregister to stop receiving chart change events.

There are situations where you might want to temporarily disable the event notification mechanism—use the following methods:

► `public boolean isNotify();`

Returns the flag that controls whether or not change events are sent to registered listeners.

► `public void setNotify(boolean notify);`

Sets the flag that controls whether or not change events are sent to registered listeners. You can use this method to temporarily turn off the notification mechanism.

For example, when a chart is displayed in a `ChartPanel`, every update to the chart’s data will trigger a repaint of the chart. If you need to add several items to the chart’s dataset, typically you’ll only want the chart to be repainted once, after the last data item is added. You can achieve that as follows:

```
chart.setNotify(false);
// do several dataset updates here...
chart.setNotify(true);
```

### 22.14.9 Creating Images

The `JFreeChart` class includes utility methods for creating a `BufferedImage` containing the chart:

```
→ public BufferedImage createBufferedImage(int width, int height);
Creates a buffered image containing the chart. The size of the image is specified by the width and height arguments.
```

```
→ public BufferedImage createBufferedImage(int width, int height,
ChartRenderingInfo info);
Creates a buffered image containing the chart. The size of the image is specified by the width and height arguments. The info argument is used to collect information about the chart as it is being drawn (required if you want to create an HTML image map for the image).
```

One other variation draws the chart at one size then scales it (up or down) to fit a different image size:

```
→ public BufferedImage createBufferedImage(int imageWidth, int imageHeight,
double drawWidth, double drawHeight, ChartRenderingInfo info)
Creates an image containing a chart that has been drawn at one size then scaled (up or down) to fit the image size.
```

### 22.14.10 Notes

Some points to note:

- the `ChartFactory` class provides a large number of methods for creating “ready-made” charts.
- the Java2D API is used throughout JFreeChart, so JFreeChart does not work with JDK1.1 (a common question from applet developers).

## 22.15 LegendItem

### 22.15.1 Overview

A class that records the attributes of an item that should appear in a legend (see the `LegendTitle` class). Instances of this class are usually created by a renderer, which should set the attributes to match the visual representation of the corresponding series. Table 22.3 lists the attributes defined by the class.

Attribute:	Description:
<code>label</code>	The label (usually the series name).
<code>description</code>	A description of the item (not currently used).
<code>shapeVisible</code>	A flag that indicates whether or not the shape is visible.
<code>shape</code>	The shape displayed for the legend item.
<code>shapeFilled</code>	A flag that controls whether or not the shape is filled.
<code>fillPaint</code>	The fill paint.
<code>shapeOutlineVisible</code>	A flag that indicates whether or not the shape outline is visible.
<code>outlinePaint</code>	The outline paint.
<code>outlineStroke</code>	The outline stroke.
<code>lineVisible</code>	A flag that indicates whether or not the line is visible.
<code>lineStroke</code>	The line stroke.
<code>linePaint</code>	The line paint.

Table 22.3: Attributes for the `LegendItem` class

### 22.15.2 Constructors

To create a legend item:

```
↳ public LegendItem(String label, String description, String toolTipText,
String urlText, Shape shape, Paint fillPaint);
Creates a new legend item record.

↳ public LegendItem(String label, String description, String toolTipText,
String urlText, Shape shape, Paint fillPaint, Stroke outlineStroke, Paint outlinePaint);
Creates a new legend item record.

↳ public LegendItem(String label, String description, String toolTipText,
String urlText, Shape line, Stroke lineStroke, Paint linePaint);
Creates a new legend item record.

↳ public LegendItem(String label, String description, String toolTipText,
String urlText, boolean shapeVisible, Shape shape, boolean shapeFilled, Paint fillPaint, boolean
shapeOutlineVisible, Paint outlinePaint, Stroke outlineStroke, boolean lineVisible, Shape line,
Stroke lineStroke, Paint linePaint);
Creates a new legend item record.

↳ public LegendItem(AttributedString label, String description, String toolTipText,
String urlText, Shape shape, Paint fillPaint);
Creates a new legend item record.

↳ public LegendItem(AttributedString label, String description, String toolTipText,
String urlText, Shape shape, Paint fillPaint, Stroke outlineStroke, Paint outlinePaint);
Creates a new legend item record.

↳ public LegendItem(AttributedString label, String description, String toolTipText,
String urlText, Shape line, Stroke lineStroke, Paint linePaint);
Creates a new legend item record.

↳ public LegendItem(AttributedString label, String description, String toolTipText,
String urlText, boolean shapeVisible, Shape shape, boolean shapeFilled, Paint fillPaint, boolean
shapeOutlineVisible, Paint outlinePaint, Stroke outlineStroke, boolean lineVisible, Shape line,
Stroke lineStroke, Paint linePaint);
Creates a new legend item record.
```

### 22.15.3 Methods

The following methods are defined:

```
↳ public int getDatasetIndex(); [1.0.2]
Returns the dataset index for this legend item.

↳ public void setDatasetIndex(int index); [1.0.2]
Sets the dataset index for this legend item.

↳ public int getSeriesIndex(); [1.0.2]
Returns the series index for this legend item.

↳ public void setSeriesIndex(int index); [1.0.2]
Sets the series index for the legend item.

↳ public String getLabel();
Returns the legend item label.

↳ public AttributedString getAttributedLabel();
Returns an attributed legend item label, or null.

↳ public String getDescription();
Returns the description for the legend item (not used).

↳ public String getToolTipText();
Returns the tool tip text for the legend item.
```

```

→ public String getURLText();
Returns the URL for the legend item (only used in HTML image maps).

→ public boolean isShapeVisible();
Returns a flag that controls whether or not the legend item shape should be displayed.

→ public Shape getShape();
Returns the shape to display as the graphic for this legend item.

→ public boolean isShapeFilled();
Returns a flag that controls whether or not the legend item shape should be filled.

→ public Paint getFillPaint();
Returns the fill paint for the series represented by this legend item.

→ public boolean isShapeOutlineVisible();
Returns a flag that controls whether or not the legend item shape should have its outline drawn.

→ public Stroke getLineStroke();
Returns the stroke used to draw the line for the legend item graphic.

→ public Paint getLinePaint();
Returns the line paint.

→ public Paint getOutlinePaint();
Returns the outline paint for the series represented by this legend item.

→ public Stroke getOutlineStroke();
Returns the outline stroke for the series represented by this legend item.

→ public boolean isLineVisible();
Returns a flag that controls whether or not a line is drawn as part of the legend item graphic.

→ public Shape getLine();
Returns the line, if any, to be drawn for the legend item graphic.

→ public GradientPaintTransformer getFillPaintTransformer(); [1.0.4]
Returns the gradient paint transformer, if any, used by the renderer for the series represented by this legend item.

→ public void setFillPaintTransformer(GradientPaintTransformer transformer); [1.0.4]
Sets the gradient paint transformer used by the renderer for the series represented by this legend item.

```

#### 22.15.4 Notes

Some points to note:

- the `LegendItemSource` interface defines a method that should return a collection of legend items;
- originally this was an immutable class, which is why there are so many constructors with varying arguments, and some attributes with no setter methods;
- this class implements the `Serializable` interface.

#### See Also

[LegendItemCollection](#), [LegendTitle](#).

## 22.16 LegendItemCollection

### 22.16.1 Overview

A collection of legend items, typically returned by the `getLegendItems()` method in the plot classes. You can create your own collection of legend items and pass it to a `CategoryPlot` or `XYPlot` via the `setFixedLegendItems()` method, as a way of overriding the automatically generated legend items.

## 22.16.2 Constructors

There is a single constructor:

```
► public LegendItemCollection();
Creates a new empty collection.
```

## 22.16.3 Methods

To add an item to the collection:

```
► public void add(LegendItem item);
Adds the specified item to the collection.
```

To add a collection of items to this collection:

```
► public void addAll(LegendItemCollection collection);
Adds all the items from the given collection to this collection (by copying references, the items
themselves are not duplicated/cloned).
```

To find out how many items there are in the collection:

```
► public int getItemCount();
Returns the number of items in the collection.
```

To retrieve an item from the collection:

```
► public LegendItem get(int index);
Returns the item with the specified index.
```

To get an iterator that provides access to the items in the collection:

```
► public Iterator iterator();
Returns an iterator that provides access to the items in the collection.
```

## 22.16.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this collection for equality with an arbitrary object. This method returns true if and
only if:


- obj is not null;
- obj is an instance of LegendItemCollection;
- both collections contain the same items in the same order.

```

Instances of this class are `Cloneable` and `Serializable`.

### See Also

[LegendItem](#).

## 22.17 LegendItemSource

### 22.17.1 Overview

An interface for obtaining a collection of legend items. This interface is implemented (or extended) by:

- [Plot](#) (to work for all plot types);
- [CategoryItemRenderer](#);
- [XYItemRenderer](#);

A [LegendTitle](#) will use one or more of these sources to obtain legend items for display on the chart. This provides an opportunity for the legend to display just a subset of the items from a chart, if required.

ID:	Description:
LegendRenderingOrder.STANDARD	Items are rendered in order.
LegendRenderingOrder.REVERSE	Items are rendered in reverse order.

*Table 22.4: Tokens defined by LegendRenderingOrder*

## 22.17.2 Methods

To obtain a collection of legend items:

```
→ public LegendItemCollection getLegendItems();
```

Returns a collection of legend items (possibly empty, but never null).

### See Also

[LegendItem](#), [LegendItemCollection](#).

## 22.18 LegendRenderingOrder

### 22.18.1 Overview

A class that defines tokens that control the order of the items in the legend. See table 22.4 for the tokens that are defined.

### 22.18.2 Notes

This class is a left-over from older versions of JFreeChart, and is not currently used. It should probably be deprecated.

## 22.19 PolarChartPanel

### 22.19.1 Overview

An extension of the [ChartPanel](#) class with a pop-up menu that applies to polar charts.

# Chapter 23

## Package: org.jfree.chart.annotations

### 23.1 Overview

The annotations framework provides a mechanism for adding small text and graphics items to charts, usually to highlight a particular data item. In the current release, annotations can be added to the `CategoryPlot` and `XYPlot` classes. This framework is relatively basic at present, additional features are likely to be added in the future.

### 23.2 AbstractXYAnnotation

#### 23.2.1 Overview

A base class that can be used by classes that need to implement the `XYAnnotation` interface. Sub-classes provided by JFreeChart include:

- `XYBoxAnnotation` – draws a box at specified data coordinates;
- `XYDrawableAnnotation` – draws an instance of `Drawable`;
- `XYImageAnnotation` – draws an image;
- `XYLineAnnotation` – draws a line between specified data coordinates;
- `XYPointerAnnotation` – draws some text plus an arrow pointing at a data point;
- `XPolygonAnnotation` – draws a polygon;
- `XYShapeAnnotation` – draws an arbitrary `Shape`;
- `XYTextAnnotation` – draws a string.

If you create your own custom annotations, you don't have to subclass `AbstractXYAnnotation`, but it will save you some work.

#### 23.2.2 Constructors

This class defines a single (protected) constructor:

► `protected AbstractXYAnnotation();`  
Initialises the tool tip text and URL to `null`.

### 23.2.3 General Attributes

To access the tool tip text for the annotation:

```
↳ public String getToolTipText();
```

Returns the tool tip text for this annotation. The default value is `null`.

```
↳ public void setToolTipText(String text);
```

Sets the tool tip text for this annotation (`null` is permitted). No change event is generated.

To access the URL for the annotation:

```
↳ public String getUrl();
```

Returns the URL that will be used for this annotation in an HTML image map. The default value is `null`.

```
↳ public void setUrl(String url);
```

Sets the URL that will be used for this annotation in an HTML image map (`null` is permitted). No change event is generated.

### 23.2.4 Other Methods

The draw method is abstract, and must be implemented by subclasses:

```
↳ public abstract void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea,
```

```
ValueAxis domainAxis, ValueAxis rangeAxis, int rendererIndex, PlotRenderingInfo info);
```

To be implemented by subclasses. This method will be called by JFreeChart when the annotation needs to be drawn—you won’t normally call this method directly from your own code.

A utility method is provided for subclasses to add an entity:

```
↳ protected void addEntity(PlotRenderingInfo info, Shape hotspot, int rendererIndex,
```

```
String toolTipText, String urlText);
```

A utility method for adding an entity—this is available for calling by subclasses (typically from the `draw()` method).

### 23.2.5 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

```
↳ public boolean equals(Object obj);
```

Tests this annotation for equality with an arbitrary object. This method returns `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `AbstractXYAnnotation`;
- both annotations have the same tool tip and URL text.

Subclasses of this class should be `Cloneable` and `Serializable`, otherwise charts that use these annotations won’t support cloning and serialization.

### 23.2.6 Notes

Some points to note:

- there is no event notification mechanism for annotations, so when you update an annotation, the chart display will not automatically be refreshed. One way to trigger a repaint (at least, if your chart is displayed in a `ChartPanel`) is to call `chart.setNotify(true)`.

#### See Also

[XYAnnotation](#).

## 23.3 CategoryAnnotation

### 23.3.1 Overview

The interface that must be supported by annotations that are to be added to a `CategoryPlot`. The classes that implement this interface are:

- `CategoryLineAnnotation`;
- `CategoryPointerAnnotation`;
- `CategoryTextAnnotation`.

You can write your own annotation that implements this interface. Annotations are added to a plot using the `addAnnotation()` method (in the `CategoryPlot` class).

### 23.3.2 Interface

This interface defines a single method:

```
→ public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
CategoryAxis domainAxis, ValueAxis rangeAxis);
```

Draws the annotation. This method is typically called by JFreeChart, not user code.

### 23.3.3 Notes

Some points to note:

- for now, a `CategoryAnnotation` can only be added directly to a `CategoryPlot`, and is positioned relative to the plot's primary axes. It would make sense to allow annotations to be assigned to a renderer (as can be done with `XYAnnotation`) so that the annotation can be plotted against secondary axes.

#### See Also

[CategoryLineAnnotation](#), [CategoryPointerAnnotation](#), [CategoryTextAnnotation](#).

## 23.4 CategoryLineAnnotation

### 23.4.1 Overview

An annotation that draws a line between two points on a `CategoryPlot` (each defined by a *(category, value)* data item).<sup>1</sup> This class implements `CategoryAnnotation`.

### 23.4.2 Constructor

To create a new instance:

```
→ public CategoryLineAnnotation(Comparable category1, double value1,
Comparable category2, double value2, Paint paint, Stroke stroke);
```

Creates a new annotation that connects *(category1, value1)* and *(category2, value2)* with a straight line drawn using the specified paint and stroke.

---

<sup>1</sup>This class was requested by a client. Personally, I don't see a compelling use for it—if you know of one, please let me know!

### 23.4.3 General Attributes

To access the location used for the start of the line:

```
→ public Comparable getCategory1();
Returns the category for the start of the line (never null).

→ public void setCategory1(Comparable category);
Sets the category for the start of the line (null is not permitted). You should ensure that this
category actually exists in the dataset.

→ public double getValue1();
Returns the value for the start of the line.

→ public void setValue1(double value);
Sets the value for the start of the line.
```

To access the location used for the end of the line:

```
→ public Comparable getCategory2();
Returns the category for the start of the line (never null).

→ public void setCategory2(Comparable category);
Sets the category for the end of the line (null is not permitted). You should ensure that this
category actually exists in the dataset.

→ public double getValue2();
Returns the value for the end of the line.

→ public void setValue2(double value);
Sets the value for the end of the line.
```

To access the paint used to draw the line:

```
→ public Paint getPaint();
Returns the paint used to draw the line (never null).

→ public void setPaint(Paint paint);
Sets the paint used to draw the line (null is not permitted).
```

To access the stroke used to draw the line:

```
→ public Stroke getStroke();
Returns the stroke used to draw the line (never null).

→ public void setStroke(Stroke stroke);
Sets the stroke used to draw the line (null is not permitted).
```

### 23.4.4 Other Methods

The annotation is drawn by the following method, which is typically called by JFreeChart rather than user code:

```
→ public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
CategoryAxis domainAxis, ValueAxis rangeAxis);
Draws the annotation.
```

### 23.4.5 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

```
→ public boolean equals(Object obj);
Tests this annotation for equality with an arbitrary object. This method returns true if and
only if:


- obj is not null;
- obj is an instance of CategoryLineAnnotation;
- both annotations have the same field values.

```

Instances of this class are `Cloneable` and `Serializable`.

**See Also**

[CategoryAnnotation](#).

## 23.5 CategoryPointerAnnotation

### 23.5.1 Overview

An annotation for a [CategoryPlot](#) that displays a text item and an arrow pointing towards a point on a chart defined by a  $(category, value)$  pair—see figure 23.1 for an example. This class implements the [CategoryAnnotation](#) interface, and was first introduced in JFreeChart 1.0.3.

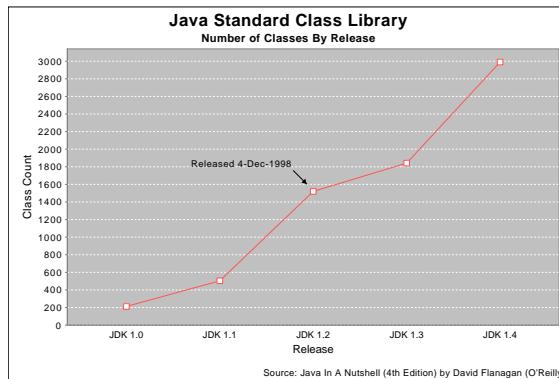


Figure 23.1: A *CategoryPointerAnnotation* (see *CategoryPointerAnnotationDemo1.java*)

### 23.5.2 Constructors

To create a new instance:

```
► public CategoryPointerAnnotation(String label, Comparable key, double value,
double angle);
```

Creates a new pointer annotation. The `label` is the text to be displayed (`null` not permitted). The `key` and `value` specify the location on the chart for the annotation. The `angle` specifies the rotation of the arrow that points at the specified point, in radians. To customise the appearance of the arrow, use the methods documented in the next section.

### 23.5.3 General Attributes

In addition to the attributes inherited from [CategoryTextAnnotation](#), this class defines a number of items concerning the appearance of the arrow that points towards a fixed location on the chart.

To control the angle of the pointer:

```
► public double getAngle(); [1.0.3]
```

Returns the angle of the pointer (in radians, using the same conventions as Java's `Arc2D` class).

```
► public void setAngle(double angle); [1.0.3]
```

Sets the angle of the pointer (in radians, using the same conventions as Java's `Arc2D` class). The arrow points towards a location on the chart (specified in the constructor).

To control the distance between the tip of the arrow and the anchor point on the chart:

```
► public double getTipRadius(); [1.0.3]
```

Returns the distance from the anchor point to the tip of the arrow, in Java2D units. The default value is 10.0.

► `public void setTipRadius(double radius); [1.0.3]`

Sets the distance from the anchor point to the tip of the arrow, in Java2D units. Since the tip of the arrow is pointing towards the anchor point, this should be a lower value than the *base radius*.

To control the length of the arrow:

► `public double getBaseRadius(); [1.0.3]`

Returns the distance from the anchor point to the base of the arrow, in Java2D units. The default value is 30.0. The difference between the base radius and the tip radius is the overall length of the arrow.

► `public void setBaseRadius(double radius); [1.0.3]`

Sets the distance from the anchor point to the base of the arrow, in Java2D units.

To control the offset from the base of the arrow to the label anchor point:

► `public double getLabelOffset(); [1.0.3]`

Returns the offset from the base of the arrow to the label, in Java2D units. The default value is 3.0.

► `public void setLabelOffset(double offset); [1.0.3]`

Sets the offset from the base of the arrow to the label, in Java2D units.

To control the length of the arrow head:

► `public double getArrowLength(); [1.0.3]`

Returns the length of the arrow head, in Java2D units. The default value is 5.0.

► `public void setArrowLength(double length); [1.0.3]`

Sets the length of the arrow head, in Java2D units.

To control the width of the arrow head:

► `public double getArrowWidth(); [1.0.3]`

Returns the width of the arrow head, in Java2D units. The default value is 3.0.

► `public void setArrowWidth(double width); [1.0.3]`

Sets the width of the arrow head, in Java2D units.

To control the stroke used to draw the arrow:

► `public Stroke getArrowStroke(); [1.0.3]`

Returns the stroke used to draw the arrow. The default is `BasicStroke(1.0f)`.

► `public void setArrowStroke(Stroke stroke); [1.0.3]`

Sets the stroke used to draw the arrow.

To control the color of the arrow:

► `public Paint getArrowPaint(); [1.0.3]`

Returns the paint used to draw/fill the arrow. The default is `Color.black`.

► `public void setArrowPaint(Paint paint); [1.0.3]`

Sets the paint used to draw/fill the arrow.

### 23.5.4 Other Methods

The following method is called by JFreeChart as required:

► `public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea, CategoryAxis domainAxis, ValueAxis rangeAxis); [1.0.3]`

Draws the annotation. This method is typically called by JFreeChart, not user code.

### 23.5.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj); [1.0.3]
    Tests this annotation for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 23.5.6 Notes

Some points to note:

- this class is a subclass of `CategoryTextAnnotation`;
- a demo (`CategoryPointerAnnotationDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[CategoryTextAnnotation](#).

## 23.6 CategoryTextAnnotation

### 23.6.1 Overview

A `CategoryAnnotation` that can be used to display an item of text at some location (defined by a *(category, value)* pair) on a `CategoryPlot`. This class extends `TextAnnotation`.

### 23.6.2 Constructor

To create a new annotation:

```
➔ public CategoryTextAnnotation(String text, Comparable category, double value);
    Creates a new annotation that displays the specified text at a point corresponding to the
    specified value for the given category.
```

### 23.6.3 General Attributes

This class inherits a number of attributes from `TextAnnotation`, and adds a few of its own. The text for the annotation is drawn relative to an alignment point that is defined on the chart using the following attributes:

- the category;
- the category anchor point;
- the data value.

To control the category:

```
➔ public Comparable getCategory();
    Returns the category key for this annotation.

➔ public void setCategory(Comparable category);
    Sets the category key for this annotation. If category is null, this method throws an
    IllegalArgumentException.
```

To control the category anchor point:

```
➔ public CategoryAnchor getCategoryAnchor();
    Returns the category anchor point, which helps to determine the position of the alignment
    point for the annotation. The default value is CategoryAnchor.MIDDLE.
```

```
→ public void setCategoryAnchor(CategoryAnchor anchor);
```

Sets the category anchor point, which is used to determine the position of the alignment point for the annotation. If `anchor` is `null`, this method throws an `IllegalArgumentException`.

To control the value:

```
→ public double getValue();
```

Returns the value that determines the alignment point for the annotation.

```
→ public void setValue(double value);
```

Sets the value that determines the alignment point for the annotation.

### 23.6.4 Other Methods

To draw the annotation:

```
→ public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
CategoryAxis domainAxis, ValueAxis rangeAxis);
```

Draws the annotation. This method is called by JFreeChart, you shouldn't need to call it directly.

### 23.6.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj);
```

Tests this annotation for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 23.6.6 Notes

Some points to note:

- there is no event notification for annotations, so automatic chart redrawing does not occur when an annotation is updated;
- `CategoryTextAnnotation` is a subclass of `TextAnnotation`;
- a demo (`SurveyResultsDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[CategoryAnnotation](#).

## 23.7 TextAnnotation

### 23.7.1 Overview

The base class for a *text annotation*. The class includes font, paint, alignment and rotation settings. Subclasses will add location information to the content represented by this class. At present, the only subclass in JFreeChart is `CategoryTextAnnotation`.

### 23.7.2 Constructor

The constructor for this class is `protected` since you won't create an instance of this class directly (use a subclass):

```
→ protected TextAnnotation(String text);
```

Creates a new text annotation that displays the given text. Default values for the remaining attributes are:

- `font = new Font("SansSerif", Font.PLAIN, 10);`
- `paint = Color.black;`
- `textAnchor = TextAnchor.CENTER;`
- `rotationAnchor = TextAnchor.CENTER;`
- `rotationAngle = 0.0;`

### 23.7.3 General Attributes

To control the text displayed by the annotation:

- `public String getText();`  
Returns the text displayed by the annotation (never `null`).
- `public void setText(String text);`  
Sets the text displayed by the annotation (`null` is not permitted).

To control the font:

- `public Font getFont();`  
Returns the font used to display the text. This method never returns `null`.
- `public void setFont(Font font);`  
Sets the font used to display the text. If `font` is `null`, this method throws an `IllegalArgumentException`.

To control the text color:

- `public Paint getPaint();`  
Returns the paint used to draw the text (never `null`).
- `public void setPaint(Paint paint);`  
Sets the paint used to draw the text. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the anchor point that will be aligned to some point (defined by the subclass):

- `public TextAnchor getTextAnchor();`  
Returns the anchor point for the text, this will be aligned to a specified point on the chart (that is defined by the subclass).
- `public void setTextAnchor(TextAnchor anchor);`  
Sets the anchor point for the text. This will be aligned to some point on the chart (that is specified by the subclass).

To control the rotation anchor point:

- `public TextAnchor getRotationAnchor();`  
Returns the text anchor point about which any rotation is performed.
- `public void setRotationAnchor(TextAnchor anchor);`  
Sets the rotation anchor point for the text.

To control the rotation angle:

- `public double getRotationAngle();`  
Returns the rotation angle (in radians).
- `public void setRotationAngle(double angle);`  
Sets the rotation angle for the text (in radians). The text is rotated about the rotation anchor point (see the `getRotationAnchor()` method).

### 23.7.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this annotation for equality with an arbitrary object. This method returns `true` if and only if:
  - `obj` is not `null`;
  - `obj` is an instance of `TextAnnotation`;
  - `obj` has the same attributes as this annotation.

Instances of this class are `Serializable`, but not `Cloneable`.

### 23.7.5 Notes

Some points to note:

- the `XYTextAnnotation` class is NOT a subclass of this class.

#### See Also

[CategoryTextAnnotation](#).

## 23.8 XYAnnotation

### 23.8.1 Overview

An `XYAnnotation` is a small text or graphical item that can be added to an arbitrary location on a chart. This interface defines the drawing method that must be supported by annotations that are to be added to an `XYPlot` (or an `XYItemRenderer`). This interface is implemented by:

- `XYBoxAnnotation`;
- `XYDrawableAnnotation`;
- `XYImageAnnotation`;
- `XYLineAnnotation`;
- `XYPointerAnnotation`;
- `XYPolygonAnnotation`;
- `XYShapeAnnotation`;
- `XYTextAnnotation`;

You can, of course, provide your own implementations of the interface.

### 23.8.2 Interface

This interface defines one method for drawing the annotation:

```
→ public void draw(Graphics2D g2, Rectangle2D dataArea, XYPlot plot,
ValueAxis domainAxis, ValueAxis rangeAxis);
```

Draws the annotation. The `dataArea` is the space defined by (within) the two axes. If the annotation defines its location in terms of data values, the axes can be used to convert these values to Java2D coordinates.

### 23.8.3 Notes

Some points to note:

- there is no event notification mechanism (yet) for annotations. If you modify an annotation, you will need to manually trigger a redraw of the chart (for example, by calling `chart.setNotify(true)`);
- a couple of demos (`AnnotationDemo1.java` and `AnnotationDemo2.java`) are included in the JFreeChart demo collection.

#### See Also

[AbstractXYAnnotation](#).

## 23.9 XYBoxAnnotation

### 23.9.1 Overview

An [XYAnnotation](#) that highlights a rectangular region within the data area of an [XYPlot](#)—see figure 23.2 for an example. [XYBoxAnnotation](#) is a subclass of [AbstractXYAnnotation](#).

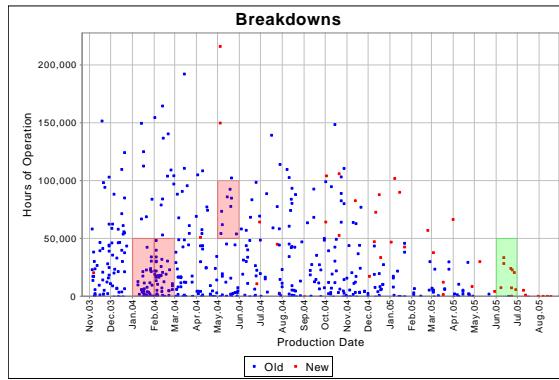


Figure 23.2: An [XYBoxAnnotation](#) (see [XYBoxAnnotationDemo1.java](#))

### 23.9.2 Constructors

To create a new annotation:

```
→ public XYBoxAnnotation(double x0, double y0, double x1, double y1);
Creates a new annotation covering the rectangular region from (x0, y0) to (x1, y1). The box annotation will be drawn as a black outline using a 1 unit wide plain stroke (BasicStroke\(1.0f\)).
```

```
→ public XYBoxAnnotation(double x0, double y0, double x1, double y1,
Stroke stroke, Paint outlinePaint);
Creates a new annotation covering the rectangular region from (x0, y0) to (x1, y1). The box annotation will be drawn as an outline using the specified stroke and paint. If either stroke or paint is null, the annotation will not be visible.
```

```
→ public XYBoxAnnotation(double x0, double y0, double x1, double y1,
Stroke stroke, Paint outlinePaint, Paint fillPaint);
Creates a new annotation covering the rectangular region from (x0, y0) to (x1, y1). The box annotation will be drawn and filled using the specified stroke, outline paint and fill paint. If stroke or outlinePaint is null, no outline will be drawn. If fillPaint is null, the box will not be filled.
```

### 23.9.3 Methods

JFreeChart calls the following method to draw the annotation:

```
→ public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea,
ValueAxis domainAxis, ValueAxis rangeAxis, int rendererIndex, PlotRenderingInfo info);
Draws the annotation within the specified area of the plot. This method is called by JFreeChart, you won't normally call it directly from your own code.
```

### 23.9.4 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

```
→ public boolean equals(Object obj);
Tests this annotation for equality with an arbitrary object. The method returns true if and only if:
```

- `obj` is not `null`;
- `obj` is an instance of `XYBoxAnnotation`;
- both annotations have the same attributes.

Instances of this class are `Cloneable` (`PublicCloneable`) and `Serializable`.

### 23.9.5 Notes

Some points to note:

- the annotation will only be visible if it falls within the current bounds of the plot's axes;
- you can define a tool tip and/or URL for the annotation, using methods inherited from `AbstractXYAnnotation`;
- you can use this annotation with an `XYPlot` that uses a `DateAxis`—just specify the relevant coordinates in terms of milliseconds since 1-Jan-1970;
- a demo (`XYBoxAnnotationDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[XYAnnotation](#).

## 23.10 XYDrawableAnnotation

### 23.10.1 Overview

An `XYAnnotation` that draws an object at some  $(x, y)$  location on an `XYPlot`. The object can be any implementation of the `Drawable` interface (defined in the JCommon class library). Figure 23.3 shows a chart with such an annotation—the small red circle containing the blue crosshair is an `XYDrawableAnnotation`.

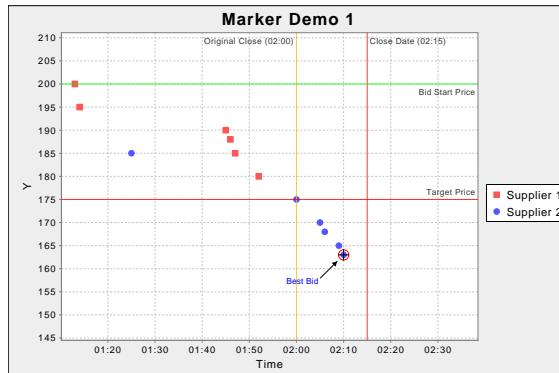


Figure 23.3: An `XYDrawableAnnotation` (see `MarkerDemo1.java`)

### 23.10.2 Constructors

To create a new annotation:

```
► public XYDrawableAnnotation(double x, double y, double width, double height, Drawable drawable);
Creates a new annotation that will be drawn within the specified rectangular area (in data space).
```

### 23.10.3 Notes

Some points to note:

- a demo (`MarkerDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[XYAnnotation](#).

## 23.11 XYImageAnnotation

### 23.11.1 Overview

An annotation that allows an image to be displayed at an arbitrary  $(x, y)$  location on an `XYPlot`. To add an image annotation to a plot, use code similar to the following:

```
XYPlot plot = (XYPlot) chart.getPlot();
Image image = ... // fetch a small image from somewhere
XYImageAnnotation a1 = new XYImageAnnotation(5.0, 2.0, image);
plot.addAnnotation(a1);
```

You need to ensure that the image is fully loaded before you supply it to the `XYImageAnnotation` constructor, otherwise it may not appear the first time your chart is drawn (see section [20.4](#)).

### 23.11.2 Constructor

The following constructors are defined:

► `public XYImageAnnotation(double x, double y, Image image);`  
 Equivalent to `XYImageAnnotation(x, y, image, RectangleAnchor.CENTER)`—see the next constructor.  
 ► `public XYImageAnnotation(double x, double y, Image image, RectangleAnchor anchor); [1.0.4]`  
 Creates an annotation that will display the specified `image` at the given  $(x, y)$  location. The coordinates are specified in data-space (that is, the axis coordinates of the chart) and the image will be aligned to this point according to the `anchor` setting. If `image` or `anchor` is null, this method throws an `IllegalArgumentException`.

### 23.11.3 Attributes

All the attributes for this class are specified via the constructor and cannot be updated:

► `public double getX(); [1.0.4]`  
 Returns the x-coordinate for the anchor point to which the image will be aligned.  
 ► `public double getY(); [1.0.4]`  
 Returns the y-coordinate for the anchor point to which the image will be aligned.  
 ► `public Image getImage(); [1.0.4]`  
 Returns the image to be displayed by this renderer.  
 ► `public RectangleAnchor getImageAnchor(); [1.0.4]`  
 Returns the image anchor, which will be aligned to the  $(x, y)$  location on the chart when the image annotation is displayed.

### 23.11.4 Drawing

Once an annotation has been added to a plot, the plot will take care of drawing it every time the chart is redrawn. The following method is used:

```
► public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea,
ValueAxis domainAxis, ValueAxis rangeAxis);
```

Draws the annotation within the specified `dataArea`. This method is called by the plot, you shouldn't need to call it yourself.

### 23.11.5 Equals, Cloning and Serialization

This class overrides the `equals()` method specified in `Object`:

```
► public boolean equals(Object object);
```

Tests this annotation for equality with an arbitrary object. This method will return `true` if `object` is an instance of `XYImageAnnotation` with the same coordinates and image as this annotation.

The annotation can be cloned:

```
► public Object clone() throws CloneNotSupportedException;
```

Returns a clone of the annotation.

At present, serialization is not supported because images are not automatically serializable. Hopefully this will be fixed in a future release by writing our own image serialization code (for instance, by writing the image data to PNG format, then decoding it again upon deserialization).

### 23.11.6 Notes

Some points to note:

- the `PlotOrientationDemo1` application (source code is included in the JFreeChart Demo distribution) includes an image annotation for each sub-chart.

#### See Also

[XYAnnotation](#).

## 23.12 XYLineAnnotation

### 23.12.1 Overview

A simple annotation that draws a line between a starting point ( $x_0, y_0$ ) and an ending point ( $x_1, y_1$ ) on an `XYPlot`. To add a line annotation to a plot, use code similar to the following:

```
XYPlot plot = (XYPlot) chart.getPlot();
XYLineAnnotation a1 = new XYLineAnnotation(1.0, 2.0, 3.0, 4.0,
new BasicStroke(1.5f), Color.red);
plot.addAnnotation(a1);
```

### 23.12.2 Constructors

To create a new annotation:

```
► public XYLineAnnotation(double x1, double y1, double x2, double y2);
```

Creates an annotation that will draw a line from ( $x_1, y_1$ ) to ( $x_2, y_2$ ) on the chart. By default, the line is black and uses a stroke width of 1.0.

```
► public XYLineAnnotation(double x1, double y1, double x2, double y2,
Stroke stroke, Paint paint);
```

Creates an annotation that will draw a line from ( $x_1, y_1$ ) to ( $x_2, y_2$ ) on the chart. The line is drawn using the specified `stroke` and `paint`.

### 23.12.3 Drawing

Once an annotation has been added to a plot, the plot will take care of drawing it every time the chart is redrawn. The following method is used:

```
► public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea, ValueAxis domainAxis,
ValueAxis rangeAxis);
```

Draws the annotation within the specified `dataArea`. This method is called by the plot, you shouldn't need to call it yourself.

### 23.12.4 Equals, Cloning and Serialization

This class overrides the `equals()` method specified in `Object`:

```
► public boolean equals(Object object);
```

Tests this annotation for equality with an arbitrary object. This method will return `true` if `object` is an instance of `XYLineAnnotation` with the same coordinates, stroke and paint settings as this annotation.

The annotation can be cloned:

```
► public Object clone() throws CloneNotSupportedException;
```

Returns a clone of the annotation.

This class is `Serializable`.

### 23.12.5 Notes

Some points to note:

- if you want to use a line annotation on a time series chart, the x-coordinates of the annotation should be specified in “milliseconds since 1-Jan-1970, GMT”.

## See Also

[XYAnnotation](#).

## 23.13 XYPointerAnnotation

### 23.13.1 Overview

An annotation that displays an arrow pointing towards a specific  $(x, y)$  location on an `XYPlot` (see figure 23.4). The arrow can have a label at one end.



Figure 23.4: An `XYPointerAnnotation` example

### 23.13.2 Usage

To add a pointer annotation to an `XYPlot`:

```
XYPlot plot = (XYPlot) chart.getPlot();
XYPointerAnnotation pointer = new XYPointerAnnotation(
    "Best Bid", millis, 163.0, 3.0 * Math.PI / 4.0
);
pointer.setTipRadius(10.0);
pointer.setBaseRadius(35.0);
pointer.setFont(new Font("SansSerif", Font.PLAIN, 9));
pointer.setPaint(Color.blue);
pointer.setTextAnchor(TextAnchor.HALF_ASCENT_RIGHT);
plot.addAnnotation(pointer);
```

### 23.13.3 Constructor

To create a new pointer annotation:

```
➔ public XYPointerAnnotation(String label, double x, double y, double angle);
```

Creates a new pointer annotation to highlight the specified  $(x, y)$  location on the chart.

### 23.13.4 General Attributes

To control the angle of the arrow:

```
➔ public double getAngle();
```

Returns the angle of the arrow (in radians).

```
➔ public void setAngle(double angle);
```

Sets the angle of the arrow (in radians). If you imagine a clockface, an angle of 0 results in an arrow pointing from 3 o'clock to the center of the clock face, with positive values proceeding from 3 o'clock in a clockwise direction.

To control the distance between the  $(x, y)$  location and the tip of the arrow:

```
➔ public double getTipRadius();
```

Returns the radius of the circle that determines how far from the  $(x, y)$  location the tip of the arrow is.

```
➔ public void setTipRadius(double radius);
```

Sets the radius of the circle that determines the end point of the arrow.

To control the distance between the  $(x, y)$  location and the base of the arrow:

```
➔ public double getBaseRadius();
```

Returns the radius of the circle that determines how far from the  $(x, y)$  location to the base of the arrow.

```
➔ public void setBaseRadius(double radius);
```

Sets the radius of the circle that determines the base point for the arrow.

To control the offset between the base of the arrow and the label anchor point:

```
➔ public double getLabelOffset();
```

Returns the label offset (in Java2D units).

```
➔ public void setLabelOffset(double offset);
```

Sets the label offset from the base of the arrow (in Java2D units).

To control the length of the arrow head:

```
➔ public double getArrowLength();
```

Returns the length of the arrow head (in Java2D units).

```
➔ public void setArrowLength(double length);
```

Sets the length of the arrow head (in Java2D units).

To control the width of the arrow head:

```
➔ public double getArrowWidth();
```

Returns the width of the arrow head in Java2D units.

```
➔ public void setArrowWidth(double width);
```

Sets the width of the arrow head in Java2D units.

To control the **Stroke** used to draw the arrow:

```
➔ public Stroke getArrowStroke();
```

Returns the stroke used to draw the arrow (never `null`).

```
➔ public void setArrowStroke(Stroke stroke);
```

Sets the stroke used to draw the arrow (`null` not permitted).

To control the **Paint** used to draw the arrow:

```
➔ public Paint getArrowPaint();
```

Returns the paint used to draw the arrow (never `null`).

```
➔ public void setArrowPaint(Paint paint);
```

Sets the paint used to draw the arrow (`null` not permitted).

### 23.13.5 Other Methods

To draw the annotation (this method is called by the plot, you shouldn't need to call it directly yourself):

```
→ public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea, ValueAxis domainAxis,
ValueAxis rangeAxis);
Draws the annotation.
```

### 23.13.6 Notes

Some points to note:

- annotations don't currently have a change notification mechanism, so charts do not automatically refresh when an annotation is modified;
- a demo (`XYPolygonAnnotationDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[XYAnnotation](#).

## 23.14 XYPolygonAnnotation

### 23.14.1 Overview

A simple annotation that draws a polygon on an `XYPlot`. The polygon's coordinates are specified in *data space* (that is, the coordinate system defined by the plot's axes). See figure 23.5 for an example.

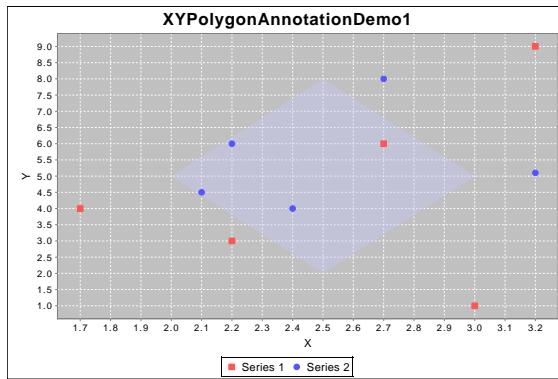


Figure 23.5: An `XYPolygonAnnotation` (see `XYPolygonAnnotationDemo1.java`)

### 23.14.2 Usage

A demo is provided by `XYPolygonAnnotationDemo1.java`.

### 23.14.3 Constructors

There are several constructors for this class. For each one, the first argument is an array containing the  $(x, y)$  coordinates of the polygon's vertices. These coordinates should be specified using the coordinate system defined by the chart's axes.

```
➔ public XYPolygonAnnotation(double[] polygon);
Creates a new annotation that draws a polygon with the supplied coordinates. The polygon
will be drawn with a black outline, one Java2D unit wide. The polygon is not filled.

➔ public XYPolygonAnnotation(double[] polygon, Stroke stroke, Paint outlinePaint)
Creates a new annotation that draws the specified polygon with the given stroke and outlinePaint.
The polygon is not filled.

➔ public XYPolygonAnnotation(double[] polygon, Stroke stroke, Paint outlinePaint,
Paint fillPaint);
Creates a new annotation that draws a polygon with the specified vertices, using the supplied
stroke, outlinePaint and fillPaint.
```

For all constructors, the `polygon` array must contain an even number of items, since it contains a sequence of  $(x, y)$  coordinates.

### 23.14.4 Methods

To get the coordinates of the polygon:

```
➔ public double[] getPolygonCoordinates(); [1.0.2]
Returns the coordinates of the polygon's vertices.
```

To get the outline attributes for the annotation:

```
➔ public Stroke getOutlineStroke(); [1.0.2]
Returns the stroke used to draw the outline of the polygon. If this is null, no outline is drawn.

➔ public Paint getOutlinePaint(); [1.0.2]
Returns the paint used to draw the outline of the polygon. If this is null, no outline is drawn.
```

To get the fill paint for the annotation:

```
➔ public Paint getFillPaint(); [1.0.2]
Returns the paint used to fill the polygon. If this is null, the polygon is not filled.
```

The annotation is drawn (by the plot) using this method (which you shouldn't need to call yourself):

```
➔ public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea,
ValueAxis domainAxis, ValueAxis rangeAxis, int rendererIndex,
PlotRenderingInfo info);
Draws the annotation within the specified dataArea.
```

### 23.14.5 Equals, Cloning and Serialization

To test this class for equality with an arbitrary object:

```
➔ public boolean equals(Object obj);
Returns true if this annotation is equal to the specified obj. This method will return true if
and only if:


- obj is not null;
- obj is an instance of XYPolygonAnnotation;
- obj defines a polygon with the same vertices (in the same order) as this annotation;
- obj has the same stroke, outlinePaint and fillPaint as this annotation;

```

This class is cloneable and implements the `PublicCloneable` interface. This class is also serializable.

### 23.14.6 Notes

Some points to note:

- the polygon annotation will only be visible on a chart if it falls within the current axis bounds;
- for a demo, see `XYPolygonAnnotationDemo1.java`.

**See Also**[XYAnnotation](#).

## 23.15 XYShapeAnnotation

### 23.15.1 Overview

A simple annotation that draws a shape on an [XYPlot](#). The shape's coordinates are specified in “data space” (that is, the coordinate system defined by the plot's axes).

### 23.15.2 Constructors

This class has several constructors. The attributes of the annotation are specified via the constructor and cannot be modified subsequently:

- ➔ `public XYShapeAnnotation(Shape shape);`  
Creates a new annotation that will draw the given shape with a black outline.
- ➔ `public XYShapeAnnotation(Shape shape, Stroke stroke, Paint outlinePaint);`  
Creates a new annotation that will draw the given shape with the specified stroke and outline paint.
- ➔ `public XYShapeAnnotation(Shape shape, Stroke stroke, Paint outlinePaint, Paint fillPaint);`  
Creates a new annotation that will draw the given shape with the specified stroke, outline paint and fill paint. If either the `stroke` or `outlinePaint` is `null`, no outline is drawn. If the `fillPaint` is `null`, the shape is not filled.

### 23.15.3 Methods

There are no methods for setting the attributes of the annotation—these are set in the constructor and cannot be modified.

The following method is called by the plot to draw the annotation, normally you won't need to call it directly:

- ➔ `public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea, ValueAxis domainAxis, ValueAxis rangeAxis, int rendererIndex, PlotRenderingInfo info);`  
Draws the annotation.

### 23.15.4 Equals, Cloning and Serialization

To test this annotation for equality with an arbitrary object:

- ➔ `public boolean equals(Object obj);`  
Tests the annotation for equality with `obj`. This method returns `true` if and only if:
  - `obj` is not `null`;
  - `obj` is an instance of [XYShapeAnnotation](#);
  - `obj` has the same attributes as this annotation.

This class is [Cloneable](#)<sup>2</sup> and [Serializable](#).

### 23.15.5 Notes

Before drawing, the shape must be transformed to Java2D coordinates. The transformation code assumes linear scales on the axes, so this type of annotation may not work well with logarithmic axes.

---

<sup>2</sup>Technically, this probably isn't necessary since instances of this class are immutable.

**See Also**

[XYAnnotation](#).

## 23.16 XYTextAnnotation

### 23.16.1 Overview

A text annotation that can be added to an [XYPlot](#). You can use this class to add a small text label at some  $(x, y)$  location on a chart. This class is a subclass of [AbstractXYAnnotation](#).

### 23.16.2 Usage

To add a simple annotation to an [XYPlot](#):

```
XYPlot plot = (XYPlot) chart.getPlot();
XYAnnotation annotation = new XYTextAnnotation("Hello World!", 10.0, 25.0);
plot.addAnnotation(annotation);
```

The text will be centered on the specified  $(x, y)$  location.

### 23.16.3 Constructors

To create a new annotation:

► public XYTextAnnotation(String text, double x, double y);  
Creates a new text annotation for display at the specified  $(x, y)$  location (in data space). An exception is thrown if the `text` argument is `null`.

### 23.16.4 General Attributes

To control the text for the annotation:

► public String getText();  
Returns the text displayed by the annotation (never `null`). The initial value is specified in the constructor.  
  
► public void setText(String text);  
Sets the text for the annotation (no event is generated). If `text` is `null`, this method throws an `IllegalArgumentException`.

To control the location on the chart that the annotation will be aligned to:

► public double getX();  
Returns the x-coordinate (in data space).  
  
► public void setX(double x);  
Sets the x-coordinate (in data space) for the annotation. No event is generated.  
  
► public double getY();  
Returns the y-coordinate (in data space).  
  
► public void setY(double y);  
Sets the y-coordinate (in data space) for the annotation. No event is generated.

To control the font used to display the text annotation:

► public Font getFont();  
Returns the font used to display the text annotation (never `null`). The default value is `Font("SansSerif", Font.PLAIN, 10)`.  
  
► public void setFont(Font font);  
Sets the font used to display the text annotation (no event is generated). If `font` is `null`, this method throws an `IllegalArgumentException`.

To control the paint used to display the text annotation:

```
► public Paint getPaint();
```

Returns the paint used to display the text (never `null`). The default value is `Color.black`.

```
► public void setPaint(Paint paint);
```

Sets the paint used to display the text annotation (no event is generated). If `paint` is `null`, this method throws an `IllegalArgumentException`.

The *text anchor* defines a point on the text's framing rectangle that will be aligned to the  $(x, y)$  location on the chart:

```
► public TextAnchor getTextAnchor();
```

Returns the text anchor (never `null`). This is a point on the framing rectangle for the text that is aligned to the  $(x, y)$  location on the chart. The default value is `TextAnchor.CENTER` (in other words, the text annotation will be centered over the  $(x, y)$  location).

```
► public void setTextAnchor(TextAnchor anchor)
```

Sets the text anchor (no event is generated). If `anchor` is `null`, this method throws an `IllegalArgumentException`.

The *rotation anchor* defines a point on the text's framing rectangle about which the text will be rotated:

```
► public TextAnchor getRotationAnchor();
```

Returns the rotation anchor (never `null`). The default value is `TextAnchor.CENTER`.

```
► public void setRotationAnchor(TextAnchor anchor);
```

Sets the rotation anchor (no event is generated). If `anchor` is `null`, this method throws an `IllegalArgumentException`.

To control the rotation angle:

```
► public double getRotationAngle();
```

Returns the rotation angle, which is measured in radians (clockwise). The default value is `0.0`.

```
► public void setRotationAngle(double angle);
```

Sets the rotation angle in radians (no event is generated).

### 23.16.5 Other Methods

The following method is used to draw the annotation. It is called by the plot, you won't normally need to call this method yourself:

```
► public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea,  
ValueAxis domainAxis, ValueAxis rangeAxis);
```

Draws the annotation.

### 23.16.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
```

Tests this annotation for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 23.16.7 Notes

Some points to note:

- annotations added directly to the plot are positioned relative to the primary axes. You can also add the annotation to an `XYItemRenderer`, in which case the annotation is drawn relative to the axes used by that renderer;
- a demo (`AnnotationDemo1.java`) is included in the JFreeChart demo collection;
- the `XYPointerAnnotation` subclass can be used to display a label with an arrow pointing to some  $(x, y)$  value.

**See Also**

[XYAnnotation](#).

# Chapter 24

## Package: org.jfree.chart.axis

### 24.1 Overview

This package contains all the axis classes plus a few assorted support classes and interfaces:

- the `CategoryPlot` and `XYPlot` classes maintain references to two axes (by default), which we refer to as the *domain axis* and *range axis*. These terms are based on the idea that these plots are providing a visual representation of a function that maps a set of *domain values* onto a set of *range values*. For most purposes, you can think of the domain axis as the *X-axis* and the range axis as the *Y-axis*, but we prefer the more generic terms.
- the default settings provided by the axis classes should work well for a wide range of applications. However, there are many ways to customise the behaviour of the axes by modifying attributes via the JFreeChart API. Be sure to read through the API documentation to become familiar with the options that are available.
- a powerful feature of JFreeChart is the support for multiple domain and range axes on a single plot. If you plan to make use of this feature, you should refer to section 13 for more information.

The axis classes are `Cloneable` and `Serializable`.

### 24.2 Axis

#### 24.2.1 Overview

An abstract base class representing an axis. Some subclasses of `Plot`, including `CategoryPlot` and `XYPlot`, will use axes to display data.

Figure 24.1 illustrates the axis class hierarchy.

#### 24.2.2 Constructors

The constructors for this class are `protected`, you cannot create an instance of this class directly—you must use a subclass.

#### 24.2.3 Attributes

The attributes maintained by the `Axis` class are listed in Table 24.1. There are methods to read and update most of these attributes. In most cases, updating an axis attribute will result in an `AxisChangeEvent` being sent to all (or any) registered listeners.

The default values used to initialise the axis attributes are listed in Table 24.2.

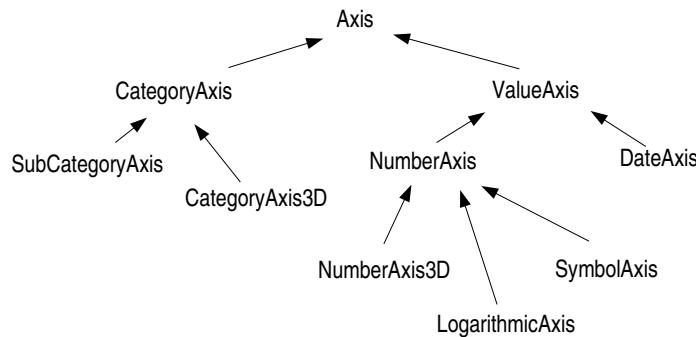


Figure 24.1: Axis classes

Attribute:	Description:
<code>plot</code>	The plot to which the axis belongs.
<code>visible</code>	A flag that controls whether or not the axis is visible.
<code>label</code>	The axis label.
<code>labelFont</code>	The font for the axis label.
<code>labelPaint</code>	The foreground colour for the axis label.
<code>labelInsets</code>	The space to leave around the outside of the axis label.
<code>labelAngle</code>	The angle of rotation for the axis label.
<code>axisLineVisible</code>	A flag that controls whether or not a line is drawn for the axis.
<code>axisLinePaint</code>	The paint used to draw the axis line if it is visible.
<code>axisLineStroke</code>	The stroke used to draw the axis line if it is visible.
<code>tickLabelsVisible</code>	A flag controlling the visibility of tick labels.
<code>tickLabelFont</code>	The font for the tick labels.
<code>tickLabelPaint</code>	The color for the tick labels.
<code>tickLabelInsets</code>	The space to leave around the outside of the tick labels.
<code>tickMarksVisible</code>	A flag controlling the visibility of tick marks.
<code>tickMarkStroke</code>	The stroke used to draw the tick marks.
<code>tickMarkPaint</code>	The paint used to draw the tick marks.
<code>tickMarkInsideLength</code>	The amount by which the tick marks extend into the plot area (in Java2D units).
<code>tickMarkOutsideLength</code>	The amount by which the tick marks extend outside the plot area (in Java2D units).

Table 24.1: Attributes for the `Axis` class

#### 24.2.4 Usage

To change the attributes of an axis, you must first obtain a reference to the axis. Usually, you will obtain the reference from the plot that uses the axis. For example:

```

CategoryPlot plot = (CategoryPlot) chart.getPlot();
CategoryAxis axis = plot.getDomainAxis();
// change axis attributes here...
  
```

Notice that the `getDomainAxis()` method returns a particular subclass of `Axis` (`CategoryAxis` in this case). That's okay, because the subclass inherits all the attributes defined by `Axis` anyway.

#### 24.2.5 The Axis Label

The axis label typically describes what an axis is measuring (for example, "Sales in US\$"). To access the axis label:

```

public String getLabel();
Returns the axis label (possibly null).
  
```

Name:	Value:
DEFAULT_AXIS_LABEL_FONT	<code>new Font("SansSerif", Font.PLAIN, 14);</code>
DEFAULT_AXIS_LABEL_PAINT	<code>Color.black;</code>
DEFAULT_AXIS_LABEL_INSETS	<code>new Insets(2, 2, 2, 2);</code>
DEFAULT_TICK_LABEL_FONT	<code>new Font("SansSerif", Font.PLAIN, 10);</code>
DEFAULT_TICK_LABEL_PAINT	<code>Color.black;</code>
DEFAULT_TICK_LABEL_INSETS	<code>new Insets(2, 1, 2, 1);</code>
DEFAULT_TICK_STROKE	<code>new BasicStroke(1);</code>

Table 24.2: Axis class default attribute values

► `public void setLabel(String label);`

Sets the axis label and sends an `AxisChangeEvent` to all registered listeners. If you set the label to `null`, no label is displayed for the axis.

To access the font used to display the axis label:

► `public Font getLabelFont();`

Returns the `Font` used to display the axis label.

► `public void setLabelFont(Font font);`

Sets the `Font` used to display the axis label and sends an `AxisChangeEvent` to all registered listeners.

To access the paint used to display the axis label:

► `public Paint getLabelPaint();`

Returns the paint used to display the axis label.

► `public void setLabelPaint(Paint paint);`

Sets the paint used to display the axis label and sends an `AxisChangeEvent` to all registered listeners.

To control the rotation angle for the axis label:

► `public double getLabelAngle();`

Returns the angle of rotation (in radians) for the axis label. The default value is `0.0`.

► `public void setLabelAngle(double angle);`

Sets the angle of rotation for the axis label and sends an `AxisChangeEvent` to all registered listeners. The angle is specified in radians.

#### 24.2.6 The Axis Line

The axis draws a line along the edge of the plot's *data area*:

► `public boolean isAxisLineVisible();`

Returns `true` if the axis draws a line along the edge of the data area, and `false` otherwise. The default value is `true`.

► `public void setAxisLineVisible(boolean visible);`

Sets the flag that controls whether or not a line is drawn along the edge of the data area by the axis, and sends an `AxisChangeEvent` to all registered listeners.

The stroke used to draw the axis line (if it is visible) is controlled by the following methods:

► `public Stroke getAxisLineStroke();`

Returns the stroke used to draw the axis line (never `null`). The default value is `BasicStroke(1.0f)`.

► `public void setAxisLineStroke(Stroke stroke);`

Sets the stroke used to draw the axis line and sends an `AxisChangeEvent` to all registered listeners. If `stroke` is `null`, this method throws an `IllegalArgumentException`.

The paint used to draw the axis line (if it is visible) is controlled by the following methods:

► `public Paint getAxisLinePaint();`

Returns the paint used to draw the axis line (never `null`). The default value is `Color.GRAY`.

► `public void setAxisLinePaint(Paint paint);`

Sets the paint used to draw the axis line and sends an `AxisChangeEvent` to all registered listeners.

If `paint` is `null`, this method throws an `IllegalArgumentException`.

Note that the `CategoryPlot` and `XYPlot` classes also draw an outline around the data area. The outline is drawn before (under) the axis line(s). The plot outline stroke and paint are defined in the `Plot` class.

#### 24.2.7 Tick Marks and Labels

It is common for axes to have small marks at regular intervals to show the scale of values displayed by the axis. In JFreeChart, we refer to these marks as “tick marks”, and the labels corresponding to these marks as “tick labels”. This class defines the basic attributes that control the appearance of tick marks and labels, but leaves the actual generation and formatting up to specific subclasses.

To control the visibility of the tick marks for an axis:

► `public boolean isTickMarksVisible();`

Returns the flag that controls whether or not the tick marks are visible. The default value is `true`.

► `public void setTickMarksVisible(boolean flag);`

Sets the flag that controls whether or not tick marks are visible, then sends an `AxisChangeEvent` to all registered listeners.

To control the stroke used to draw the tick marks:

► `public Stroke getTickMarkStroke();`

Returns the stroke used to draw the tick marks (never `null`). The default value is `BasicStroke(1.0f)`.

► `public void setTickMarkStroke(Stroke stroke);`

Sets the stroke used to draw the tick marks (`null` not permitted) then sends an `AxisChangeEvent` to all registered listeners.

To control the paint used to draw the tick marks:

► `public Paint getTickMarkPaint();`

Returns the paint used to draw the tick marks (never `null`). The default value is `Color.black`.

► `public void setTickMarkPaint(Paint paint);`

Sets the paint used to draw the tick marks (`null` not permitted) then sends an `AxisChangeEvent` to all registered listeners.

To control the length of the tick marks, you can set the “inside” and “outside” lengths:

► `public float getTickMarkInsideLength();`

Returns the length of the tick mark on the inside of the data area, in Java2D units. The default value is `0.0f`.

► `public void setTickMarkInsideLength(float length);`

Sets the length of the tick mark on the inside of the data area, in Java2D units, and sends an `AxisChangeEvent` to all registered listeners.

► `public float getTickMarkOutsideLength();`

Returns the length of the tick mark extension into the plot area, in Java2D units. The default value is `2.0f`.

► `public void setTickMarkOutsideLength(float length);`

Sets the length of the tick mark on the outside of the data area, in Java2D units, and sends an `AxisChangeEvent` to all registered listeners.

To control the visibility of the tick labels for an axis:

► public boolean isTickLabelsVisible();

Returns the flag that controls whether or not the tick labels are visible. The default value is `true`.

► public void setTickLabelsVisible(boolean flag);

Sets the flag that controls whether or not the tick labels are visible and sends an `AxisChangeEvent` to all registered listeners.

To control the font used to draw the tick labels:

► public Font getTickLabelFont();

Returns the tick label font (never `null`). The default value is `Font("SansSerif", Font.PLAIN, 10)`.

► public void setTickLabelFont(Font font);

Sets the tick label font and sends an `AxisChangeEvent` to all registered listeners. If `font` is `null`, this method throws an `IllegalArgumentException`.

To control the paint used to draw the tick labels:

► public Paint getTickLabelPaint();

Returns the tick label paint. The default value is `Color.black`.

► public void setTickLabelPaint(Paint paint);

Sets the tick label paint and sends an `AxisChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

#### 24.2.8 The Fixed Dimension

It is possible to specify a fixed “dimension” for an axis. This is an ugly hack to help align subplots in the combined plots. For a vertical axis, the fixed dimension applies to the width of the axis and for a horizontal axis the fixed dimension applies to the height of the axis.

► public double getFixedDimension();

Returns the fixed dimension for the axis, in Java2D units.

► public void setFixedDimension(double dimension);

Sets the fixed dimension for the axis, in Java2D units. During layout, if the axis width or height (depending on the orientation) is less than this value, it is increased to match `dimension`. The value defaults to 0.0 which means it is ignored.

Note that the `CategoryAxis` class completely ignores this setting.

#### 24.2.9 Other Methods

All axes are drawn by the plot that owns the axis, using this method:

► public abstract AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea, Rectangle2D dataArea, RectangleEdge edge);

Draws the axis along the specified edge of the data area. Given that there may be more than one axis on a particular edge, the cursor value specifies the distance from the edge that the axis should be drawn (to take account of other axes that have already been drawn). An `AxisState` object is returned which provides information about the axis (for example, the tick values which the plot will use to draw gridlines if they are visible).

All axes are given the opportunity to refresh the axis ticks during the drawing process, which allows for dynamic adjustment depending on the amount of space available for drawing the axis:

► public abstract List refreshTicks(Graphics2D g2, AxisState state, Rectangle2D plotArea, Rectangle2D dataArea, RectangleEdge edge);

Creates a list of ticks for the axis and updates the axis state.

### 24.2.10 Change Notification

This class implements a *change notification mechanism* that is used to notify other objects whenever an axis is changed in some way. This is part of a JFreeChart-wide mechanism that makes it possible to receive notifications whenever a component of a chart is changed. Most often, such notifications result in the chart being redrawn.

The following methods are used:

- `public void addChangeListener(AxisChangeListener listener);`  
Registers an object to receive notification whenever the axis changes.
- `public void removeChangeListener(AxisChangeListener listener);`  
Deregisters an object, so that it no longer receives notification when the axis changes.
- `public void notifyListeners(AxisChangeEvent event);`  
Notifies all registered listeners that a change has been made to the axis.

#### See Also

[AxisChangeEvent](#), [AxisChangeListener](#), [CategoryAxis](#), [DateAxis](#), [NumberAxis](#).

## 24.3 AxisCollection

### 24.3.1 Overview

A storage structure that is used to record the axes that have been assigned to the top, bottom, left and right sides of a plot.

### 24.3.2 Notes

Axis collections are maintained only temporarily during the process of drawing a chart.

## 24.4 AxisLocation

### 24.4.1 Overview

This class is used to represent the possible axis locations for a 2D chart:

- `AxisLocation.TOP_OR_LEFT;`
- `AxisLocation.TOP_OR_RIGHT;`
- `AxisLocation.BOTTOM_OR_LEFT;`
- `AxisLocation.BOTTOM_OR_RIGHT;`

The final position of the axis is dependent on the orientation of the plot (horizontal or vertical) and whether the axis is being used as a domain or a range axis.

### 24.4.2 Notes

The axis location is set using methods in the [CategoryPlot](#) and [XYPlot](#) classes.

## 24.5 AxisSpace

### 24.5.1 Overview

This class is used to record the amount of space (in Java2D units) required to display the axes around the edges of a plot. Since the plot may contain many axes (or, in the most complex case, many subplots containing many axes) this class is used to collate the space requirements for all the axes.

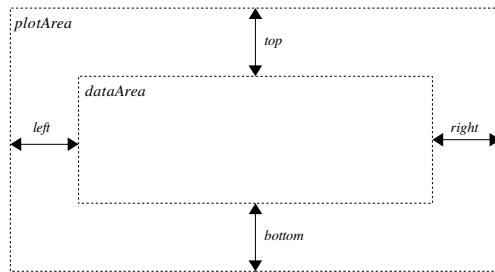


Figure 24.2: *AxisSpace* Attributes

Axes are always drawn around the edges of the *data area* but should never extend outside the *plot area*.

### 24.5.2 Methods

There are methods to get and set each of the attributes `top`, `bottom`, `left` and `right` maintained by this class.

To *add* space to a particular edge:

```
→ public void add(double space, RectangleEdge edge);
    Adds the specified amount of space (in Java2D units) to one edge.
```

Sometimes you want to ensure that there is *at least* a specified amount of space for the axis along a particular edge (this is used to ensure that the data areas in combined plots are aligned). The following methods achieve this:

```
→ public void ensureAtLeast(double space, RectangleEdge edge);
    Ensures that there is at least the specified amount of space for the axes along the specified edge.

→ public void ensureAtLeast(AxisSpace space);
    As above, but applied to all the edges.
```

Given a rectangle and an instance of `AxisSpace`, you can calculate the size of an inner rectangle (essentially this is how the data area is computed from the plot area):

```
→ public Rectangle2D shrink(Rectangle2D area, Rectangle2D result);
    Calculates an inner rectangle based on the current space settings. If result is null a new
    Rectangle2D is created for the result, otherwise the supplied rectangle is recycled.
```

## 24.6 AxisState

### 24.6.1 Overview

Instances of this class are used to record state information for an axis during the process of drawing the axis to some output target.

## 24.6.2 Notes

By recording state information *per drawing* of an axis, it should be possible for separate threads to draw the same axis to different output targets simultaneously without interfering with one another. This is part of an effort to (eventually) make JFreeChart thread-safe.

# 24.7 CategoryAnchor

## 24.7.1 Overview

An enumeration of the anchor points within the space allocated for a single category on a [CategoryAxis](#):

Default:	Value:
<code>CategoryAnchor.START</code>	The start of the category.
<code>CategoryAnchor.MIDDLE</code>	The middle of the category.
<code>CategoryAnchor.END</code>	The end of the category.

## 24.7.2 Usage

This class is used to control the position of the domain axis gridlines drawn in a [CategoryPlot](#) (see the `setDomainGridlinePosition()` method).

# 24.8 CategoryAxis

## 24.8.1 Overview

A *category axis* is used as the domain axis in a [CategoryPlot](#). Categories are displayed at regular intervals along the axis, with a gap before the first category (the *lower margin*), a gap after the last category (the *upper margin*) and a gap between each category (the *category margin*).



Figure 24.3: The CategoryAxis margins

The axis will usually display a label for each category. There are a range of options for controlling the position, alignment and rotation of the labels—these are described in section [24.8.6](#).

## 24.8.2 Constructor

This class has two constructors:

► `public CategoryAxis();`  
Equivalent to `CategoryAxis(null)`—see the next constructor.

► `public CategoryAxis(String label);`  
Creates a new category axis with the specified label. If you prefer no axis label, you can use `null` for the `label` argument. All other attributes are set to default values.

Attribute:	Description:
<i>lowerMargin</i>	The margin that appears before the first category, expressed as a percentage of the overall axis length (defaults to 0.05 or five percent).
<i>upperMargin</i>	The margin that appears after the last category, expressed as a percentage of the overall axis length (defaults to 0.05 or five percent).
<i>categoryMargin</i>	The margin between categories, expressed as a percentage of the overall axis length (to be distributed between $N-1$ gaps, where $N$ is the number of categories). The default value is 0.20 (twenty percent).
<i>categoryLabelPositionOffset</i> <i>categoryLabelPositions</i>	The offset between the axis line and the category labels. A structure that defines label positioning information for each possible axis location (the axis may be located at the top, bottom, left or right of the plot).

Table 24.3: Attributes for the *CategoryAxis* class

### 24.8.3 Attributes

The attributes maintained by the *CategoryAxis* class are listed in Table 24.3. These attributes are in addition to those inherited from the [Axis](#) class (see section 24.2.3 for details). The following default values are used:

Default:	Value:
DEFAULT_AXIS_MARGIN	0.05 (5 percent).
DEFAULT_CATEGORY_MARGIN	0.20 (20 percent).

### 24.8.4 Setting Axis Margins

To control the lower margin for the axis:

```
➔ public double getLowerMargin();  
Returns the lower margin for the axis, as a percentage of the total axis length. The default value is 0.05 (five percent).  
➔ public void setLowerMargin(double margin);  
Sets the lower margin for the axis and sends an AxisChangeEvent to all registered listeners. The margin is a percentage of the axis length (for example, 0.05 for a five percent margin).
```

To control the upper margin for the axis:

```
➔ public double getUpperMargin();  
Returns the upper margin for the axis, as a percentage of the total axis length. The default value is 0.05 (five percent).  
➔ public void setUpperMargin(double margin);  
Sets the upper margin for the axis and sends an AxisChangeEvent to all registered listeners. The margin is a percentage of the axis length (for example, 0.05 for a five percent margin).
```

To control the margin between categories:

```
➔ public double getCategoryMargin();  
Returns the (total) margin between all categories, as a percentage of the total axis length. The default value is 0.20 (that is, twenty percent of the axis length is allocated to the gaps between categories).  
➔ public void setCategoryMargin(double margin);  
Sets the category margin for the axis and sends an AxisChangeEvent to all registered listeners. The margin is a percentage of the axis length (for example, 0.20 for a twenty percent margin). The overall margin is distributed over  $N-1$  gaps where  $N$  is the number of categories displayed on the axis.
```

### 24.8.5 Category Labels

The labels displayed on the axis to represent each category are obtained directly from the dataset, by calling the `toString()` method on the dataset's column key (similarly, the series label displayed in the legend is obtained by calling the `toString()` method on the row key).

There are many options available for positioning, aligning and rotating the category labels on the axis—these options are described in more detail in the next section. Here, we simply describe the technique for rotating the category labels by 90 degrees, which is a common requirement:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
CategoryAxis axis = plot.getDomainAxis();
axis.setCategoryLabelPositions(CategoryLabelPositions.UP_90);
```

### 24.8.6 Category Label Positioning and Alignment

There are many options for controlling the positioning, alignment and rotation of category labels. This provides a great deal of flexibility, but at the price of being somewhat complex.

By default, JFreeChart will display category labels on a single line, truncated if necessary. However, multi-line labels are supported:

```
► public int getMaximumCategoryLabelLines();
Returns the current maximum number of lines for displaying category labels. The default value is 1.

► public void setMaximumCategoryLabelLines(int lines);
Sets the maximum number of lines for displaying category labels and sends an AxisChangeEvent to all registered listeners.
```

Line wrapping occurs when longer labels reach the maximum width allowed for category labels. This maximum category label width is specified in a relative way, in the `CategoryLabelPosition` class. In addition, there is an override setting in this class:

```
► public float getMaximumCategoryLabelWidthRatio();
Returns the maximum category label width setting, which is expressed as a percentage of either (a) the category label rectangle, or (b) the length of the range axis. The default value is 0.0, which means this override setting is ignored.

► public void setMaximumCategoryLabelWidthRatio(float ratio);
Sets the maximum category label width, expressed as a percentage of (a) the category label rectangle, or (b) the length of the range axis. This setting overrides the value specified in the CategoryLabelPosition class (see below). After setting the value, an AxisChangeEvent is sent to all registered listeners.
```

To control the offset between the axis and the category labels:

```
► public int getCategoryLabelPositionOffset();
Returns the offset (in Java2D units) between the axis and the category labels. The default value is 4.

► public void setCategoryLabelPositionOffset(int offset);
Sets the offset, in Java2D units, between the axis and the category labels, then sends an AxisChangeEvent to all registered listeners.
```

To control the position and rotation of the category labels:

```
► public CategoryLabelPositions getCategoryLabelPositions();
Returns an object containing the four CategoryLabelPosition instances that apply for each possible location of the axis. This method never returns null.

► public void setCategoryLabelPositions(CategoryLabelPositions positions);
Sets the attribute that controls the position, alignment and rotation of the category labels along the axis, then sends an AxisChangeEvent to all registered listeners.
```

The `CategoryLabelPositions` class is just a structure containing four instances of the `CategoryLabelPosition` class. When the axis needs to determine where it is going to draw the category labels, it will select one of those instances depending on the current location of the axis (at the top, bottom, left or right of the plot). It is the attributes of the `CategoryLabelPosition` object that ultimately determine where the labels are drawn.

- the first attribute is an anchor point relative to a notional category rectangle that is computed by the axis (see figure 24.4). Within this rectangle, an *anchor point* is specified using the `RectangleAnchor` class.



Figure 24.4: A category label rectangle

- the second attribute is a text anchor, which defines a point on the category label which is aligned with the anchor point on within the category rectangle mentioned previously. This is specified using the `TextBlockAnchor` class. Try running the `DrawStringDemo` class in the JCommon distribution to see how the anchor is used to align text to a point on the screen.
- two additional attributes define a rotation anchor point and a rotation angle. These are applied once the label text has been positioned using the previous two attributes;
- a width ratio and width ratio type control the maximum width of the category labels.

#### 24.8.7 Per Category Tick Label Settings

The category label font and paint settings are inherited from the `Axis` class. However, the `CategoryAxis` class also provides the ability to override these settings on a per-category basis:

```
➔ public Font getTickLabelFont(Comparable category);
Returns the override font for the specified category, or null if there is no setting for that category.

➔ public void setTickLabelFont(Comparable category, Font font);
Sets the override font for the specified category, and sends an AxisChangeEvent to all registered listeners.

➔ public Paint getTickLabelPaint(Comparable category);
Returns the override paint for the specified category, or null if there is no setting for that category.

➔ public void setTickLabelPaint(Comparable category, Paint paint);
Sets the override paint for the specified category, and sends an AxisChangeEvent to all registered listeners.
```

#### 24.8.8 Category Label Tool Tips

It is possible to specify tooltips for the labels along the category axis. This can be useful if you want to use short category names, but have the opportunity to display a longer description. To add a tool tip:

```
➔ public void addCategoryLabelToolTip(Comparable category, String tooltip);
Adds a tooltip for the specified category.
```

To remove a tool tip:

```
→ public void removeCategoryLabelToolTip(Comparable category);
    Removes the tooltip for the specified category.
```

To remove all tool tips:

```
→ public void clearCategoryLabelToolTips();
    Removes all category label tool tips.
```

This feature is not supported by other axis types yet.

#### 24.8.9 Other Methods

To control whether or not a line is drawn for the axis:

```
→ public void setAxisLineVisible(boolean visible);
    Sets the flag that controls whether or not a line is drawn for the axis. Often, this isn't required because the CategoryPlot draws an outline around itself by default. However, sometimes the plot will have no outline OR the axis may be offset from the plot.
```

The Java2D coordinates for the start, middle and end of the category along the axis are given by:

```
→ public double getCategoryStart(int category, int categoryCount, Rectangle2D area,
    RectangleEdge edge);
    Returns the start of the specified category (in Java2D units), assuming the axis lies along the specified edge of the given area.

→ public double getCategoryMiddle(int category, int categoryCount, Rectangle2D area,
    RectangleEdge edge);
    Returns the middle of the specified category (in Java2D units), assuming the axis lies along the specified edge of the given area.

→ public double getCategoryEnd(int category, int categoryCount, Rectangle2D area,
    RectangleEdge edge);
    Returns the end of the specified category (in Java2D units), assuming the axis lies along the specified edge of the given area.
```

#### 24.8.10 Internals

In JFreeChart, axes are owned/managed by a plot. The plot is responsible for assigning drawing space to all of the axes in a plot, which it does by first asking the axes to estimate the space they require (primarily for the axis labels). The following method is used:

```
→ public AxisSpace reserveSpace(Graphics2D g2, Plot plot,
    Rectangle2D plotArea, RectangleEdge edge, AxisSpace space);
    Updates the axis space to allow room for this axis to be drawn.
```

When reserving space, the axis needs to determine the tick marks along the axis, which it does via the following method:

```
→ public List refreshTicks(Graphics2D g2, AxisState state,
    Rectangle2D plotArea, Rectangle2D dataArea, RectangleEdge edge);
    Returns a list of the ticks along the axis.
```

After the plot has estimated the space required for each axis, it then computes the “data area” and draws all the axes around the edges of this area:

```
→ public AxisState draw(Graphics2D g2, double cursor,
    Rectangle2D plotArea, Rectangle2D dataArea, RectangleEdge edge);
    Draws the axis along a specific edge of the data area. The cursor is a measure of how far from the edge of the data area the axis should be drawn (another axis may have been drawn along the same edge already, for example) and the plot area is the region inside which all the axes should fit (it contains the data area).
```

For a given rectangular region in Java2D space, the axis can be used to calculate an x-coordinate or a y-coordinate (depending on which edge of the rectangle the axis is aligned) for the start, middle or end of a particular category on the axis:

```

→ public double getCategoryJava2DCoordinate(CategoryAnchor anchor,
int category, int categoryCount, Rectangle2D area, RectangleEdge edge);
Returns the x- or y-coordinate (in Java2D space) of the specified category.

→ protected double calculateCategorySize(int categoryCount, Rectangle2D area,
RectangleEdge edge);
Returns the width (in Java2D units) of one category assuming the axis lies along the specified
edge of the given area. The size is a function of the length of the edge along which the axis lies,
the number of categories, and the upper, lower and category margins specified for the axis.

→ protected double calculateCategoryGapSize(int categoryCount, Rectangle2D area,
RectangleEdge edge);
Returns the width (in Java2D units) of the gap between categories, assuming the axis lies
along the specified edge of the given area. The gap size is a function of the length of the edge
along which the axis lies, the number of categories, and the upper, lower and category margins
specified for the axis.

```

To draw the category labels, JFreeChart calls the following method:

```

→ protected AxisState drawCategoryLabels(Graphics2D g2, Rectangle2D plotArea,
Rectangle2D dataArea, RectangleEdge edge, AxisState state,
PlotRenderingInfo plotState); [1.0.2]
Draws the category labels.1

```

### 24.8.11 Cloning and Serialization

This class is `Cloneable` and `Serializable`.

### 24.8.12 Notes

Some points to note:

- tick marks are not supported by this axis (yet);
- the foreground paint can be set for tick labels, but not the background paint.

#### See Also

[CategoryPlot](#), [CategoryAxis3D](#).

## 24.9 CategoryAxis3D

### 24.9.1 Overview

An extension of the `CategoryAxis` class that adds a 3D effect. If you use a `CategoryItemRenderer` that draws items with a 3D effect, then you need to ensure that you are using this class rather than a regular `CategoryAxis`. Eventually, the aim is to combine this class into the `CategoryAxis` class.

### 24.9.2 Constructors

There are two constructors:

```

→ public CategoryAxis3D();
Creates a new axis with no label.

→ public CategoryAxis3D(String label);
Creates a new axis with the specified label (null is permitted).

```

---

<sup>1</sup>Prior to 1.0.2, a `drawCategoryLabels()` method without the `dataArea` argument was used.

### 24.9.3 Methods

The 3D effect is implemented simply by overriding two key methods:

```
→ public AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea,
    Rectangle2D dataArea, RectangleEdge edge, PlotRenderingInfo plotState);
    Draws the axis with a 3D effect. The offsets for the 3D effect are obtained from the plot's main
    renderer.

→ public double getCategoryJava2DCoordinate(CategoryAnchor anchor, int category,
    int categoryCount, Rectangle2D area, RectangleEdge edge);
    Returns the Java2D coordinate for the specified category, taking into account the 3D effect.
```

#### See Also

[NumberAxis3D](#).

## 24.10 CategoryLabelPosition

### 24.10.1 Overview

This class records the attributes that control the positioning (including alignment and rotation) of category labels along a [CategoryAxis](#):

- the *category anchor* - a [RectangleAnchor](#) that is used to determine the point on the axis against which the category label is aligned. This is specified relative to a rectangular area that the [CategoryAxis](#) allocates for the category (see figure 24.4);
- the *label anchor* - a [TextBlockAnchor](#) that determines the point on the category label (a [TextBlock](#)) that is aligned with the category anchor;
- the *rotation anchor* - the point on the category label about which the label is rotated (note that there may be no rotation);
- the *rotation angle* - the angle of the rotation, specified in radians;
- the *category label width type* - controls whether the maximum width for the labels is relative to the width of the category label rectangle (the default) or the length of the range axis (useful when labels are rotated so that they are perpendicular to the category axis);
- the maximum category label width ratio, measured as a percentage of either the category label rectangle or the length of the range axis (see the previous setting).

### 24.10.2 Usage

To customise the label positioning, alignment and rotation, you would typically create four instances of this class (one for each of the possible axis locations) and use these to create a [CategoryLabelPositions](#) object.

### 24.10.3 Notes

The following points should be noted:

- instances of this class are immutable, a fact that is relied upon by code elsewhere in the JFreeChart library.

## 24.11 CategoryLabelPositions

### 24.11.1 Overview

This class is used to specify the positions of category labels on a `CategoryAxis`. To account for the fact that an axis can appear in one of four different locations (the top, bottom, left or right of the plot) this class contains four instances of the `CategoryLabelPosition` class—the axis will choose the appropriate one when the labels are being drawn.

Several static instances of this class have been predefined in order to simplify general usage of the `CategoryAxis` class:

Value:	Description:
STANDARD	The default label positions.
UP_90	The labels are rotated 90 degrees, with the text running from the bottom to the top of the chart.
DOWN_90	The labels are rotated 90 degrees, with the text running from the top to the bottom of the chart.
UP_45	The labels are rotated 45 degrees, with the text running towards the top of the chart.
DOWN_45	The labels are rotated 45 degrees, with the text running towards the bottom of the chart.

Table 24.4: Static instances of the `CategoryLabelPositions` class

### 24.11.2 Usage

For example, to change the category axis labels to a 45 degree angle:

```
CategoryAxis domainAxis = plot.getDomainAxis();
domainAxis.setCategoryLabelPositions(CategoryLabelPositions.UP_45);
```

The above example uses one of the predefined instances of this class. However, you can also experiment with creating your own instance, to fully customise the category label positions.

## 24.12 CategoryLabelWidthType

### 24.12.1 Overview

This class defines tokens that are used to specify how the maximum category label width ratio—a setting that limits the width of category labels relative to the size of the plot—is applied. See table 24.5 for the tokens that are defined.

### 24.12.2 Usage

This class is used for the creation of `CategoryLabelPosition` instances.

ID:	Description:
<code>CategoryLabelWidthType.CATEGORY</code>	The maximum width is a percentage of the category width (for example, 0.90 for 90 percent).
<code>CategoryLabelWidthType.RANGE</code>	The maximum width is a percentage of the length of the range axis (typically used when the labels are displayed perpendicular to the category axis).

Table 24.5: Tokens defined by `CategoryLabelWidthType`

### 24.12.3 Notes

Some points to note:

- the maximum category label width ratio is set using the `setMaximumCategoryLabelWidthRatio()` method in the `CategoryPlot` class (or, if this is 0.0, the ratio is taken from the `CategoryLabelPosition` instance);
- when a category label reaches its maximum width, it will wrap to another line (up to the maximum number of lines allowed). If the full label cannot be displayed within the maximum number of lines allowed, the label is truncated.

## 24.13 CategoryTick

### 24.13.1 Overview

A class used to represent a single tick on a `CategoryAxis`. This class is used internally and it is unlikely that you should ever need to use it directly.

## 24.14 ColorBar

### 24.14.1 Overview

A *color bar* is used with a `ContourPlot`. *This class is deprecated as of version 1.0.4.*

## 24.15 CompassFormat

### 24.15.1 Overview

A custom `NumberFormat` class that can be used to display numerical values as compass directions—see figure 24.5 for an example. In the example, the range axis on the left side of the chart displays

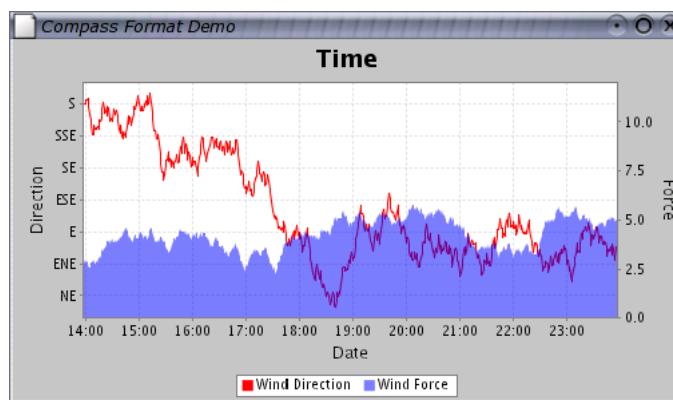


Figure 24.5: A chart that uses the `CompassFormat` class

compass directions in place of numerical values.

### 24.15.2 Usage

There is a demo (`CompassFormatDemo1.java`) included in the JFreeChart demo collection.

### 24.15.3 Methods

To convert an angle (in degrees) to a compass direction (for example, “NE”):

► `public String getDirectionCode(double direction);`

Returns the compass direction (as a `String`) that corresponds to the given `direction` (which is expressed in degrees). The return value is one of: N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW.

The following methods perform the required formatting, but are usually not called directly (see Java’s `NumberFormat` class for more details):

► `public StringBuffer format(double number, StringBuffer toAppendTo, FieldPosition pos);`

Converts the given number to a string containing the corresponding direction.

► `public StringBuffer format(long number, StringBuffer toAppendTo, FieldPosition pos);`

Converts the given number to a string containing the corresponding direction.

Parsing is not supported:

► `public Number parse(String source, ParsePosition parsePosition);`

This method always returns `null`, which means this formatter cannot be used for parsing.

### 24.15.4 Notes

Some points to note:

- this class cannot be used for parsing numbers;
- a demo application (`CompassFormatDemo1.java`) is included in the JFreeChart demo collection.

## 24.16 CyclicNumberAxis

### 24.16.1 Overview

An extension of the `NumberAxis` class that is used to generate cyclic plots.

### 24.16.2 Constructors

To create a new axis:

► `public CyclicNumberAxis(double period);`

Creates a new axis with the specified `period` and a zero offset. No label is set for the axis.

► `public CyclicNumberAxis(double period, double offset);`

Creates a new axis with the specified `period` and `offset`. No label is set for the axis.

► `public CyclicNumberAxis(double period, String label);`

Creates a new axis with the specified `period` and axis `label`. The offset is zero.

► `public CyclicNumberAxis(double period, double offset, String label);`

Creates a new axis with the specified `period`, `offset` and `label`.

### 24.16.3 Methods

To control the visibility of the “advance line”:

► `public boolean isAdvanceLineVisible();`

Returns the flag that controls whether or not the advance line is displayed.

► `public void setAdvanceLineVisible(boolean visible);`

Sets the flag that controls whether or not the advance line is displayed.

► `public Paint getAdvanceLinePaint();`

Returns the paint used to draw the advance line (never `null`).

```

→ public void setAdvanceLinePaint(Paint paint);
Sets the paint used to draw the advance line (null not permitted).

→ public Stroke getAdvanceLineStroke();
Returns the stroke used to draw the advance line (never null).

→ public void setAdvanceLineStroke(Stroke stroke);
Sets the stroke used to draw the advance line (null not permitted).

→ public boolean isBoundMappedToLastCycle();
To be documented.

→ public void setBoundMappedToLastCycle(boolean boundMappedToLastCycle);
To be documented.

```

## 24.17 DateAxis

### 24.17.1 Overview

An axis that displays date/time values—extends `ValueAxis`. This class is designed to be flexible about the range of dates/times that it can display—anything from a few milliseconds to several centuries can be handled.

A date axis can be used for the domain and/or range axis in an `XYPlot`. In a `CategoryPlot`, a date axis can only be used for the range axis.

### 24.17.2 Usage

To change the attributes of the axis, you need to obtain a `DateAxis` reference—because of the way JFreeChart is designed, this usually involves a “cast”:

```

XYPlot plot = (XYPlot) chart.getPlot();
ValueAxis domainAxis = plot.getDomainAxis();
if (domainAxis instanceof DateAxis) {
    DateAxis axis = (DateAxis) domainAxis;
    // customise axis here...
}

```

Given a `DateAxis` reference, you can change:

- the axis range, see section [24.17.5](#);
- the size and formatting of the tick labels, see section [24.17.7](#);
- other inherited attributes, see section [24.44.4](#).

### 24.17.3 Constructors

The default constructor creates a new axis with no label:

```

→ public DateAxis();
Creates a new date axis with no label.

```

You can specify the label using:

```

→ public DateAxis(String label);
Creates a new axis with the specified label (null permitted, in which case no label is displayed
for the axis).

```

Sometimes it is useful to be able to specify the time zone used for the tick marks and labels on the axis:

```

→ public DateAxis(String label, TimeZone zone);
Creates a new date axis where the tick marks and labels are calculated for the specified time
zone.

```

#### 24.17.4 Attributes

The following attributes are defined, in addition to those inherited from the `ValueAxis` class:

Attribute:	Description:
<code>dateFormatOverride</code>	A date formatter that, if set, overrides the format of the tick labels displayed on the axis.
<code>tickUnit</code>	Controls the size and formatting of the tick labels on the axis (an instance of <code>DateTickUnit</code> ).
<code>minimumDate</code>	The minimum date/time visible on the axis.
<code>maximumDate</code>	The maximum date/time visible on the axis.
<code>verticalTickLabels</code>	A flag that controls whether or not the tick labels on the axis are displayed “vertically” (that is, rotated 90 degrees from horizontal).

Refer to section [24.44.3](#) for information about the attributes inherited by this class.

#### 24.17.5 The Axis Range

The range of dates displayed by the axis is controlled with the following methods:

- `public Date getMinimumDate();`  
Returns the earliest date along the axis range.
- `public void setMinimumDate(Date date);`  
Sets the earliest date for the axis.
- `public Date getMaximumDate();`  
Returns the latest date along the axis range.
- `public void setMaximumDate(Date maximumDate);`  
Sets the latest date for the axis.

To set the axis range:<sup>2</sup>

- `public void setRange(Range range);`  
Sets the range of values to be displayed by the axis and sends an `AxisChangeEvent` to all registered listeners.
- `public void setRange(Range range, boolean turnOffAutoRange, boolean notify);`  
Sets the range of values to be displayed by the axis. The `turnOffAutoRange` flag controls whether the auto range calculation is disabled or not (usually you want to disable it) and the `notify` flag controls whether or not an `AxisChangeEvent` is sent to all registered listeners.
- `public void setRange(Date lower, Date upper);`  
Sets the range of values to be displayed by the axis.
- `public void setRange(double lower, double upper);`  
Sets the range of values to be displayed by the axis and sends an `AxisChangeEvent` to all registered listeners.

For example:

```
// start and end are instances of java.util.Date
axis.setRange(start, end);
```

#### 24.17.6 Time Zone

To control the time zone for the axis (which affects the conversion of date values to string labels):

- `public TimeZone getTimeZone(); [1.0.4]`  
Returns the time zone for the axis (normally specified in the constructor).
- `public void setTimeZone(TimeZone zone); [1.0.4]`  
Sets the time zone for the axis and sends an `AxisChangeEvent` to all registered listeners.

<sup>2</sup>Note that when you set the axis range in this way, the `auto-range` attribute is set to `false`. It is assumed that by setting a range manually, you do not want that subsequently overridden by the auto-range calculation.

### 24.17.7 Tick Units

The tick units on the date axis are controlled by a similar “auto tick unit selection” mechanism to that used in the `NumberAxis` class. This mechanism relies on a collection of “standard” tick units (stored in an instance of `TickUnits`). The axis will try to select the smallest tick unit that doesn’t cause the tick labels to overlap.

If you want to specify a fixed tick size and format, you can use code similar to this:

```
// set the tick size to one week, with formatting...
DateFormat formatter = new SimpleDateFormat("d-MMM-yyyy");
DateTickUnit unit = new DateTickUnit(DateTickUnit.DAY, 7, formatter);
axis.setTickUnit(unit);
```

Note that setting a tick unit manually in this way disables the “auto” tick unit selection mechanism. You may find that the tick size you have requested results in overlapping labels.

If you just want to control the tick label format, one option is to specify an *override format*:

```
// specify an override format...
DateFormat formatter = new SimpleDateFormat("d-MMM");
axis.setDateFormatOverride(formatter);
```

This is a simple and effective approach in some situations, but has the limitation that the same format is applied to all tick sizes.

A final approach to controlling the formatting of tick labels is to create your own `TickUnits` collection. The collection can contain any number of `DateTickUnit` objects, and should be registered with the axis as follows:

```
// supply a new tick unit collection...
axis.setStandardTickUnits(myCollection);
```

### 24.17.8 Tick Label Orientation

To control the orientation of the tick labels on the axis:

```
axis.setVerticalTickLabels(true);
```

*This code survives from the early days of JFreeChart development when there were separate classes `HorizontalDateAxis` and `VerticalDateAxis`...it needs to be changed to be more generic for axes that could have either a horizontal or vertical orientation.*

### 24.17.9 Timelines

This class uses a `Timeline` to provide an opportunity for the axis to map from Java time (measured in milliseconds since 1 January 1970, 00:00:00 GMT), to some other time scale. The default time line performs an “identity” mapping—that is, the millisecond values are not changed.

Use the following methods to change the time line:

► public `Timeline` `getTimeline()`  
Returns the current time line.

► public void `setTimeline(Timeline timeline)`  
Sets the time line and sends an `AxisChangeEvent` to all registered listeners.

#### 24.17.10 Other Methods

You can specify a fixed tick unit for the axis:

```
→ public DateTickUnit getTickUnit();
Returns the tick unit (possibly null, in which case a tick unit will be selected automatically.)
```

```
→ public void setTickUnit(DateTickUnit unit);
Sets the fixed tick unit for the axis and sends an AxisChangeEvent to all registered listeners.
```

```
→ public void setTickUnit(DateTickUnit unit, boolean notify,
boolean turnOffAutoSelection);
Sets the fixed tick unit for the axis.
```

You can specify an override formatter for the tick labels:

```
→ public DateFormat getDateFormatOverride();
Returns the formatter for the tick labels. If this is non-null, it is used to override any other
formatter.
```

```
→ public void setDateFormatOverride(DateFormat formatter)
Sets the formatter and sends an AxisChangeEvent to all registered listeners. You should be
careful using this method, it overrides the date formatting without consideration for the size
of the tick units. If you choose an inappropriate date format you will get bad axis labelling.
```

Tick marks and labels are displayed at regular intervals along the axis. You can control whether the marks are positioned at the start, middle or end of the interval:

```
→ public DateTickMarkPosition getTickMarkPosition();
Returns the position for the tick marks within each interval along the axis.
```

```
→ public void setTickMarkPosition(DateTickMarkPosition position);
Sets the position for the tick marks within each interval along the axis and sends an AxisChangeEvent
to all registered listeners.
```

```
→ public void configure();
Configures the axis which involves recalculating the axis range (if the autoRange flag is switched
on).
```

```
→ public boolean isHiddenValue(long millis);
Returns true if the specified millisecond is hidden by the Timeline, and false otherwise.
```

```
→ public double valueToJava2D(double value, Rectangle2D area, RectangleEdge edge);
Converts a data value to Java2D coordinates, assuming that the axis lies along one edge of the
specified area.
```

```
→ public double dateToJava2D(Date date, Rectangle2D area, RectangleEdge edge);
Converts a date to Java2D coordinates, assuming that the axis lies along one edge of the
specified area.
```

```
→ public double java2DToValue(double java2DValue, Rectangle2D area, RectangleEdge edge);
Translates a Java2D coordinate into a data value.
```

```
→ public Date calculateLowestVisibleTickValue(DateTickUnit unit);
Calculates the value of the first tick mark on the axis.
```

```
→ public Date calculateHighestVisibleTickValue(DateTickUnit unit);
Calculates the value of the last tick mark on the axis.
```

```
→ public static TickUnitSource createStandardDateTickUnits();
Creates a set of standard tick units for a date axis.
```

```
→ public static TickUnitSource createStandardDateTickUnits(TimeZone zone);
Creates a set of standard tick units for a date axis.
```

```
→ public List refreshTicks(Graphics2D g2, AxisState state, Rectangle2D plotArea, Rectangle2D
dataArea, RectangleEdge edge);
Returns a list of ticks for the axis. You can override this method to customise the list of
ticks displayed on the axis—see YieldCurveDemo.java in the JFreeChart demo collection for an
example.
```

```

→ public AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea, Rectangle2D dataArea,
    RectangleEdge edge, PlotRenderingInfo plotState);
    Draws the axis. Normally, this method is called by the plot that owns the axis—you shouldn't
    need to call this method yourself.

→ public void zoomRange(double lowerPercent, double upperPercent);
    Changes the axis range to simulate a “zoom” function.

→ public boolean equals(Object obj);
    Tests for equality with an arbitrary object.

```

### 24.17.11 Notes

Some points to note:

- although the axis displays dates for tick labels, at the lowest level it is still working with `double` primitives obtained from the `Number` objects supplied by the plot's dataset. The values are interpreted as *the number of milliseconds since 1 January 1970* (that is, the same encoding used by `java.util.Date`).
- a `DateAxis` is typically used as the domain axis (or x-axis) in a chart, but it can also be used as the range axis (or y-axis)—for example, see the `EventFrequencyDemo1.java` application included in the JFreeChart demo collection.

## 24.18 DateTickMarkPosition

### 24.18.1 Overview

A simple enumeration of the possible tick mark positions for a `DateAxis`. The positions are:

- `DateTickMarkPosition.START`;
- `DateTickMarkPosition.MIDDLE`;
- `DateTickMarkPosition.END`.

Use the `setTickMarkPosition()` method in the `DateAxis` class to change this setting.

## 24.19 DateTick

### 24.19.1 Overview

A class used to represent a single tick on a `DateAxis`.

### 24.19.2 Usage

This class is used internally and it is unlikely that you should ever need to use it directly.

### 24.19.3 Constructor

To create a new instance:

```

→ public DateTick(Date date, String label, TextAnchor textAnchor,
    TextAnchor rotationAnchor, double angle);
    Creates a new tick representing the specified date.

```

#### 24.19.4 General Methods

To get the date for this tick:

```
➔ public Date getDate();
    Returns the date for this tick.
```

#### 24.19.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
    Tests this tick for equality with an arbitrary object.
```

### 24.20 DateTickUnit

#### 24.20.1 Overview

A date tick unit for use by subclasses of `DateAxis` (extends the `TickUnit` class).

The unit size can be specified as a multiple of one of the following time units:

Time Unit:	Constant:
Year	<code>DateTickUnit.YEAR</code>
Month	<code>DateTickUnit.MONTH</code>
Day	<code>DateTickUnit.DAY</code>
Hour	<code>DateTickUnit.HOUR</code>
Minute	<code>DateTickUnit.MINUTE</code>
Second	<code>DateTickUnit.SECOND</code>
Millisecond	<code>DateTickUnit.MILLISECOND</code>

Note that these constants are not the same as those defined by Java's `Calendar` class.

#### 24.20.2 Usage

There are two ways to make use of this class. The first is where you know the exact tick size that you want for your axis. In this case, you create a new date tick unit then call the `setTickUnit()` method in the `DateAxis` class. For example, to set the tick unit size on the axis to one week:

```
XYPlot plot = myChart.getXYPlot();
ValueAxis axis = plot.getDomainAxis();
axis.setTickUnit(new DateTickUnit(DateTickUnit.DAY, 7));
```

The second usage is to create a collection of tick units using the `TickUnits` class, and then allow the `DateAxis` to automatically select an appropriate unit. See the `setStandardTickUnits()` method for more details.

#### 24.20.3 Constructors

To create a new date tick unit:

```
➔ public DateTickUnit(int unit, int count);
    Creates a new tick unit with a default date formatter for the current locale.
```

Alternatively, you can supply your own date formatter:

```
➔ public DateTickUnit(int unit, int count, DateFormat formatter);
    Creates a new date tick unit with the specified date formatter.
```

For both constructors, the `unit` argument should be defined using one of the constants listed in section 24.20.1. The `count` argument specifies the multiplier (often just 1).

#### 24.20.4 Methods

To get the units used to specify the tick size:

► `public int getUnit();`

Returns a constant representing the units used to specify the tick size. The constants are listed in section [24.20.1](#).

To get the number of units:

► `public int getCount();`

Returns the number of units.

To format a date using the tick unit's internal formatter:

► `public String dateToString(Date date);`

Formats the date as a `String`.

The following method is used for simple date addition:

► `public Date addToDate(Date base);`

Creates a new `Date` that is calculated by adding this `DateTickUnit` to the `base` date.

#### 24.20.5 Notes

This class is immutable, a requirement for all subclasses of `TickUnit`.

#### See Also

[NumberTickUnit](#).

### 24.21 ExtendedCategoryAxis

#### 24.21.1 Overview

An extension of the `CategoryAxis` class that allows sublabels to be displayed with the categories.

#### 24.21.2 Notes

Some points to note:

- a couple of demos (`SurveyResultsDemo2.java` and `SurveyResultsDemo3.java`) are included in the JFreeChart demo collection.

### 24.22 LogAxis

#### 24.22.1 Overview

An axis that displays a logarithmic scale. This class was first introduced in JFreeChart version 1.0.7.

#### 24.22.2 Constructors

To create a new axis instance:

► `public LogAxis(); [1.0.7]`

Equivalent to `LogAxis(null)`—see the next constructor.

► `public LogAxis(String label); [1.0.7]`

Creates a new axis with the specified label. If `label` is `null`, the axis will be drawn without a label.

### 24.22.3 General Attributes

To control the base of the logarithm calculation used by the axis:

► public double getBase(); [1.0.7]

Returns the base for the logarithm calculation. The default value is 10.0.

► public void setBase(double base); [1.0.7]

Sets the base for the logarithm calculation, and sends an `AxisChangeEvent` to all registered listeners. If `base` is less than or equal to 1.0, this method throws an `IllegalArgumentException`.

This axis can only display positive values. The smallest positive value that will be displayed on the axis is controlled by:

► public double getSmallestValue(); [1.0.7]

Returns the smallest (positive) value that will be displayed on the axis. The default value is 1E-100.

► public void setSmallestValue(double value); [1.0.7]

Sets the smallest value that will be displayed on the axis, and sends an `AxisChangeEvent` to all registered listeners. If `value` is less than or equal to 0.0, this method throws an `IllegalArgumentException`.

To control the tick unit for the axis:

► public `NumberTickUnit` getTickUnit(); [1.0.7]

Returns the current tick unit for the axis.

► public void setTickUnit(`NumberTickUnit` unit); [1.0.7]

Equivalent to `setTickUnit(unit, true, true)`—see the next method.

► public void setTickUnit(`NumberTickUnit` unit, boolean notify, boolean turnOffAutoSelect); [1.0.7]

Sets the current tick unit and, if requested, sends an `AxisChangeEvent` to all registered listeners.

► public `NumberFormat` getNumberFormatOverride(); [1.0.7]

Returns the override formatter for the tick labels on the axis. The default value is `null`.

► public void setNumberFormatOverride(`NumberFormat` formatter); [1.0.7]

Sets the override formatter (`null` is permitted) for the tick labels on the axis, and sends an `AxisChangeEvent` to all registered listeners.

To control the number of minor tick marks shown between each major tick mark:

► public int getMinorTickCount(); [1.0.7]

Returns the minor tick count. The default value is 10.

► public void setMinorTickCount(int count); [1.0.7]

Sets the minor tick count and sends an `AxisChangeEvent` to all registered listeners.

### 24.22.4 Other Methods

This class provides a number of methods that are typically called by JFreeChart rather than by user code.

► public double calculateLog(double value); [1.0.7]

Returns  $x$ , where  $value = \text{Math.pow}(\text{base}, x)$ .

► public double calculateValue(double log); [1.0.7]

Returns  $y$ , where  $y = \text{Math.pow}(\text{base}, \log)$ .

To convert from data-space to Java2D-space and back again:

► public double java2DToValue(double java2DValue, `Rectangle2D` area, `RectangleEdge` edge); [1.0.7]

Converts a Java2D coordinate into data-space, assuming the axis lies along the specified `edge` of the given `area`.

```
→ public double valueToJava2D(double value, Rectangle2D area, RectangleEdge edge); [1.0.7]
Converts a data value into Java2D space, assuming the axis lies along the specified edge of the given area.
```

There are a couple of methods related to the auto-range calculation:

```
→ public void configure(); [1.0.7]
```

Updates the axis bounds if the `autoRange` flag is set. This method is usually called when an axis is first assigned to a plot.

```
→ protected void autoAdjustRange(); [1.0.7]
```

Updates the axis bounds to match the available data.

To draw the axis:

```
→ public AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea,
Rectangle2D dataArea, RectangleEdge edge, PlotRenderingInfo plotState); [1.0.7]
Draws the axis along one edge of the specified dataArea.
```

Some methods relating to the tick units on the axis:

```
→ public List refreshTicks(Graphics2D g2, AxisState state, Rectangle2D dataArea,
RectangleEdge edge); [1.0.7]
```

Returns a list of tick marks and labels for the axis.

```
→ protected void selectAutoTickUnit(Graphics2D g2, Rectangle2D dataArea, RectangleEdge edge); [1.0.7]
```

Selects a tick unit from the current `TickCountSource` in such a way that the tick labels will not overlap.

```
→ public double exponentLengthToJava2D(double length, Rectangle2D area, RectangleEdge edge); [1.0.7]
```

Returns the length, in Java2D units, of a value given in the linear scale for the axis.

```
→ public static TickUnitSource createLogTickUnits(Locale locale); [1.0.7]
```

Creates a default set of tick units for the axis.

The following method is overridden to support zooming:

```
→ public void zoomRange(double lowerPercent, double upperPercent); [1.0.7]
```

Adjusts the axis range to show the interval specified.

#### 24.22.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj); [1.0.7]
```

Tests this axis for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### 24.22.6 Notes

A demo (`LogAxisDemo1.java`) is included in the JFreeChart demo collection.

#### See Also:

[LogarithmicAxis](#).

## 24.23 LogarithmicAxis

### 24.23.1 Overview

A numerical axis that displays values using a logarithmic scale (with base 10). This class extends `NumberAxis` and can be used anywhere that a `NumberAxis` can be used, including:

- as the range axis on a `CategoryPlot`;
- as the domain and/or range axis on an `XYPlot`;
- as the range axis on a `ThermometerPlot`.

*Note: This class has some quirks and isn't quite as flexible as it could be. There's now an alternative axis that you might want to try—`LogAxis`.*

### 24.23.2 Constructors

This class has a single constructor:

```
➔ public LogarithmicAxis(String label);
```

Creates a new axis with the specified label. If the label is `null`, the axis is displayed without a label. The default tick label format for the axis will be regular numeric labels, rather than the scientific or power notations.

### 24.23.3 The Axis Range

By default, the axis range is automatically calculated to match the range of values plotted against the axis. If you prefer to set the axis range manually, the following method will do this:

```
➔ public void setRange(Range range);
```

Sets the bounds of the axis to the given `range`.

Once a range has been manually set, changes to the dataset do not alter the axis range. You can restore the automatic range calculation using:

```
axis.setAutoRange(true);
```

A flag is used to control whether the automatically calculated range is expanded to include the next “power of ten” value:

```
➔ public boolean getAutoRangeNextLogFlag();
```

Returns `true` if the next power of ten is included in the automatic range, and `false` otherwise.

```
➔ public void setAutoRangeNextLogFlag(boolean flag);
```

Sets the flag that controls whether the next “power of ten” is included in the automatic range calculation. The default value is `false`.

The following method updates the axis bounds according to the values in the dataset (it is usually called by JFreeChart, you shouldn't need to call this method directly):

```
➔ public void autoAdjustRange();
```

Updates the axis bounds to reflect the range of values in the dataset.

### 24.23.4 Negative Values

A logarithmic axis can only display positive values. However, the JFreeChart implementation provides an option to allow negative values to be plotted on a logarithmic scale.

```
➔ public boolean getAllowNegativesFlag();
```

Returns `true` if the axis is configured to display negative values, and `false` otherwise.

```
➔ public void setAllowNegativesFlag(boolean flgVal);
```

Sets the flag that controls whether or not the axis will display negative values.

The `strictValues` flag controls whether or not a `RuntimeException` is thrown when a negative value is encountered and the `allowNegativeValues` flag is `false`. Note: setting this flag to `false` appears to be equivalent to setting the `allowNegativesFlag` to `true`.

```
► public boolean getStrictValuesFlag();
  Returns the value of the strictValuesFlag.

► public void setStrictValuesFlag(boolean flgVal);
  Sets the flag that controls whether or not a RuntimeException is thrown when the allowNegativesFlag is false and a negative value is encountered.
```

### 24.23.5 Tick Label Formatting

The axis can display tick labels in several formats:

- as a regular number (the default);
- in the form  $10^x$ ;
- using scientific notation (for example, `1E8` which is  $1 \times 10^8$ ).

The `log10TickLabelsFlag` flag controls the selection of the “ $10^x$ ” format:

```
► public boolean getLog10TickLabelsFlag();
  Returns true if the tick labels are displayed in the form “ $10^x$ ”, and false otherwise.

► public void setLog10TickLabelsFlag(boolean flag);
  Sets the flag that controls whether the tick labels are displayed in the form “ $10^x$ ”. This flag takes precedence over the expTickLabelsFlag flag.
```

The `expTickLabelsFlag` flag controls the selection of the scientific format:

```
► public boolean getExpTickLabelsFlag();
  Returns true if the tick labels are displayed using scientific notation (for example, 1E2, which is equivalent to  $1 \times 10^2$ ), and false otherwise.

► public void setExpTickLabelsFlag(boolean flgVal);
  Sets a flag that controls whether the tick labels are displayed in scientific notation. This flag is ignored if the log10TickLabelsFlag is true.
```

### 24.23.6 Other Methods

The remaining methods in this class are used to convert data values to Java2D coordinates and vice-versa:

```
► public double valueToJava2D(double value, Rectangle2D plotArea, RectangleEdge edge);
  Converts value to a Java2D coordinate along the edge of the given PlotArea.

► public double java2DToValue(double java2DValue, Rectangle2D plotArea, RectangleEdge edge);
  Converts java2DValue to an axis value.

► public double adjustedLog10(double val);
  To be documented.
```

### 24.23.7 Notes

An alternative to this class is the [LogAxis](#) class.

## 24.24 MarkerAxisBand

### 24.24.1 Overview

A band that can be added to a [NumberAxis](#) to highlight certain value ranges. NOTE: this facility is broken at present, so this class should not be used.

## 24.25 ModuloAxis

### 24.25.1 Overview

This axis is a special extension of [NumberAxis](#) that presents a fixed range of values in a “circular” or “cyclic” fashion. It was originally developed to display directional measurements (that is, values in the range 0 to 360 degrees), but should be general enough to be applied for other uses. The [CompassFormatDemo2](#) application (included in the JFreeChart demo collection) provides one example of this axis in use—see figure 24.6.

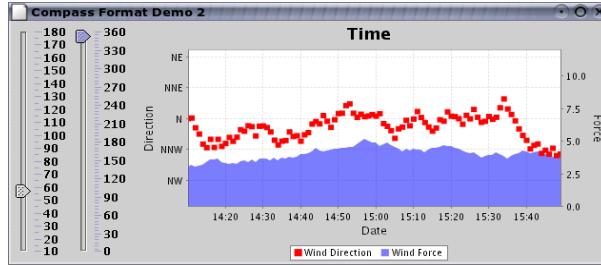


Figure 24.6: A chart that uses a *ModuloAxis*

### 24.25.2 Constructor

There is a single constructor:

```
→ public ModuloAxis(String label, Range fixedRange);
```

Creates a new axis with the specified *label* and *fixedRange*.

### 24.25.3 The Display Range

The display range is the subset (of the fixed range) that is currently displayed by the axis. It is defined by a start value and an end value. It is possible for the start value to be greater than the end value—in this case, the displayed range is formed from two parts: (1) the start value to the upper bound of the fixed range, and (2) the lower bound of the fixed range to the end value.

To find the current display range:

```
→ public double getDisplayStart();
```

Returns the start value of the range being displayed by the axis. This value will always fall within the fixed range specified in the constructor.

```
→ public double getDisplayEnd();
```

Returns the end value of the range being displayed by the axis. This value will always fall within the fixed range specified in the constructor.

To set the display range:

```
→ public void setDisplayRange(double start, double end);
```

Sets the display range for the axis. If either *start* or *end* fall outside the fixed range specified in the constructor, they will first be mapped to the fixed range (using a modulo-like calculation). It is possible for *start* to be greater than *end*—in this case, the displayed range is formed from two parts: (1) the start value to the upper bound of the fixed range, and (2) the lower bound of the fixed range to the end value.

#### 24.25.4 Other Methods

Other methods defined for this class are mainly for internal use:

► `public double valueToJava2D(double value, Rectangle2D area, RectangleEdge edge);`

Converts a data value to a Java2D coordinate, assuming that the axis lies along the specified `edge` of the given `area`. This method overrides the method provided by `NumberAxis` to account for the fact that the display range may be in two pieces.

► `public double java2DToValue(double java2DValue, Rectangle2D area, RectangleEdge edge);`

Converts a Java2D coordinate into a data value, assuming that the axis lies along the specified `edge` of the given `area`. This method overrides the method provided by `NumberAxis` to account for the fact that the display range may be in two pieces.

► `public void resizeRange(double percent);`

Resizes the display range, about its central value, by the specified percentage (values less than 1.0 or 100% will shrink the range, while values greater than 1.0 will expand the range).

► `public void resizeRange(double percent, double anchorValue);`

Resizes the display range by the specified percentage about the `anchorValue`. Percentage values less than 1.0 or 100% will shrink the range, while values greater than 1.0 will expand the range).

► `public double lengthToJava2D(double length, Rectangle2D area, RectangleEdge edge);`

Converts a length (specified in data space) into Java2D units. This method overrides the method specified in `NumberAxis` to account for the fact that the displayed range on the axis may be in two pieces.

### 24.26 MonthDateFormat

#### 24.26.1 Overview

A custom date formatter that displays the month and, optionally, the year. This formatter is typically used with the `PeriodAxis` class. An example of the sequence of date strings that can be generated with this formatter is:

```
Jan06 Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Notice how the first month has the year appended to it (this is configurable for every month).

#### 24.26.2 Constructors

There are a range of constructors that allow you to configure the formatter as appropriate:

► `public MonthDateFormat();`

Creates a new formatter for the default time zone. This is equivalent to:

```
new MonthDateFormat(TimeZone.getDefault());
```

► `public MonthDateFormat(TimeZone zone);`

Creates a new formatter for the given time zone. This is equivalent to:

```
new MonthDateFormat(zone, Locale.getDefault(), 1, true, false);
```

This means that months are labelled with a single letter, and a two-digit year indicator is added to January only.

► `public MonthDateFormat(TimeZone zone, int chars);`

Creates a new formatter for the given time zone, with the specified number of characters for each month. This is equivalent to:

```
new MonthDateFormat(zone, Locale.getDefault(), chars, true, false);
```

A two-digit year indicator is added to January only.

► `public MonthDateFormat(Locale locale);`

Creates a new formatter for the given locale. This is equivalent to:

```
new MonthDateFormat(TimeZone.getDefault(), locale, 1, true, false);
```

This means that months are labelled with a single letter, and a two-digit year indicator is added to January only.

► `public MonthDateFormat(Locale locale, int chars);`

Creates a new formatter for the given locale, with the specified number of characters for each month. This is equivalent to:

```
new MonthDateFormat(TimeZone.getDefault(), locale, chars, true, false);
```

A two-digit year indicator is added to January only.

► `public MonthDateFormat(TimeZone zone, Locale locale, int chars, boolean showYearForJan, boolean showYearForDec);`

Creates a new formatter for the given time zone and locale. The `chars` argument specifies the number of characters to display for each month name. The remaining flags control whether or not the year is displayed for the months January and December (the year is NOT displayed for the other months). The year is formatted using `new SimpleDateFormat("yy")`.

The remaining constructor allows every attribute to be customised:

► `public MonthDateFormat(TimeZone zone, Locale locale, int chars, boolean[] showYear, DateFormat yearFormatter);`

Creates a new formatter for the given time zone and locale. The time zone determines which month a given date falls into, while the locale determines the labels used for the months. The `chars` argument specifies the number of characters to display for each month name. The `showYear` array should contain 12 flags that determine whether the year is appended to each month label (sometimes you will want the year appended to every month, sometimes you may just want the first month (January) of each year to have the year displayed). The final argument controls the formatting of the year.

### 24.26.3 Methods

The following methods from `DateFormat` are required to be overridden—you won't normally call these methods directly:

► `public StringBuffer format(Date date, StringBuffer toAppendTo, FieldPosition fieldPosition);`  
Formats the given date.

► `public Date parse(String source, ParsePosition pos);`  
This method returns `null` always, which means this formatter cannot be used to parse text into dates.

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`  
Tests this formatter for equality with an arbitrary object.

## 24.27 NumberAxis

### 24.27.1 Overview

An axis that displays numerical data along a linear scale. This class extends `ValueAxis`. You can create your own subclasses if you have special requirements.

### 24.27.2 Usage

A `NumberAxis` can be used for:

- the range axis in a `CategoryPlot`.
- the domain and/or range axes in an `XYPlot`;

The methods for obtaining a reference to the axis typically return a `ValueAxis`, so you will need to “cast” the reference to a `NumberAxis` before using any of the methods specific to this class. For example:

```
ValueAxis rangeAxis = plot.getRangeAxis();
if (rangeAxis instanceof NumberAxis) {
    NumberAxis axis = (NumberAxis) rangeAxis;
    axis.setAutoRangeIncludesZero(true);
}
```

This casting technique is used often in JFreeChart.<sup>3</sup>

### 24.27.3 Constructors

To create a new axis:

<code>► public NumberAxis();</code> Creates a new axis with no label.	<code>► public NumberAxis(String label);</code> Creates a new axis with the specified label. If <code>label</code> is <code>null</code> , the axis will be displayed without a label.
--	--

### 24.27.4 Attributes

The following table lists the properties maintained by `NumberAxis`, in addition to those inherited from `ValueAxis`.

Attribute:	Description:
<code>rangeType</code>	Defines the permitted range for the axis: <code>RangeType.FULL</code> , <code>RangeType.POSITIVE</code> and <code>RangeType.NEGATIVE</code> .
<code>autoRangeIncludesZero</code>	A flag that indicates whether or not zero is always included when the axis range is determined automatically.
<code>autoRangeStickyZero</code>	A flag that controls the behaviour of the auto-range calculation when zero falls within the lower or upper margin for the axis. If <code>true</code> , the margin will be truncated at zero.
<code>numberFormatOverride</code>	A <code>NumberFormat</code> that, if set, overrides the formatting of the tick labels for the axis.
<code>verticalTickLabels</code>	A flag that indicates whether or not the tick labels are rotated to vertical.
<code>markerBand</code>	An optional band that highlights ranges along the axis (see <code>MarkerAxisBand</code> ).

The following default values are used for attributes wherever necessary:

Name:	Value:
<code>DEFAULT_MINIMUM_AXIS_VALUE</code>	<code>0.0</code>
<code>DEFAULT_MAXIMUM_AXIS_VALUE</code>	<code>1.0</code>
<code>DEFAULT_MINIMUM_AUTO_RANGE</code>	<code>new Double(0.000001);</code>
<code>DEFAULT_TICK_UNIT</code>	<code>new NumberTickUnit(new Double(1.0), new DecimalFormat("0"));</code>

### 24.27.5 The Axis Range

You can control most aspects of the axis range using methods inherited from the `ValueAxis` class—see section 24.44.5 for details. Some additional controls are added by this class.

To restrict the axis to display only positive values, or only negative values, you can set the `rangeType` attribute:

<code>► public RangeType getRangeType();</code> Returns the range type (never <code>null</code> ).
---

---

<sup>3</sup>If you are sure of what you are doing, you can drop the `instanceof` check.

► public void setRangeType(**RangeType** rangeType);  
 Sets the range type and sends an **AxisChangeEvent** to all registered listeners.

If you have set the `autoRange` flag to `true` (so that the axis range automatically adjusts to fit the current data), you may also want to set the `autoRangeIncludesZero` flag to ensure that the axis range always includes zero:

► public boolean getAutoRangeIncludesZero();  
 Returns `true` if the auto-range calculation ensures that zero is included in the range, and `false` otherwise.  
 ► public void setAutoRangeIncludesZero(boolean flag);  
 Sets the `autoRangeIncludesZero` flag and sends an **AxisChangeEvent** to all registered listeners.  
 Note that some renderers (for example, `BarRenderer`) have a flag to control the inclusion of some “base” value in the axis range—since the base value often defaults to zero, you may need to set the flag in the renderer also, to get the required range for the axis.

If the `autoRangeIncludesZero` flag is set to `true`, then you can further control how the axis margin is calculated when zero falls within the axis margin. By setting the `autoRangeStickyZero` flag to `true` you can truncate the margin at zero:

► public boolean getAutoRangeStickyZero();  
 Returns `true` if the axis margin should be truncated at zero if the zero value falls within the margin.  
 ► public void setAutoRangeStickyZero(boolean flag);  
 Sets the flag that controls whether or not the axis margin is truncated if the zero value falls within the margin. An **AxisChangeEvent** is sent to all registered listeners.

#### 24.27.6 Auto Tick Unit Selection

The `NumberAxis` class contains a mechanism for automatically selecting a tick unit from a collection of “standard” tick units. The aim is to display as many ticks as possible, without the tick labels overlapping. The appropriate tick unit will depend on the axis range (which is often a function of the available data) and the amount of space available for displaying the chart.

The `default` standard tick unit collection contains about 50 tick units ranging in size from 0.0000001 to 1,000,000,000. The collection is created and returned by the `createStandardTickUnits()` method.

You can replace the default collection with any other collection of tick units you care to create. One common situation where this is necessary is the case where your data consists of integer values only. In this case, you only want the axis to display integer tick values, but sometimes the axis will show values like 0.00, 2.50, 5.00, 7.50, 10.00, when you might prefer 0, 2, 4, 6, 8, 10. For this situation, a set of standard integer tick units has been created. Use the following code:

```
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
TickUnits units = NumberAxis.createIntegerTickUnits();
rangeAxis.setStandardTickUnits(units);
```

For greater control over the tick sizes or formatting, create your own `TickUnits` object.

#### 24.27.7 Specifying a Formatting Override

For convenience, you can supply a `NumberFormat` instance as an override for the tick label formatting.

► public NumberFormat getNumberFormatOverride();  
 Returns the override formatter for the tick labels on the axis. The default value is `null` (no override).  
 ► public void setNumberFormatOverride(NumberFormat formatter);  
 Sets the override formatter for the tick labels on the axis and sends an **AxisChangeEvent** to all registered listeners. You can set this to `null` to revert to using the standard formatters.

For example, to format the tick labels along the axis as percentages, you could use the following:

```
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setNumberFormatOverride(new DecimalFormat('0.00%'));
```

### 24.27.8 Methods

When the *auto-tick-unit-selection* flag is set to `true`, the axis will select a tick unit from a set of standard tick units. You can define your own standard tick units for an axis with the following method:

```
► public void setStandardTickUnits(TickUnits units);
Sets the standard tick units for the axis.
```

You don't have to use the auto tick units mechanism. To specify a fixed tick size (and format):

```
► public void setTickUnit(NumberTickUnit unit);
Sets a fixed tick unit for the axis. This allows you to control the size and format of the ticks, but you need to be sure to choose a tick size that doesn't cause the tick labels to overlap.
```

```
► public void setTickUnit(NumberTickUnit unit, boolean notify, boolean turnOffAutoSelect);
Sets the current tick unit for the axis and, if notify is true, sends an AxisChangeEvent to all registered listeners. If turnOffAutoSelect is true, this method sets autoTickUnitSelection to false.
```

```
► public NumberTickUnit getTickUnit();
Returns the current tick unit for the axis. This controls the spacing between tick marks along the axis, and also the format of the tick labels.
```

The following methods provide access to marker bands for the axis (these are currently broken, so you should not use these methods):

```
► public MarkerAxisBand getMarkerBand();
Returns the marker band for the axis, or null if no band is installed.
```

```
► public void setMarkerBand(MarkerAxisBand band);
Sets the marker band for the axis and sends an AxisChangeEvent to all registered listeners.
```

### 24.27.9 Coordinate Translation

A core function of the axis is to translate values between data space (axis coordinates) and Java2D space (for rendering on the screen or some other output target). This is handled via the following methods:

```
► public double valueToJava2D(double value, Rectangle2D area, RectangleEdge edge);
Translates a value along the axis scale to a value in Java2D space, assuming that the axis runs along the specified edge of the given area.
```

```
► public double java2DToValue(double java2DValue, Rectangle2D area, RectangleEdge edge);
Translates a Java2D coordinate to a value on the axis scale, assuming that the axis runs along the specified edge of the given area.
```

### 24.27.10 Other Methods

The remaining methods are typically used by other JFreeChart components—you won't normally call these methods yourself:

```
► public void configure();
Reconfigures the axis—this updates the axis range if the auto-range calculation flag is set.
```

```
► public List refreshTicks(Graphics2D g2, AxisState state, Rectangle2D dataArea,
RectangleEdge edge);
Creates and returns a list of ticks for display along the axis. This list is refreshed every time the axis is drawn. You can override this method to take full control of the values along the axis that will display tick labels.
```

```
► public AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea,
Rectangle2D dataArea, RectangleEdge edge, PlotRenderingInfo plotState);
Draws the axis along the given edge of the specified dataArea.
```

### 24.27.11 Standard Tick Units

```
► public static TickUnitSource createStandardTickUnits();
>Returns a collection of standard sizes (and label formats) for the ticks along the axis.
```

```
► public static TickUnitSource createStandardTickUnits(Locale locale);
>As for the previous method, except the standard number format for the given locale is used to format the tick labels.
```

```
► public static TickUnitSource createIntegerTickUnits();
>Returns a collection of (integer-only) standard sizes (and associated label formats) for the ticks along the axis.
```

```
► public static TickUnitSource createIntegerTickUnits(Locale locale);
>As for the previous method, except the standard integer format for the given locale is used to format the tick labels.
```

### 24.27.12 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
>Tests this axis for equality with an arbitrary object.
```

```
► public int hashCode();
>Returns a hash code for the axis.
```

### 24.27.13 Notes

Some points to note:

- you can reverse the direction of the values on the axis by calling `setInverted(true)`—this method is inherited from the `ValueAxis` class;
- this class defines a default set of standard tick units. You can override the default settings by calling the `setStandardTickUnits()` method.

#### See Also

[ValueAxis](#), [TickUnits](#).

## 24.28 NumberAxis3D

### 24.28.1 Overview

An extension of the `NumberAxis` class that adds a 3D effect. The offset for the 3D effect is obtained from the plot's main renderer, assuming that it implements the `Effect3D` interface.

### 24.28.2 Constructors

There are two constructors:

```
► public NumberAxis3D();
>Creates a new axis with no label.
```

```
► public NumberAxis3D(String label);
>Creates a new axis with the specified label (null is permitted).
```

### 24.28.3 Methods

The 3D effect is implemented by overriding the drawing method:

```
➔ public AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea,
    Rectangle2D dataArea, RectangleEdge edge, PlotRenderingInfo plotState);
    Draws the axis with a 3D effect (the offsets for the 3D effect are obtained from the plot's main
    renderer).
```

### 24.28.4 Notes

Ideally, this class will be combined with the [NumberAxis](#) class.

#### See Also

[CategoryAxis3D](#).

## 24.29 NumberTick

### 24.29.1 Overview

A class used to represent a single tick on a [NumberAxis](#).

### 24.29.2 Usage

This class is used internally and it is unlikely that you should ever need to use it directly.

### 24.29.3 Constructors

To create a new instance:

```
➔ public NumberTick(Number number, String label, TextAnchor textAnchor,
    TextAnchor rotationAnchor, double angle);
    Creates a new instance.

➔ public NumberTick(TickType tickType, double value, String label, TextAnchor textAnchor,
    TextAnchor rotationAnchor, double angle); [1.0.7]
    Creates a new instance with the specified type.4
```

### 24.29.4 Methods

In addition to the methods inherited from [ValueTick](#), this class defines the following method:

```
➔ public Number getNumber();
    Returns the numerical value associated with this tick.
```

### 24.29.5 Notes

Instances of this class are created on-the-fly during the chart rendering process—they’re never used to represent a chart structure, so there’s no need to support cloning and serialization (although it probably wouldn’t hurt to add this).

#### See Also:

[ValueTick](#)

---

<sup>4</sup>For now, the tick type is used to support major and minor tick marks in the [LogAxis](#) class.

## 24.30 NumberTickUnit

### 24.30.1 Overview

A number tick unit for use by subclasses of [NumberAxis](#) (extends the [TickUnit](#) class).

### 24.30.2 Usage

There are two ways that this class is typically used.

The first is where you know the exact tick size that you want for an axis. In this case, you create a new tick unit then call the `setTickUnit()` method in the [ValueAxis](#) class. For example:

```
XYPlot plot = (XYPlot) chart.getPlot();
ValueAxis axis = plot.getRangeAxis();
axis.setTickUnit(new NumberTickUnit(25.0));
```

The second is where you prefer to leave the axis to automatically select a tick unit. In this case, you should create a collection of tick units (see the [TickUnits](#) class for details).

### 24.30.3 Constructors

To create a new number tick unit:

- `public NumberTickUnit(double size);`  
Creates a new number tick unit with a default number formatter for the current locale.

Alternatively, you can supply your own number formatter:

- `public NumberTickUnit(double size, NumberFormat formatter);`  
Creates a new number tick unit with the specified number formatter.
- `public NumberTickUnit(double size, NumberFormat formatter, int minorTickCount);`  
Creates a new number tick unit with the specified number formatter and minor tick count.

### 24.30.4 Methods

To format a value using the tick unit's internal formatter:

- `public String valueToString(double value);`  
Formats the value as a [String](#) using the internal number formatter. This method is usually called by code in one of the axis classes (for example, [NumberAxis](#)).

### 24.30.5 Equals, Cloning and Serialization

To test this object for equality:

- `public boolean equals(Object obj);`  
Tests this object for equality with an arbitrary object. If `obj` is `null`, this method returns `false`.

Instances of this class are immutable, so the class does not implement [Cloneable](#). The class is [Serializable](#).

### 24.30.6 Notes

This class is immutable, a requirement for all subclasses of [TickUnit](#).

#### See Also

[DateTickUnit](#).

## 24.31 PeriodAxis

### 24.31.1 Overview

A date/time axis with the following features:

- supports multiple label bands, where each band is divided up into time periods;
- automatic range calculation based on (whole unit) time periods;
- a user specified time zone;

See figure 24.7 for an example. You can use this axis in place of a `DateAxis`, it does a similar job but with a slightly different set of features.

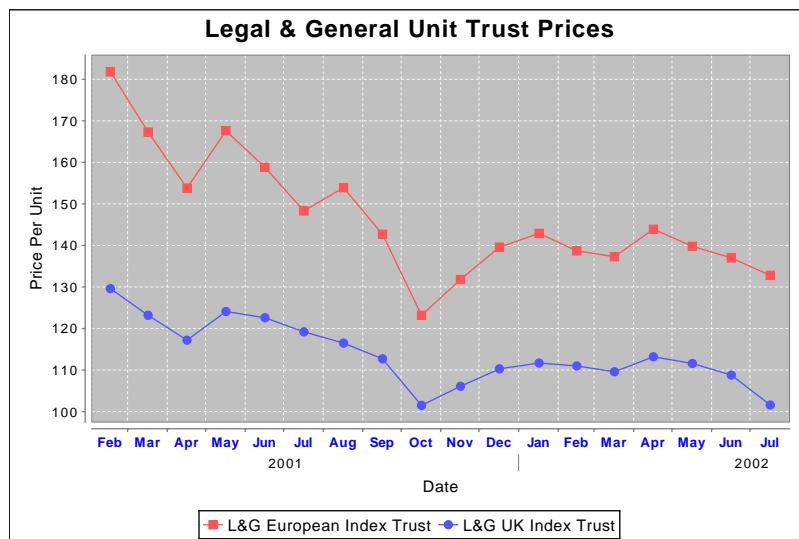


Figure 24.7: A chart that uses a `PeriodAxis` (see `PeriodAxisDemo1.java`)

### 24.31.2 Constructors

To create a new axis:

```
➔ public PeriodAxis(String label,
RegularTimePeriod first, RegularTimePeriod last);
```

Creates a new axis—calls the next constructor, passing it the default time zone.

```
➔ public PeriodAxis(String label,
RegularTimePeriod first, RegularTimePeriod last, TimeZone timeZone);
```

Creates a new axis that displays data from the `first` to the `last` time periods. All time periods are evaluated within the specified `timeZone`.

### 24.31.3 The Axis Range

The axis range is defined by two time periods:

```
➔ public RegularTimePeriod getFirst();
```

Returns the time period that defines the start of the range of values displayed by the axis.

```
➔ public RegularTimePeriod getLast();
```

Returns the time period that defines the end of the range of values displayed by the axis.

Alternatively, you can get the range (bounds specified in milliseconds):

► public Range getRange();

Returns the current axis range. The lower bound of the range is set to the first millisecond of the first time period, and the upper bound of the range is set to the last millisecond of the last time period. The time zone is taken into account when pegging the first and last time periods to the millisecond time line.

The axis range can be specified manually or automatically calculated by JFreeChart to “fit” the available data values. To specify a manual range, use the following methods:

► public void setFirst(RegularTimePeriod first);

Sets the time period that defines the start of the range of values displayed by the axis, and sends an `AxisChangeEvent` to all registered listeners.

► public void setLast(RegularTimePeriod last);

Sets the time period that defines the end of the range of values displayed by the axis, and sends an `AxisChangeEvent` to all registered listeners.

To have the axis range calculated automatically, use the `setAutoRange()` method inherited from the `ValueAxis` class. In addition, you may want to specify the time period class used by the auto-range calculation—the axis range will always include a whole number of time periods of the class specified:

► public Class getAutoRangeTimePeriodClass();

Returns the time period class used when the axis range is calculated automatically.

► public void setAutoRangeTimePeriodClass(Class c);

Sets the time period class used when the axis range is calculated automatically. The axis range will always be a whole number of periods. Valid classes include: `Year.class`, `Quarter.class`, `Month.class`, `Week.class`, `Day.class`, `Hour.class`, `Minute.class`, `Second.class` and `Millisecond.class`.

#### 24.31.4 Axis Labelling

The axis supports one or more “bands” of labels, where each band is represented by an instance of `PeriodAxisLabelInfo`. Use the following methods to get/set the band definitions:

► public PeriodAxisLabelInfo[] getLabelInfo();

Returns an array of objects where each object defines the format for one band of labels along the axis.

► public void setLabelInfo(PeriodAxisLabelInfo[] info);

Sets an array of objects where each object defines the format for one band of labels along the axis.

Examples of specifying label bounds can be found in the `PeriodAxisDemo1` and `PeriodAxisDemo2` classes, included in the JFreeChart Demo distribution.

#### 24.31.5 Time Zones

In order to “peg” time periods to the absolute time line (in Java, measured in milliseconds since 1-Jan-1970 GMT), you need to specify a time zone. Use the following methods:

► public TimeZone getTimeZone();

Returns the `TimeZone` used to “peg” time periods to the absolute time line.

► public void setTimeZone(TimeZone zone);

Sets the `TimeZone` that is used to “peg” time periods to the absolute time line.

#### 24.31.6 Other Methods

The remaining methods defined by this class are mostly for internal use:

► public double valueToJava2D(double value, Rectangle2D area, RectangleEdge edge);

Converts a data value to a Java2D coordinate, assuming that the axis lies along the specified `edge` of the given `area`.

```
→ public double java2DToValue(double java2DValue, Rectangle2D area, RectangleEdge edge);
Converts a Java2D coordinate back into a data value, assuming that the axis lies along the
specified edge of the given area.
```

```
→ public void configure();
Configures the axis for use. This method is usually called by the plot when the axis is first
assigned to the plot, because a new plot means a new set of data and therefore the axis range
may need to be updated. You won't normally need to call this method yourself.
```

```
→ public AxisSpace reserveSpace(Graphics2D g2, Plot plot, Rectangle2D plotArea,
RectangleEdge edge, AxisSpace space);
Reserves additional space in space to allow room for this axis to be displayed. This method is
called by the plot during the process of laying out and drawing the chart, you won't normally
need to call this method yourself.
```

```
→ public AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea,
Rectangle2D dataArea, RectangleEdge edge, PlotRenderingInfo plotState);
Draws the axis. This method is called by the plot, you won't normally need to call it yourself.
```

```
→ public List refreshTicks(Graphics2D g2, AxisState state, Rectangle2D plotArea,
Rectangle2D dataArea, RectangleEdge edge);
For this axis, this method returns an empty list.
```

### 24.31.7 Equals, Cloning and Serialization

This class overrides the `equals()` method from the `Object` class:

```
→ public boolean equals(Object obj);
Tests this axis for equality with an arbitrary object. Another object is considered equal if it is
a PeriodAxis with the same attributes as this axis.
```

The axis is `Cloneable` and `PublicCloneable`:

```
→ public Object clone() throws CloneNotSupportedException;
Returns a clone of the axis.
```

The axis is `Serializable`.

### 24.31.8 Notes

Some points to note:

- two demos (`PeriodAxisDemo1.java` and `PeriodAxisDemo2.java`) are included in the JFreeChart demo collection.

#### See Also

[DateAxis](#), [PeriodAxisLabelInfo](#).

## 24.32 PeriodAxisLabelInfo

### 24.32.1 Overview

A helper class that records the information for one “band” of labels on a `PeriodAxis`. When you are specifying the label bands for the axis, you create an array of `PeriodAxisLabelInfo` objects—for example:

```
PeriodAxisLabelInfo[] info = new PeriodAxisLabelInfo[2];
info[0] = new PeriodAxisLabelInfo(Month.class, new SimpleDateFormat("MMM"));
info[1] = new PeriodAxisLabelInfo(Year.class, new SimpleDateFormat("yyyy"));
domainAxis.setLabelInfo(info);
```

In the above example, there are two bands. The first band is split into 1 month time periods and the second band is split into 1 year time periods. The sample code comes from the `PeriodAxisDemo1.java` file that is included in the JFreeChart Demo distribution.

### 24.32.2 Constructors

To create a new instance:

```
→ public PeriodAxisLabelInfo(Class periodClass, DateFormat dateFormat);
Creates a new instance based on the specified periodClass (see below). The dateFormat used to
format the labels for each time period.

→ public PeriodAxisLabelInfo(Class periodClass, DateFormat dateFormat,
RectangleInsets padding, Font labelFont, Paint labelPaint,
boolean drawDividers, Stroke dividerStroke, Paint dividerPaint);
Creates a new instance based on the specified periodClass (see below). The dateFormat is used
to format the labels for each time period. The padding controls the minimum gap between time
period labels. The remaining arguments control the appearance of the labels and the (optional)
dividing lines between labels.
```

When constructing an instance of this class, you need to specify the class of time period that you want to use for labelling purposes. This is usually one of the following: `Year.class`, `Quarter.class`, `Month.class`, `Week.class`, `Day.class`, `Hour.class`, `Minute.class`, `Second.class` or `Millisecond.class`.

### 24.32.3 Methods

The following methods are defined:

```
→ public Class getPeriodClass();
Returns the specific class used to represent time periods—it should be some subclass of RegularTimePeriod.

→ public DateFormat getDateFormat();
Returns the formatter for the date labels.

→ public RectangleInsets getPadding();
Returns the padding that controls the minimum space between labels.

→ public Font getLabelFont();
Returns the Font used to display labels for each time period.

→ public Paint getLabelPaint();
Returns the Paint that is used as the foreground color when displaying labels for each time
period.

→ public boolean getDrawDividers();
Returns a flag that determines whether or not dividers are drawn between time periods.

→ public Stroke getDividerStroke();
Returns the Stroke used to draw dividers between time periods.

→ public Paint getDividerPaint();
Returns the Paint used to draw dividers between time periods.

→ public RegularTimePeriod createInstance(Date millisecond, TimeZone zone);
Creates a time period that includes the specified millisecond, taking into account the time zone.
The time period will be an instance of the class returned by the getPeriodClass() method.
```

### 24.32.4 Equals, Cloning and Serialization

To test this instance for equality with another object:

```
→ public boolean equals(Object obj);
Tests this instance for equality with an arbitrary object. This method will return true if obj is
an instance of PeriodAxisLabelInfo with equivalent settings to this instance.
```

To make a clone of this instance:

```
→ public Object clone() throws CloneNotSupportedException;
Creates a clone of this object.
```

This class is `Serializable`.

## 24.33 QuarterDateFormat

### 24.33.1 Overview

A subclass of `DateFormat` that is used to convert a `Date` to a `String`. The default format is “YYYY Q” where “YYYY” is replaced by the year and “Q” is replaced by a symbol representing the quarter (symbols can be defined via the constructor).

Any symbols can be used to represent the four quarters in a year, but the following default symbol sets are provided:

- `REGULAR_QUARTERS` – the symbols “1”, “2”, “3” and “4”;
- `ROMAN_QUARTERS` – the symbols “I”, “II”, “III”, and “IV”;
- `GREEK_QUARTERS` – greek symbols (since 1.0.6).

### 24.33.2 Constructors

The following constructors are available:

- ▶ `public QuarterDateFormat();`  
Equivalent to `QuarterDateFormat(TimeZone.getDefault())`—see the next constructor.
- ▶ `public QuarterDateFormat(TimeZone zone);`  
Equivalent to `QuarterDateFormat(zone, REGULAR_QUARTERS)`—see the next constructor.
- ▶ `public QuarterDateFormat(TimeZone zone, String[] quarterSymbols);`  
Equivalent to `QuarterDateFormat(zone, quarterSymbols, false)`—see the next constructor.
- ▶ `public QuarterDateFormat(TimeZone zone, String[] quarterSymbols, boolean quarterFirst);`  
[1.0.6]  
Creates a new date formatter linked to the specified `zone` using the supplied symbols for the four quarters (the array should have four entries). The `quarterFirst` flag controls whether the quarter is displayed before or after the year (for example, “2007-IV” or “IV-2007”).

### 24.33.3 Methods

The `format` method is overridden to create the formatted version of the given `Date`:

- ▶ `public StringBuffer format(Date date, StringBuffer toAppendTo, FieldPosition fieldPosition);`  
Returns a string representing the given date. The string contains the year followed by a space followed by the symbol corresponding to the quarter in which the date falls (the symbols are supplied in the constructor).

The `parse` method is overridden but not implemented:

- ▶ `public Date parse(String source, ParsePosition pos);`  
This method has not been implemented, it simply returns `null`.

### 24.33.4 Notes

Some points to note:

- a demo (`QuarterDateFormatDemo.java`) showing this class being used with a `PeriodAxis` is included in the JFreeChart demo collection.

## 24.34 SegmentedTimeline

### 24.34.1 Overview

A segmented timeline for use with a `DateAxis`.

### 24.34.2 Usage

Please refer to the Javadocs.

## 24.35 StandardTickUnitSource

### 24.35.1 Overview

A `TickUnitSource` that dynamically creates tick units where the tick size is an integer power of 10, and the number format is `DecimalFormat("0.0E0")`. The primary advantage of this source is that the tick size is calculated dynamically, so it can handle very large and very small axis ranges (unlike the `TickUnits` class which contains a finite collection of tick sizes).

### 24.35.2 Usage

To use this `TickUnitSource` with a `NumberAxis`, create a new instance and install it as follows:

```
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
TickUnitSource units = new StandardTickUnitSource();
rangeAxis.setStandardTickUnits(units);
```

### 24.35.3 Constructor

This class has a single constructor:

```
► public StandardTickUnitSource();
Creates a new instance. There are no customisable attributes for this class.
```

### 24.35.4 Methods

This class implements the three methods defined in the `TickUnitSource` method. These methods are called by the axis, you won't normally need to call these methods directly:

```
► public TickUnit getLargerTickUnit(TickUnit unit);
Returns the next larger tick unit relative to unit.

► public TickUnit getCeilingTickUnit(TickUnit unit);
Returns a tick unit that is either equal to unit or the next larger tick unit.

► public TickUnit getCeilingTickUnit(double size);
Returns a tick unit that is equal in size to size, or the next larger tick unit.
```

### 24.35.5 Equals, Cloning and Serialization

This class overrides the `equals()` method.<sup>5</sup>

```
► public boolean equals(Object obj);
Tests this tick unit source for equality with an arbitrary object.
```

Instances of this class are `Serializable` but not `Cloneable` (cloning is unnecessary, since instances of this class are immutable).

### 24.35.6 Notes

Some points to note:

- this class is not used by default, but can be installed in an axis if necessary—see `SmallNumberDemo.java` in the JFreeChart demo collection for an example.

---

<sup>5</sup>Since version 1.0.7.

## 24.36 SubCategoryAxis

### 24.36.1 Overview

An extension of the `CategoryAxis` class that allows subcategories to be displayed along the domain axis for a `CategoryPlot`. This type of axis can be usefully employed along with the `GroupedStackedBarRenderer` class.

### 24.36.2 Constructors

To create a new instance:

```
► public SubCategoryAxis(String label);
Creates a new axis with the specified label (which may be null).
```

### 24.36.3 General Attributes

To control the font used to display the sub-labels:

```
► public Font getSubLabelFont();
Returns the sub-label font (never null). The default value is Font("SansSerif", Font.PLAIN, 10).

► public void setSubLabelFont(Font font);
Sets the sub-label font and sends an AxisChangeEvent to all registered listeners. If font is null, this method throws an IllegalArgumentException.

► public Paint getSubLabelPaint();
Returns the paint used to draw the sub-labels (never null). The default value is Color.black.

► public void setSubLabelPaint(Paint paint);
Sets the sub-label paint and sends an AxisChangeEvent to all registered listeners. If paint is null, this method throws an IllegalArgumentException.
```

### 24.36.4 Subcategories

The subcategories for the axis need to be specified manually. All the subcategories are displayed within each category along the axis. The categories are driven by the content of the dataset, while the subcategories are just arbitrary labels with no formal connection to the dataset.

```
► public void addSubCategory(Comparable subCategory);
Adds a subcategory to the axis.
```

### 24.36.5 Other Methods

The remaining methods are used internally:

```
► public AxisSpace reserveSpace(...);
Calculates the amount of space required to draw the axis. This overrides the method defined in the CategoryAxis class, because additional space is required to draw the subcategory labels.

► public AxisState draw(...);
Overrides the draw() method in the CategoryAxis class to include the sublabels in the axis.

► protected AxisState drawSubCategoryLabels(...);
Draws the subcategory labels.
```

### 24.36.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this axis for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 24.36.7 Notes

A couple of demos (`SubCategoryAxisDemo1.java` and `StackedBarChartDemo4.java`) are included in the JFreeChart demo collection.

#### See Also

[GroupedStackedBarRenderer](#).

## 24.37 SymbolAxis

### 24.37.1 Overview

A `ValueAxis` that maps integer values (starting at zero) to symbols (strings). This can be used to present:

- a `CategoryPlot` with pseudo-categories displayed along the range axis (y-axis);
- an `XYPLOT` with pseudo-categories displayed along the domain axis (x-axis) and/or range axis (y-axis).

### 24.37.2 Constructors

To create a new axis:

► `public SymbolAxis(String label, String[] sv);`

Creates a new axis with the specified label. The `sv` array contains the strings that are displayed along the axis for the integer values.

### 24.37.3 Attributes

To access the symbols used for the integer values along the axis:

► `public String[] getSymbols();`

Returns the symbols used by the axis. These are the symbols that were specified in the constructor. The returned array is a copy, so modifying it will not change the axis.

To access the flag that controls whether or not grid bands are painted for alternate tick values:

► `public boolean isGridBandsVisible();`

Returns the flag that controls whether or not the alternating grid bands are drawn for the axis. The default value is `true`.

► `public void setGridBandsVisible(boolean flag);`

Sets the flag that controls whether or not the alternating grid bands are drawn for the axis, and sends an `AxisChangeEvent` to all registered listeners.

To access the grid band paint:

► `public Paint getGridBandPaint();`

Returns the paint used to color alternate bands within the plot area. The default value is `Color(232, 234, 232, 128)` (a light gray with partial transparency).

► `public void setGridBandPaint(Paint paint);`

Sets the paint used to color alternate bands within the plot area, and sends a `AxisChangeEvent` to all registered listeners. An `IllegalArgumentException` will be thrown if `paint` is `null`.

From version 1.0.7 onwards, you can specify the alternate grid band colour as well:

► `public Paint getGridBandAlternatePaint(); [1.0.7]`

Returns the paint (never `null`) used fill alternate bands within the plot area. The default value is `Color(0, 0, 0, 0)` (that is, completely transparent). See also `getGridBandPaint()`.

► `public void setGridBandAlternatePaint(Paint paint); [1.0.7]`

Sets the paint used to fill alternate bands within the plot area, and sends an `AxisChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

#### 24.37.4 Other Methods

Most of the other methods in this class are used internally:

```
► public String valueToString(double value);
    Returns the symbol for the given value. The value is rounded to an integer, then the symbol
    is obtained from the array of symbols defined for the axis. If value is out of range, an empty
    string is returned. This method is called by the refreshTicks() code.

► public AxisState draw(Graphics2D g2, double cursor, Rectangle2D plotArea,
    Rectangle2D dataArea, RectangleEdge edge, PlotRenderingInfo plotState);
    Called by the plot to draw the axis. You won't normally call this method yourself.

► protected void autoAdjustRange();
    Adjusts the axis range to fit the data. In this case, the axis range is fixed, so this method
    should not do anything.

► public List refreshTicks(Graphics2D g2, AxisState state, Rectangle2D dataArea,
    RectangleEdge edge);
    Returns a list of ticks for display on the axis. This method is called by internal code, you won't
    normally call it yourself.

► protected List refreshTicksHorizontal(Graphics2D g2, Rectangle2D dataArea,
    RectangleEdge edge);
    Creates a list of ticks for the axis when it is displayed "horizontally". That is, at the top or
    bottom of the plot.

► protected List refreshTicksVertical(Graphics2D g2, Rectangle2D dataArea,
    RectangleEdge edge);
    Creates a list of ticks for the axis when it is displayed "vertically". That is, at the left or right
    of the plot.

► protected void drawGridBands(Graphics2D g2, Rectangle2D plotArea,
    Rectangle2D dataArea, RectangleEdge edge, List ticks);
    Draws the grid bands.

► protected void drawGridBandsHorizontal(Graphics2D g2, Rectangle2D plotArea,
    Rectangle2D dataArea, boolean firstGridLineIsDark, List ticks);
    Draws the grid bands for a horizontal axis.

► protected void drawGridBandsVertical(Graphics2D g2, Rectangle2D drawArea,
    Rectangle2D plotArea, boolean firstGridLineIsDark, List ticks);
    Draws the grid bands for a vertical axis.

► protected void selectAutoTickUnit(Graphics2D g2, Rectangle2D dataArea,
    RectangleEdge edge);
    Throws an UnsupportedOperationException.
```

#### 24.37.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
    Tests this axis for equality with an arbitrary object. To be considered equal, obj must be non-
    null, an instance of SymbolAxis, have the same list of symbols as this axis, and super.equals(obj)
    must return true.
```

Instances of this class are `Cloneable` and `Serializable`.

#### 24.37.6 Notes

Some points to note:

- a demo for a `CategoryPlot` (`LineChartDemo8.java`) is included in the JFreeChart demo distribution;
- a demo for an `XYPlot` (`SymbolAxisDemo1.java`) is included in the JFreeChart demo distribution.

## 24.38 Tick

### 24.38.1 Overview

A utility class representing a tick on an axis. Used temporarily during the drawing process only—you won't normally use this class yourself.

#### See Also

[TickUnit](#).

## 24.39 TickType

### 24.39.1 Overview

This class defines tokens representing the tick type for an axis (see the [NumberTick](#) class). This class was introduced in JFreeChart version 1.0.7.

- `TickType.MAJOR`;
- `TickType.MINOR`;

### 24.39.2 Notes

The `TickType` tokens are used by the [NumberTick](#) class to support major and minor tick marks on the [LogAxis](#) class. Eventually, this support will be rolled out to the [NumberAxis](#) class also.

## 24.40 TickUnit

### 24.40.1 Overview

An abstract class representing a tick unit, with subclasses including:

- `DateTickUnit` – for use with a [DateAxis](#);
- `NumberTickUnit` – for use with a [NumberAxis](#).

### 24.40.2 Constructors

The standard constructor:

```
➔ public TickUnit(double size);
Equivalent to TickUnit(size, 0)—see the next constructor.

➔ public TickUnit(double size, int minorTickCount); [1.0.7]
Creates a new tick unit with the specified size and minor tick count.
```

### 24.40.3 General Methods

To get the tick size:

```
➔ public double getSize();
Returns the size of the tick unit—that is, the gap (in data units) between consecutive tick
marks along the axis. The value is specified in the constructor.
```

To get the minor tick count:

```
➔ public int getMinorTickCount();
Returns the number of minor tick units between consecutive major tick units.
```

To convert a data value to a string:

► `public String valueToString(double value);`  
 Returns a string representing the specified data value. Subclasses may override this method to provide custom formatting.

► `public int compareTo(Object object);`  
 Compares this instance to an arbitrary object. This method is defined by the `Comparable` interface, and is used to order tick units by size.

#### 24.40.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`  
 Tests this unit for equality with an arbitrary object.

Instances of this class are immutable and `Serializable`.

#### 24.40.5 Notes

Implements the `Comparable` interface, so that a collection of tick units can be sorted easily using standard Java methods. In particular, the `StandardTickUnitSource` class makes use of this feature.

#### See Also

[TickUnits](#).

### 24.41 TickUnits

#### 24.41.1 Overview

A collection of tick units. This class is used by the `DateAxis` and `NumberAxis` classes to store a list of “standard” tick units. The *auto-tick-unit-selection* mechanism chooses one of the standard tick units in order to maximise the number of ticks displayed without having the tick labels overlap.

#### 24.41.2 Constructors

The default constructor:

► `public TickUnits();`  
 Creates a new collection of tick units, initially empty.

#### 24.41.3 Methods

To add a new tick unit to the collection:

► `public void add(TickUnit unit);`  
 Adds the tick unit to the collection.

To find the tick unit in the collection that is the next largest in size compared to the specified tick unit:

► `public TickUnit getLargerTickUnit(TickUnit unit);`  
 Returns the tick unit that is one size larger than the specified unit.

#### 24.41.4 Notes

The `NumberAxis` class has a static method `createStandardTickUnits()` that generates a tick unit collection (of standard tick sizes) for use by numerical axes.

#### See Also

[TickUnit](#).

## 24.42 TickUnitSource

### 24.42.1 Overview

The interface through which a `ValueAxis` finds a suitable tick unit. Classes that implement this interface include:

- `TickUnits`;
- `StandardTickUnitSource`;

### 24.42.2 Methods

The following methods allow a `TickUnit` to be obtained from the source:

- ➔ public `TickUnit getLargerTickUnit(TickUnit unit)`;  
Returns a tick unit that is larger than the supplied `unit`.
- ➔ public `TickUnit getCeilingTickUnit(TickUnit unit)`;  
Returns a tick unit that is equal to or larger in size than the specified `unit`.
- ➔ public `TickUnit getCeilingTickUnit(double size)`;  
Returns a tick unit with size equal to or larger than the specified `size`.

## 24.43 Timeline

### 24.43.1 Overview

The interface that defines the methods for a timeline that can be used with a `DateAxis`.

### 24.43.2 Methods

The interface declares the following methods:

- ➔ public long `toTimelineValue(long millisecond)`;  
Translates a millisecond (as defined by `java.util.Date`) into an index along this timeline.
- ➔ public long `toTimelineValue(Date date)`;  
Translates a `Date` into an index along the timeline.
- ➔ public long `toMillisecond(long timelineValue)`;  
Converts a timeline index back into a millisecond. Note that many timeline index values can map to a single millisecond.
- ➔ public boolean `containsDomainValue(long millisecond)`;  
Returns `true` if the millisecond is contained within the timeline, and `false` otherwise.
- ➔ public boolean `containsDomainValue(Date date)`;  
Returns `true` if the date is contained within the timeline, and `false` otherwise.
- ➔ public boolean `containsDomainRange(long fromMillisecond, long toMillisecond)`;  
Returns `true` if the range of millisecond values is contained within the timeline, and `false` otherwise.
- ➔ public boolean `containsDomainRange(Date fromDate, Date toDate)`;  
Returns `true` if the range of dates is contained within the timeline, and `false` otherwise.

### 24.43.3 Notes

The `SegmentedTimeline` class implements this interface.

## 24.44 ValueAxis

### 24.44.1 Overview

The base class for all axes that display “values”, with the two key subclasses being [NumberAxis](#) and [DateAxis](#).

At the lowest level, the axis values are manipulated as `double` primitives, obtained from the `Number` objects supplied by the plot’s dataset.

### 24.44.2 Constructors

The constructors for this class are protected, you cannot create a `ValueAxis` directly—you must use a subclass.

### 24.44.3 Attributes

The attributes maintained by this class, in addition to those that it inherits from the `Axis` class, are listed in Table 24.6. There are methods to read and update most of these attributes. In general, updating an axis attribute will result in an `AxisChangeEvent` being sent to all (or any) registered listeners. The default values used to initialise the axis attributes (when necessary) are listed in Table 24.7.

Attribute:	Description:
<code>inverted</code>	A flag that is used to “invert” the axis scale.
<code>autoRange</code>	A flag controlling whether or not the axis range is automatically adjusted to fit the range of data values.
<code>defaultAutoRange</code>	The default range when there is no data (since 1.0.5).
<code>fixedAutoRange</code>	If specified, the auto-range is calculated by subtracting this value from the maximum domain value in the dataset.
<code>autoRangeMinimumSize</code>	The smallest axis range allowed when it is automatically calculated.
<code>lowerMargin</code>	The margin to allow at the lower end of the axis scale (expressed as a percentage of the total axis range).
<code>upperMargin</code>	The margin to allow at the upper end of the axis scale (expressed as a percentage of the total axis range).
<code>autoTickUnitSelection</code>	A flag controlling whether or not the tick units are selected automatically.
<code>standardTickUnits</code>	A collection of the “standard” tick units that can be used by this axis.
<code>verticalTickLabels</code>	A flag that controls whether or not the tick labels are rotated 90 degrees.
<code>positiveArrowVisible</code>	A flag that controls whether or not an arrow is drawn at the positive end of the scale.
<code>negativeArrowVisible</code>	A flag that controls whether or not an arrow is drawn at the negative end of the scale.
<code>upArrow</code>	The shape used to draw an arrow at the end of an axis pointing upwards.
<code>downArrow</code>	The shape used to draw an arrow at the end of an axis pointing downwards.
<code>leftArrow</code>	The shape used to draw an arrow at the end of an axis pointing leftwards.
<code>rightArrow</code>	The shape used to draw an arrow at the end of an axis pointing rightwards.

Table 24.6: *Attributes for the ValueAxis class*

### 24.44.4 Usage

To modify the attributes of a `ValueAxis`, you first need to obtain a reference to the axis. For a `CategoryPlot`, you can use the following code:

Name:	Value:
DEFAULT_AUTO_RANGE	true;
DEFAULT_LOWER_BOUND	0.0; [Deprecated, 1.0.5]
DEFAULT_UPPER_BOUND	1.0; [Deprecated, 1.0.5]
DEFAULT_UPPER_MARGIN	0.05 (5 percent)
DEFAULT_LOWER_MARGIN	0.05 (5 percent)

Table 24.7: *ValueAxis* class default attribute values

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
ValueAxis rangeAxis = plot.getRangeAxis();
// modify the axis here...
```

The code for an [XYPlot](#) is very similar, except that the domain axis is also a [ValueAxis](#) in this case:

```
XYPlot plot = (XYPlot) chart.getPlot();
ValueAxis domainAxis = plot.getDomainAxis();
ValueAxis rangeAxis = plot.getRangeAxis();
// modify the axes here...
```

Having obtained an axis reference, you can:

- control the axis range, see section [24.44.5](#);
- invert the axis scale, see section [24.44.6](#);

## 24.44.5 The Axis Range

The *axis range* defines the highest and lowest values that will be displayed on axis. On a chart, it is typically the case that data values outside the axis range are clipped, and therefore not visible on the chart.

### Automatic Bounds Calculation

By default, JFreeChart is configured to automatically calculate axis ranges so that all of the data in your dataset is visible. It does this by determining the highest and lowest values in your dataset, adding a small margin (to prevent the data being plotted right up to the edge of a chart), and setting the axis range. To control whether or not the axis range is automatically adjusted to fit the available data:

► **public boolean isAutoRange();**  
 Returns the flag that controls whether the axis range is automatically updated to reflect the data values.

► **public void setAutoRange(boolean auto);**  
 Sets the flag that controls whether or not the axis range is automatically adjusted to fit the available data values, and sends an [AxisChangeEvent](#) to all registered listeners.

► **protected void setAutoRange(boolean auto, boolean notify);**  
 An alternative version of the above method that lets you specify whether or not the listeners are notified.

When the axis range is calculated automatically, a margin is added to the lower and upper bounds (the default is 0.05 or 5 percent):

► **public double getLowerMargin();**  
 Returns the lower margin as a percentage of the overall axis length (the default is 0.05 or 5 percent).

► **public void setLowerMargin(double margin);**  
 Sets the lower margin (specified as a percentage of the overall axis length) and sends an [AxisChangeEvent](#) to all registered listeners.

► public double getUpperMargin();

Returns the upper margin as a percentage of the overall axis length (the default is 0.05 or 5 percent).

► public void setUpperMargin(double margin);

Sets the upper margin (specified as a percentage of the overall axis length) and sends an [AxisChangeEvent](#) to all registered listeners.

Note that the margins are only applied when the axis bounds are automatically calculated. If you set the axis bounds manually (see the next section) then the margins are ignored.

If the plot has no data, then the auto range is set to the default:

► public Range getDefaultAutoRange(); [1.0..5]

Returns the default auto range (never null).

► public void setDefaultAutoRange(Range range); [1.0..5]

Sets the default auto range and sends an [AxisChangeEvent](#) to all registered listeners. If `range` is null, this method throws an [IllegalArgumentException](#).

### Setting the Range Manually

To manually set the axis range (which automatically disables the *auto-range* flag):

► public void setRange(double lower, double upper);

Equivalent to `setRange(new Range(lower, upper))`—see below.

► public void setRange(Range range);

Equivalent to `setRange(range, true, true)`—see below.

► public void setRange(Range range, boolean turnOffAutoRange, boolean notify);

Sets the bounds of the axis so that it will display the range of values specified by `range`. If `notify` is `true`, an [AxisChangeEvent](#) is sent to all registered listeners. If `turnOffAutoRange` is `true`, the `autoRange` flag is set to false (which is what you want if you intend to control the axis range manually).

To set the lower bound for the axis:

► public void setLowerBound(double value);

Sets the lower bound for the axis. If the *auto-range* attribute is `true` it is automatically switched to `false`. Registered listeners are notified of the change.

To set the upper bound for the axis:

► public void setUpperBound(double value);

Sets the upper bound for the axis. If the *auto-range* attribute is `true` it is automatically switched to `false`. Registered listeners are notified of the change.

### 24.44.6 Inverting the Axis Scale

There is a flag that can be used to “invert” the axis scale:

► public boolean isInverted();

Returns the flag that controls whether or not the axis scale is inverted.

► public void setInverted(boolean flag);

Sets the flag that controls whether or not the axis scale is inverted and sends an [AxisChangeEvent](#) to all registered listeners.

### 24.44.7 Tick Labels

Tick labels can be rotated 90 degrees (typically to fit more labels in) by setting the following flag:

► `public boolean isVerticalTickLabels();`

Returns the flag that controls whether or not the tick labels are rotated by 90 degrees. The default value is `false`.

► `public void setVerticalTickLabels(boolean flag);`

Sets the flag that controls whether or not the tick labels are rotated by 90 degrees, and, if the flag value changes, sends a `RendererChangeEvent` to all registered listeners.

For other tick label settings, see section [24.2.7](#).

### 24.44.8 Other Methods

A key function for a `ValueAxis` is to convert a data value to an output (Java2D) coordinate for plotting purposes. The output coordinate will be dependent on the area into which the data is being drawn:

► `public double valueToJava2D(double dataValue, Rectangle2D dataArea, RectangleEdge edge);`

Converts a data value into a co-ordinate along one edge of the `dataArea` rectangle. The caller can pass in an arbitrary rectangle, but typically it should match the rectangle defined by the interior of the chart's axes.

The inverse function converts a Java2D coordinate back to a data value:

► `public double java2DToValue(double java2DValue, Rectangle2D dataArea, RectangleEdge edge);`

Converts a Java2D coordinate (defined relative to one edge of the specified `dataArea`) back to a data value.

To set a flag that controls whether or not the axis tick units are automatically selected:

► `public void setAutoTickUnitSelection(boolean flag);`

Sets a flag (commonly referred to as the *auto-tick-unit-selection* flag) that controls whether or not the tick unit for the axis is automatically selected from a collection of standard tick units.

### 24.44.9 Notes

Some points to note:

- in a `CategoryPlot`, the range axis is required to be a subclass of `ValueAxis`.
- in an `XYPlot`, both the domain and range axes are required to be a subclass of `ValueAxis`.

#### See Also

[Axis](#), [DateAxis](#), [NumberAxis](#).

## 24.45 ValueTick

### 24.45.1 Overview

The base class for the `NumberTick` and `DateTick` classes. Instances of these classes are created in the `refreshTicks()` method of the various axis classes.

### 24.45.2 Constructors

To create a new instance:

```
↳ public ValueTick(double value, String label, TextAnchor textAnchor,
TextAnchor rotationAnchor, double angle);
Creates a new tick with the specified value.

↳ public ValueTick(TickType tickType, double value, String label, TextAnchor textAnchor,
TextAnchor rotationAnchor, double angle); [1.0.7]
Creates a new tick with the specified type and value.
```

### 24.45.3 General Methods

To get the tick type:

```
↳ public TickType getTickType(); [1.0.7]
Returns the tick type (MAJOR or MINOR).
```

To get the value for the tick:

```
↳ public double getValue();
Returns the value for this tick.
```

### 24.45.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
↳ public boolean equals(Object obj);
Tests this tick for equality with an arbitrary object.
```

### 24.45.5 Notes

The minor tick type is used only by the `LogAxis` class at present.

#### See Also

[DateTick](#), [NumberTick](#).

# Chapter 25

## Package: org.jfree.chart.block

### 25.1 Introduction

The `org.jfree.chart.block` package contains classes that are used for laying out rectangular items (blocks) within containers. Primarily, the classes in this package are used by the `LegendTitle` class.

### 25.2 AbstractBlock

#### 25.2.1 Overview

A base class for implementing a `Block`, which is used as a layout unit in JFreeChart (particularly for the `LegendTitle` class). Subclasses include:

- `BlockContainer`;
- `ColorBlock`;
- `EmptyBlock`;
- `LabelBlock`;
- `LegendGraphic`;
- `Title`.

#### 25.2.2 Constructor

To create a new block:

```
► protected AbstractBlock();  
Creates a new block.
```

#### 25.2.3 General Attributes

The block has an identifier:

```
► public String getID();  
Returns the block id. The default value is null.  
  
► public void setID(String id);  
Sets the block id (null is permitted).
```

You can specify the preferred height and width for the block:

```
► public double getHeight();  
Returns the block height.
```

```
→ public void setHeight(double height);
Sets the block height. This is a “preferred” height which may or may not be observed by the
layout manager.
```

```
→ public double getWidth();
Returns the block width.
```

```
→ public void setWidth(double width);
Sets the block width. This is a “preferred” width which may or may not be observed by the
layout manager.
```

The margin is the space around the outside of the block’s border:

```
→ public RectangleInsets getMargin();
Returns the margin around the outside of the block’s border. The default value is
RectangleInsets.ZERO_INSETS.
```

```
→ public void setMargin(RectangleInsets margin);
Sets the margin around the outside of the block’s border.
```

You can specify a frame (or border) for the block:

```
→ public BlockFrame getFrame(); [1.0.5]
Returns the border that will be drawn around the block. The default value is BlockBorder.NONE.
```

```
→ public void setFrame(BlockFrame frame); [1.0.5]
Sets the border that will be drawn around the block. If frame is null, this method throws an
IllegalArgumentException.
```

The padding is an area of whitespace inside the block’s frame:

```
→ public RectangleInsets getPadding();
Returns the padding between the block’s content and its border. The default value is
RectangleInsets.ZERO_INSETS.
```

```
→ public void setPadding(RectangleInsets padding);
Sets the padding between the block’s content and its border. If padding is null, this method
throws an IllegalArgumentException.
```

#### 25.2.4 Layout

For layout purposes, a block can be asked to arrange itself subject to some constraint, and return the space required by the block:

```
→ public Size2D arrange(Graphics2D g2);
Arranges the block without constraint, and returns its size. Keep in mind that the block may
be a BlockContainer that contains other blocks.
```

```
→ public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);
Arranges the block subject to the specified constraint and returns its size. Keep in mind that
the block may be a BlockContainer that contains other blocks.
```

To control the current bounds for the block:

```
→ public Rectangle2D getBounds();
Returns the bounds for the block.
```

```
→ public void setBounds(Rectangle2D bounds);
Sets the bounds for the block. This method is often called by a layout manager.
```

The following utility methods are provided for subclasses to use:

```
→ protected double trimToContentWidth(double fixedWidth);
Reduces the given width to account for the margin, border and padding.
```

```
→ protected double trimToContentHeight(double fixedHeight);
Reduces the given height to account for the margin, border and padding.
```

```

→ protected RectangleConstraint toContentConstraint(RectangleConstraint c);
Translates a bounds constraint into a content constraint.

→ protected double calculateTotalWidth(double contentWidth);
Calculates the bounds width from the content width.

→ protected double calculateTotalHeight(double contentHeight);
Calculates the bounds height from the content height.

→ protected Rectangle2D trimMargin(Rectangle2D area);
Trims the block's margin from area.

→ protected Rectangle2D trimBorder(Rectangle2D area);
Trims the block's border from area.

→ protected Rectangle2D trimPadding(Rectangle2D area);
Trims the block's padding from area.

```

### 25.2.5 Drawing

This class is abstract, so it doesn't have a `draw()` method implemented. However, it does provide a method to draw the current border/frame:

```

→ protected void drawBorder(Graphics2D g2, Rectangle2D area);
Draws the border for the block.

```

### 25.2.6 Equals, Cloning and Serialization

To test a block for equality with an arbitrary object:

```

→ public boolean equals(Object obj);
Returns true if this block is equal to obj, and false otherwise.

```

Instances of this class are `Cloneable` and `Serializable`.

### 25.2.7 Notes

Some points to note:

- the `get/setBorder()` methods have been deprecated in favour of the `get/setFrame()` methods;

## 25.3 Arrangement

### 25.3.1 Overview

A layout manager that can arrange blocks.

### 25.3.2 Methods

This interface defines the following methods:

```

→ public void add(Block block, Object key);
Adds a block to the layout, with the specified key. The layout manager has an opportunity to
record the key associated with any block (or it can choose to ignore this information).

→ public void arrange(BlockContainer container, RectangleConstraint constraint,
Graphics2D g2);
Arranges the blocks within the given container, subject to the specified constraint.

→ public void clear();
Clears any cached layout information.

```

### 25.3.3 Notes

Some points to note:

- classes that implement this interface include:
  - `BorderArrangement`;
  - `CenterArrangement`;
  - `ColumnArrangement`;
  - `FlowArrangement`; and
  - `GridArrangement`.

## 25.4 Block

### 25.4.1 Overview

This interface defines methods that allow a rectangular graphical object (referred to generically as a “block”) to:

- identify itself;
- provide information about its size, perhaps subject to an external constraint;
- set its bounds.

Some blocks draw their own content, while other blocks act as containers for yet more blocks.

### 25.4.2 Methods

To access the block’s ID:

```
➔ public String getID();
Returns the ID for the block (depending on the application, this might be null).
```

```
➔ public void setID(String id);
Sets the id for the block.
```

To layout the contents of the block:

```
➔ public Size2D arrange(Graphics2D g2);
Arranges the block without any constraints and returns the block size.
```

```
➔ public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);
Arranges the block, subject to the given constraint, and returns the resulting size.
```

To access the current bounds for the block:

```
➔ public Rectangle2D getBounds();
Gets the bounds for the block.
```

```
➔ public void setBounds(Rectangle2D bounds);
Sets the bounds for the block.
```

## 25.5 BlockBorder

### 25.5.1 Overview

A simple border that can be assigned to any subclass of `AbstractBlock` (via that class’s `setFrame()` method). This class implements the `BlockFrame` interface.

### 25.5.2 Constructors

There are two constructors:

- `public BlockBorder();`  
Equivalent to `BlockBorder(Color.black)`—see below.
- `public BlockBorder(Paint paint);`  
Equivalent to `BlockBorder(new RectangleInsets(1, 1, 1, 1), paint)`—see below.
- `public BlockBorder(double top, double left, double bottom, double right);`  
Equivalent to `BlockBorder((new RectangleInsets(top, left, bottom, right), Color.black))`—see below.
- `public BlockBorder(double top, double left, double bottom, double right, Paint paint);`  
Equivalent to `BlockBorder((new RectangleInsets(top, left, bottom, right), paint))`—see below.
- `public BlockBorder(RectangleInsets insets, Paint paint);`  
Creates a new block border using the specified `insets` and `paint`. If `insets` or `paint` is `null`, this constructor throws an `IllegalArgumentException`.

### 25.5.3 General Attributes

The following read-only attributes are defined:

- `public RectangleInsets getInsets();`  
Returns the insets that define the available drawing space for the border (never `null`).
- `public Paint getPaint();`  
Returns the paint that is used to draw the border. The initial value is specified in the constructor for this class. This method never returns `null`.

### 25.5.4 Other Methods

JFreeChart calls the following method to draw the border:

- `public void draw(Graphics2D g2, Rectangle2D area);`  
Draws the border around the edges of the specified `area`, always staying within the area.

### 25.5.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this border for equality with an arbitrary object.

#### See Also

[BlockFrame](#).

## 25.6 BlockContainer

### 25.6.1 Overview

A container for blocks that uses an [Arrangement](#) to organise the layout of the blocks. The container is itself a [Block](#), which makes it possible to nest block containers to arbitrary levels.

## 25.6.2 Constructors

To create a new container:

- `public BlockContainer();`  
Creates a new container using a `BorderArrangement`.
- `public BlockContainer(Arrangement arrangement);`  
Creates a new container using the specified `arrangement`.

## 25.6.3 Methods

To get or set the layout manager:

- `public Arrangement getArrangement();`  
Returns the object responsible for the block layout.
- `public void setArrangement(Arrangement arrangement);`  
Sets the object responsible for the block layout.

To check if the container has an content:

- `public boolean isEmpty();`  
Returns `true` if the container is empty (contains no blocks), and `false` otherwise.

To get a list of the blocks within the container:

- `public List getBlocks();`  
Returns an unmodifiable list of the blocks in the container.

To add a block:

- `public void add(Block block);`  
Adds a block to the container.
- `public void add(Block block, Object key);`  
Adds a block to the container along with the given key (which is intended for the use of the layout manager).

To remove all blocks from the container:

- `public void clear();`  
Clears all the blocks in the container.

To arrange the blocks within the container (this will set the bounds for all the blocks):

- `public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);`  
Arranges the blocks in the container, subject to the specified constraint.

To draw the contents of the container:

- `public void draw(Graphics2D g2, Rectangle2D area);`  
Draws the blocks within the specified area.

## 25.6.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Returns `true` if this container is equal to `obj` and `false` otherwise.

This class is `Cloneable` and `Serializable`.

## 25.7 BlockFrame

### 25.7.1 Overview

An interface that defines the API for a border that can be assigned to any [AbstractBlock](#) (via that class's `setFrame()` method). This interface is implemented by:

- [BlockBorder](#);
- [LineBorder](#).

### 25.7.2 Interface Methods

This interface defines two methods:

- `public RectangleInsets getInsets(); [1.0.5]`  
Returns the space used to draw the frame.
- `public void draw(Graphics2D g2, Rectangle2D area); [1.0.5]`  
Draws the frame within the specified `area`.

### 25.7.3 Notes

This interface was introduced in JFreeChart version 1.0.5.

#### See Also

[AbstractBlock](#).

## 25.8 BlockParams

### 25.8.1 Overview

A carrier for the (optional) parameters passed to a [Block](#) in its `draw()` method.

### 25.8.2 Methods

To access the flag that controls whether or not entities are being generated:

- `public boolean getGenerateEntities();`  
Returns `true` if entities should be generated.
- `public void setGenerateEntities(boolean generate);`  
Sets the flag that controls whether or not entities are generated.

The translation from the local coordinates of the block to the container's coordinates:

- `public double getTranslateX();`  
Returns the x-translation.
- `public void setTranslateX(double x);`  
Sets the x-translation.
- `public double getTranslateY();`  
Returns the y-translation.
- `public void setTranslateY(double y);`  
Sets the y-translation.

## 25.9 BlockResult

### 25.9.1 Overview

A carrier for the result from the `draw()` method in the [BlockContainer](#) class.

### 25.9.2 Methods

```
► public EntityCollection getEntityCollection();
Returns the entity collection from the block drawing.
```

```
► public void setEntityCollection(EntityCollection entities);
Sets the entity collection.
```

## 25.10 BorderArrangement

### 25.10.1 Overview

A layout manager ([Arrangement](#)) that is similar to the `BorderLayout` class in AWT.

### 25.10.2 Constructor

To create a new instance:

```
► public BorderArrangement();
Creates a new layout manager.
```

### 25.10.3 Methods

The layout manager records the “key” for each block in the following method, which is usually called by the `BlockContainer`:

```
► public void add(Block block, Object key);
Records the block and its key (valid keys are defined by the RectangleEdge class).
```

```
► public Size2D arrange(BlockContainer container, RectangleConstraint constraint, Graphics2D g2);
Arranges the blocks within the container, subject to the given constraint, and returns the overall size of the container.
```

```
► public void clear();
Clears any cached layout information.
```

## 25.11 CenterArrangement

### 25.11.1 Overview

An [Arrangement](#) that places a single block at the center of its container.

## 25.12 ColorBlock

### 25.12.1 Overview

A simple block that is filled with a color. This is a useful class for visual testing of layout classes.

### 25.12.2 Constructor

To create a new block:

```
► public ColorBlock(Paint paint, double width, double height);
Creates a new block with the specified “preferred” dimensions. If paint is null, this method throws an IllegalArgumentException.
```

### 25.12.3 Methods

To get the color for the block:

```
➔ public Paint getPaint(); [1.0.5]
    Returns the paint specified in the constructor. This is never null.
```

To draw the block:

```
➔ public void draw(Graphics2D g2, Rectangle2D area);
    Draws the block inside the given area.
```

### 25.12.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
    Tests this ColorBlock for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

## 25.13 ColumnArrangement

### 25.13.1 Overview

An [Arrangement](#) that lays out the blocks in a container into columns. This is the “vertical” equivalent of the [FlowArrangement](#) class.

### 25.13.2 Constructors

```
➔ public ColumnArrangement();
    Creates a new arrangement.

➔ public ColumnArrangement(HorizontalAlignment hAlign, VerticalAlignment vAlign, double hGap,
    double vGap);
    Creates a new arrangement with the specified horizontal and vertical alignments and gaps.
```

### 25.13.3 Methods

To arrange the blocks within a container:

```
➔ public void arrange(BlockContainer container, RectangleConstraint constraint, Graphics2D
    g2);
    Arranges the blocks in container, subject to the given constraint.
```

To add a block to the layout:

```
➔ public void add(Block block, Object key);
    Adds a block to the layout. The key is ignored.
```

To clear the blocks:

```
➔ public void clear();
    Clears any cached information. In this case, the method does nothing.
```

### 25.13.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
    Tests this arrangement for equality with an arbitrary object.
```

This class is immutable, so it doesn’t need to be `Cloneable`.

## 25.14 EmptyBlock

### 25.14.1 Overview

An empty block, which can be useful for inserting fixed amounts of white space into a layout.

```
► public EmptyBlock(double width, double height);
Creates a new empty block with the specified “preferred” dimensions.
```

### 25.14.2 Methods

To draw the block:

```
► public void draw(Graphics2D g2, Rectangle2D area);
Draws the block (since the block is empty, this does nothing).
► public Object clone() throws CloneNotSupportedException;
Returns a clone of the block.
```

## 25.15 EntityBlockParams

### 25.15.1 Overview

To be documented.

## 25.16 EntityBlockResult

### 25.16.1 Overview

To be documented.

## 25.17 FlowArrangement

### 25.17.1 Overview

An [Arrangement](#) that lays out blocks horizontally from left to right (with wrapping if necessary).

### 25.17.2 Constructors

To create a new arrangement:

```
► public FlowArrangement();
Creates a new arrangement with default settings.
► public FlowArrangement(HorizontalAlignment hAlign, VerticalAlignment vAlign, double hGap,
double vGap);
Creates a new arrangement with the given alignment and gap settings.
```

### 25.17.3 Methods

To perform an arrangement on a container:

```
► public void arrange(BlockContainer container, RectangleConstraint constraint,
Graphics2D g2);
Arranges the blocks in the specified container according to the given constraint.
```

The following methods are also defined:

```
► public void add(Block block, Object key);
Adds a block to the arrangement. This method does nothing.
► public void clear();
Clears any cached information held by this instance.
```

#### 25.17.4 Equals, Cloning and Serialization

```
► public boolean equals(Object obj);
Tests this arrangement for equality with an arbitrary object.
```

### 25.18 GridArrangement

#### 25.18.1 Overview

A layout manager ([Arrangement](#)) that places blocks within a fixed size grid.

#### 25.18.2 Constructor

To create a new instance:

```
► public GridArrangement(int rows, int columns);
Creates a new instance with the specified number of rows and columns.
```

#### 25.18.3 Methods

```
► public void add(Block block, Object key);
Adds a block to the layout. This method does nothing, because the grid layout doesn't require any information about the blocks.

► public Size2D arrange(BlockContainer container, RectangleConstraint constraint,
Graphics2D g2);
Arranges the blocks in the specified container subject to the given constraint.
```

#### See Also

[FlowArrangement](#)

### 25.19 LabelBlock

#### 25.19.1 Overview

A label that can be incorporated into a block layout. For example, the series labels in a [LegendTitle](#) are displayed using instances of this class.

#### 25.19.2 Constructors

To create a new instance:

```
► public LabelBlock(String text);
Creates a new label block with the given (non-null) text and a default font (Sans Serif, PLAIN, 10) and color (black).

► public LabelBlock(String text, Font font);
Creates a new label block with the specified text, font and a default color (black). Both text and font should be non-null.

► public LabelBlock(String text, Font font, Paint paint);
Creates a new label block with the specified text, font and paint (all of which must be non-null).
```

### 25.19.3 Attributes

To get/set the font used for the label:

```
► public Font getFont();
Returns the font used for the label (never null).

► public void setFont(Font font);
Sets the font for the label (null is not permitted).
```

To get/set the paint used for the label text:

```
► public Paint getPaint();
Returns the paint used for the label text (never null). The default value is Color.BLACK.

► public void setPaint(Paint paint);
Sets the paint for the label text (null is not permitted).
```

To get/set the tooltip text for the label (if any):

```
► public String getToolTipText();
Returns the tooltip text (possibly null).

► public void setToolTipText(String text);
Sets the tooltip text (null permitted).
```

To get/set the URL text for the label (if any):

```
► public String getURLText();
Returns the URL text for the label block. This may be null.

► public void setURLText(String text);
Sets the tooltip text (null permitted).
```

### 25.19.4 Other Methods

The following methods are used by the layout and drawing mechanism in JFreeChart. You won't normally call them yourself.

```
► public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);
Fits the label block to the specified constraints, and returns the dimensions.

► public void draw(Graphics2D g2, Rectangle2D area);
Draws the label within the specified area.

► public Object draw(Graphics2D g2, Rectangle2D area, Object params);
Draws the label within the specified area.
```

### 25.19.5 Equals, Cloning and Serialization

To test an instance for equality with an arbitrary object:

```
► public boolean equals(Object obj);
Tests this instance for equality with obj. Returns true if and only if:

• obj is not null;
• obj is an instance of LabelBlock;
• each field in this instance is the same as the corresponding field in obj.
```

Instances of this class are `Cloneable` and `Serializable`.

### 25.19.6 Notes

Some points to note:

- this class implements the `Block` interface, and thus supports margins, borders and padding as do all blocks.

## 25.20 LengthConstraintType

### 25.20.1 Overview

This class defines three constraint types:

- `LengthConstraintType.NONE`;
- `LengthConstraintType.FIXED`;
- `LengthConstraintType.RANGE`;

These types are used when creating `RectangleConstraint` instances.

### 25.20.2 Methods

The following methods are implemented:

- `public String toString();`  
Returns a string representation of the instance, primarily used for debugging.
- `public boolean equals(Object obj);`  
Tests this instance for equality with an arbitrary object.
- `public int hashCode();`  
Returns a hash code for the instance.

## 25.21 LineBorder

### 25.21.1 Overview

A simple border that can be assigned to any subclass of `AbstractBlock` (via that class's `setFrame()` method). This class implements the `BlockFrame` interface.

### 25.21.2 Constructors

There are two constructors:

- `public LineBorder();`  
Equivalent to `LineBorder(Color.black, new BasicStroke(1.0f), new RectangleInsets(1.0, 1.0, 1.0, 1.0))`—see the next constructor.
- `public LineBorder(Paint paint, Stroke stroke, RectangleInsets insets);`  
Creates a new line border using the specified `paint`, `stroke` and `insets`. The `insets` determines how much space is reserved for the border (but note that the border is drawn regardless of the size of the insets). If any of the arguments is `null`, this constructor throws an `IllegalArgumentException`.

### 25.21.3 General Attributes

The following read-only attributes are defined:

- `public Paint getPaint(); [1.0.5]`  
Returns the paint that is used to draw the border. The initial value is specified in the constructor for this class. This method never returns `null`.
- `public Stroke getStroke(); [1.0.5]`  
Returns the stroke that is used to draw the border. The initial value is specified in the constructor for this class. This method never returns `null`.
- `public RectangleInsets getInsets(); [1.0.5]`  
Returns the insets that define the drawing space reserved for the border (never `null`).

### 25.21.4 Other Methods

JFreeChart calls the following method to draw the border:

► `public void draw(Graphics2D g2, Rectangle2D area);`

Draws the border around the edges of the specified `area`, always staying within the area.

### 25.21.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`

Tests this border for equality with an arbitrary object.

Instances of this class are immutable (so this class doesn't implement `Cloneable`) and `Serializable`.

#### See Also

[BlockFrame](#).

## 25.22 RectangleConstraint

### 25.22.1 Overview

A specification of the constraints that a rectangular shape must meet. For each dimension (width and height) there are three possible constraints: `NONE`, `FIXED` and `RANGE`—refer to the constant class `LengthConstraintType`. These constraints are used by the layout code implemented by JFreeChart.

### 25.22.2 Constructors

There are several constructors:

► `public RectangleConstraint(double w, double h);`

Creates a new constraint where both the width and height are fixed at the given dimensions.

► `public RectangleConstraint(Range w, Range h);`

Creates a new constraint where the width and height must fall within the given ranges.

► `public RectangleConstraint(double w, Range widthRange,`

`LengthConstraintType widthConstraintType, double h, Range heightRange,`

`LengthConstraintType heightConstraintType);`

Creates a new constraint with the specified attributes (this method gives you full control over all attributes). Note that the width and height ranges may be specified as `null`.

### 25.22.3 Accessor Methods

To access the attributes of this class:

► `public double getWidth();`  
Returns the fixed width.

► `public Range getWidthRange();`  
Returns the width range (possibly `null`).

► `public LengthConstraintType getWidthConstraintType();`  
Returns the width constraint type (never `null`).

► `public double getHeight();`  
Returns the fixed height.

► `public Range getHeightRange();`  
Returns the height range (possibly `null`).

► `public LengthConstraintType getHeightConstraintType();`  
Returns the height constraint type (never `null`).

#### 25.22.4 Other Methods

Other methods include:

- ↳ `public RectangleConstraint toUnconstrainedWidth();`  
Returns a new instance with the same height constraint and NO width constraint.
- ↳ `public RectangleConstraint toUnconstrainedHeight();`  
Returns a new instance with the same width constraint and NO height constraint.
- ↳ `public RectangleConstraint toFixedWidth(double width);`  
Returns a new instance with the same height constraint and a FIXED width constraint.
- ↳ `public RectangleConstraint toFixedHeight(double height);`  
Returns a new instance with the same width constraint and a FIXED height constraint.
- ↳ `public Size2D calculateConstrainedSize(Size2D base);`  
Applies the constraint to the supplied dimensions and returns the “constrained” dimensions.
- ↳ `public String toString();`  
Returns a string representing this class, primarily for debugging purposes.

# Chapter 26

## Package: org.jfree.chart.editor

### 26.1 Introduction

This package contains a framework for editing chart properties. At present, the implementation is incomplete. The API is minimalistic, in the hope that it will be possible to plug in a more complete implementation later on without requiring major changes to the API.

### 26.2 ChartEditor

#### 26.2.1 Overview

An interface that defines the API that needs to be supported by a chart editor. A chart editor should be a subclass of `JComponent`.

#### 26.2.2 Methods

This interface defines a single method:

```
→ public void updateChart(JFreeChart chart);  
Applies the updates that the user has made via the chart editor to the given chart.
```

#### 26.2.3 Notes

To obtain a chart editor, use the `getChartEditor()` method in the `ChartEditorManager` class.

### 26.3 ChartEditorFactory

#### 26.3.1 Overview

An interface that defines the API that needs to be supported by a chart editor factory, a class that creates new instances of `ChartEditor`. The `ChartEditorManager` class maintains a factory for creating new editors—you can replace the default factory with your own custom factory if you want to install your own chart editor.

#### 26.3.2 Methods

This interface defines a single method:

```
→ public ChartEditor createEditor(JFreeChart chart);  
Creates a new editor for the given chart.
```

### 26.3.3 Notes

The `DefaultChartEditorFactory` class provides the default implementation of this interface.

## 26.4 ChartEditorManager

### 26.4.1 Overview

This class is the central source for new `ChartEditor` instances. You can use the default chart editor (which is incomplete at this time) or install your own `ChartEditorFactory` class to return your own custom chart editor.

### 26.4.2 Methods

This class defines several static methods:

- ▶ `public static ChartEditorFactory getChartEditorFactory();`  
Returns the current chart editor factory.
- ▶ `public static void setChartEditorFactory(ChartEditorFactory f);`  
Sets the chart editor factory. This allows you to install a custom chart editor implementation, since the `getChartEditor()` method will return an editor created by the installed factory.
- ▶ `public static ChartEditor getChartEditor(JFreeChart chart);`  
Returns a chart editor for the given chart. The editor is created by the installed chart editor factory (which you can change via the `setChartEditorFactory()` method).

### 26.4.3 Notes

This package contains default implementations of `ChartEditorFactory` and `ChartEditor`. These classes are not publicly visible and are subject to change.

## 26.5 DefaultAxisEditor

### 26.5.1 Overview

A panel for editing the properties of an axis.

The code for this panel is out of date. Many features are missing, and some of the existing features may not work. It is planned to rewrite this class.

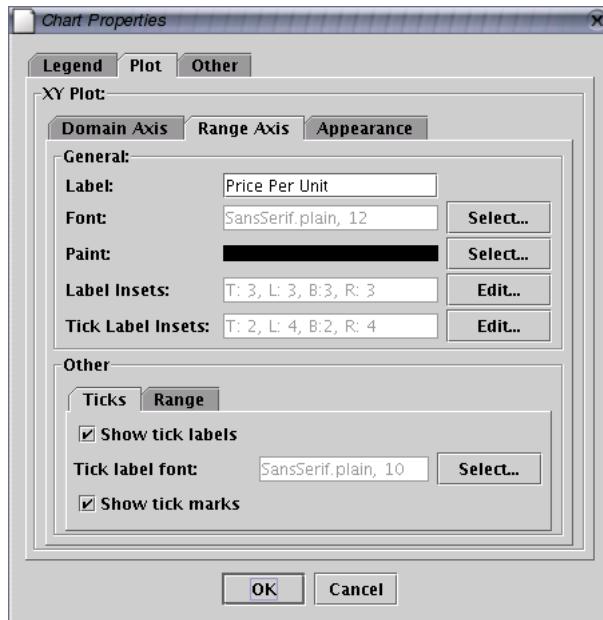
## 26.6 DefaultChartEditor

### 26.6.1 Overview

A panel that displays all the properties of a chart, and allows the user to edit the properties. The panel uses a `JTabbedPane` to display three sub-panels:

- a `DefaultTitleEditor`;
- a `DefaultPlotEditor`;
- a panel containing “other” properties (such as the anti-alias setting and the background paint for the chart).

The constructors for this class require a reference to a `Dialog` or a `Frame`. Whichever one is specified is passed on to the `DefaultTitleEditor` and is used if and when a sub-dialog is required for editing titles.



### 26.6.2 Notes

This class is not publicly visible and its API is subject to change.

## 26.7 DefaultChartEditorFactory

### 26.7.1 Overview

A default factory used by the [ChartEditorManager](#) class.

### 26.7.2 Constructors

To create a new instance:

```
➔ public DefaultChartEditorFactory();
Creates a new factory instance.
```

### 26.7.3 Methods

To create a new chart editor:

```
➔ public ChartEditor createEditor(JFreeChart chart);
Creates a new editor for the given chart.
```

### 26.7.4 Notes

The [ChartEditorManager](#) class installs an instance of this class as the default chart editor factory.

## 26.8 DefaultColorBarEditor

### 26.8.1 Overview

A panel for editing the properties of a [ColorBar](#). *This class is deprecated as of version 1.0.4.*

## 26.9 DefaultNumberAxisEditor

### 26.9.1 Overview

A panel for displaying and editing the properties of a [NumberAxis](#).

## 26.10 DefaultPlotEditor

### 26.10.1 Overview

A panel for displaying and editing the properties of a plot.

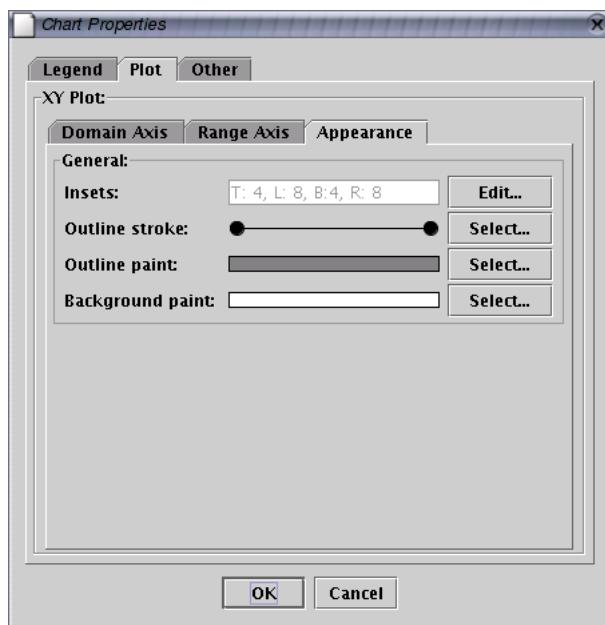


Figure 26.1: The plot property editor

The code for this panel is out of date. Many features are missing, and some of the existing features may not work. It is planned to rewrite this class.

## 26.11 DefaultTitleEditor

### 26.11.1 Overview

A panel for displaying and editing the properties of a chart title. The code for this panel is out of date. Many features are missing, and some of the existing features may not work. It is planned to rewrite this class.

## 26.12 PaletteChooserPanel

### 26.12.1 Overview

A panel for selecting a color palette. *This class is deprecated as of version 1.0.4.*

## 26.13 PaletteSample

### 26.13.1 Overview

*This class is deprecated as of version 1.0.4.*

# Chapter 27

## Package: org.jfree.chart.encoders

### 27.1 Introduction

The `org.jfree.chart.encoders` package provides a mechanism to allow encoders from Java's Image IO framework to be used where they are available (JDK 1.4 onwards) while ensuring that alternative encoders are provided as a fallback in other cases (that is, on JDK 1.3).

This mechanism is employed by several methods in the `ChartUtilities` class, for example `writeChartAsPNG()` and `writeChartAsJPEG()`.

### 27.2 EncoderUtil

#### 27.2.1 Overview

A utility class containing static methods for encoding images in several formats (PNG and JPEG are supported in most cases). The methods in this class are called by the `ChartUtilities` class, and make use of encoders pre-configured in the `ImageEncoderFactory` class.

#### 27.2.2 Methods

To encode an image:

```
➔ public static byte[] encode(BufferedImage image, String format);  
    Returns a byte array containing an encoded version of the image in the specified format.  
  
➔ public static byte[] encode(BufferedImage image, String format,  
    boolean encodeAlpha) throws IOException;  
    Returns a byte array containing an encoded version of the image in the specified format. The  
    encodeAlpha flag determines whether or not an alpha channel is included in the encoded image  
    (assuming the format supports this).  
  
➔ public static byte[] encode(BufferedImage image, String format,  
    float quality) throws IOException;  
    Returns a byte array containing an encoded version of the image in the specified format. The  
    quality argument controls the image quality (for encoders that support this).  
  
➔ public static byte[] encode(BufferedImage image, String format,  
    float quality, boolean encodeAlpha);  
    Returns a byte array containing an encoded version of the image in the specified format. The  
    encodeAlpha flag determines whether or not an alpha channel is included in the encoded image  
    (assuming the format supports this). The quality argument controls the image quality (for  
    encoders that support this).
```

To write an image to an output stream:

```

→ public static void writeBufferedImage(BufferedImage image, String format,
OutputStream outputStream) throws IOException;
Writes an image to the given output stream in the specified format.

→ public static void writeBufferedImage(BufferedImage image, String format,
OutputStream outputStream, float quality) throws IOException;
Writes an image to the given output stream in the specified format.

→ public static void writeBufferedImage(BufferedImage image, String format,
OutputStream outputStream, boolean encodeAlpha) throws IOException;
Writes an image to the given output stream in the specified format.

→ public static void writeBufferedImage(BufferedImage image, String format,
OutputStream outputStream, float quality, boolean encodeAlpha) throws IOException;
Writes an image to the given output stream in the specified format.

```

#### See Also

[ChartUtilities](#).

## 27.3 ImageEncoderFactory

### 27.3.1 Overview

A factory class for image encoders. The static initialisation code in this class checks if we are running on JRE 1.4.2 or later. If yes, then the following encoders are pre-configured:

- “png”: `org.jfree.chart.encoders.SunPNGEncoderAdapter`;
- “jpeg”: `org.jfree.chart.encoders.SunJPEGEncoderAdapter`;

Otherwise, JRE 1.3.1 must be the runtime. In that case, Java’s ImageIO library is not available and we register the KeyPoint PNG encoder:

- “png”: `org.jfree.chart.encoders.KeypointPNGEncoderAdapter`;

### 27.3.2 Methods

The following method is used to register an encoder with the factory (note that JFreeChart pre-installs PNG and JPEG encoders):

```

→ public static void setImageEncoder(String format, String imageEncoderClassName);
Adds a mapping between a format name (for example, “png”) and an encoder class name (for
example, “org.jfree.chart.encoders.SunPNGEncoderAdapter”).

```

Use the following methods to obtain an encoder:

```

→ public static ImageEncoder newInstance(String format);
Returns a new instance of an encoder for the given format.

→ public static ImageEncoder newInstance(String format, float quality);
Returns a new instance of the encoder for the given format and the specified quality setting
(in the range 0.0f (lowest quality) to 1.0f (highest quality)).

→ public static ImageEncoder newInstance(String format, boolean encodingAlpha);
Returns a new instance of the encoder for the given format and the specified encodeAlpha flag.

→ public static ImageEncoder newInstance(String format, float quality, boolean encodingAlpha);
Returns a new instance of the encoder for the given format and the specified quality and
encodeAlpha flag settings.

```

#### See Also

[EncoderUtil](#).

## 27.4 ImageEncoder

### 27.4.1 Overview

An interface that provides an abstract view of the image encoders supported by this package. Classes that implement this interface include:

- [KeypointPNGEncoderAdapter](#);
- [SunJPEGEncoderAdapter](#);
- [SunPNGEncoderAdapter](#).

### 27.4.2 Methods

To encode an image:

- ▶ `public byte[] encode(BufferedImage bufferedImage) throws IOException;`  
Returns a byte array containing an encoded version of the given image.
- ▶ `public void encode(BufferedImage bufferedImage, OutputStream outputStream) throws IOException;`  
Writes an encoded version of an image to the given output stream.

To control the quality for the encoding (typically there is a trade-off between image size and quality):

- ▶ `public float getQuality();`  
Returns the image quality setting.
- ▶ `public void setQuality(float quality);`  
Sets the quality. Note that some encoders ignore the quality setting.

To control whether or not the encoding should support an alpha-transparency channel:

- ▶ `public boolean isEncodingAlpha();`  
Returns the flag that controls whether or not the alpha channel is encoded with the image (note that some encoders ignore this setting).
- ▶ `public void setEncodingAlpha(boolean encodingAlpha);`  
Sets the flag that controls whether or not the alpha channel is encoded with the image.

#### See Also

[ImageEncoderFactory](#).

## 27.5 ImageFormat

### 27.5.1 Overview

An interface that defines string constants used to identify several common image formats:

- `ImageFormat.PNG` for the PNG format,
- `ImageFormat.JPG` for the JPEG format,
- `ImageFormat.GIF` for the GIF format.

You can use these constants in the methods provided by the [EncoderUtil](#) class.

## 27.6 KeyPointPNGEncoderAdapter

### 27.6.1 Overview

An adapter for the `com.keypoint.PNGEncoder` included in the JCommon distribution. This adapter will be used when JFreeChart is compiled or run with JDK/JRE 1.3.1 (in this case, the ImageIO framework is not available).

## 27.6.2 Methods

To set the image quality:

► `public float getQuality();`

Returns the quality setting for the encoder. The default value is 9.

► `public void setQuality(float quality);`

Sets the quality setting for the encoder. Since PNG is a “lossless” format, the image is always encoded without loss of quality. This setting in fact controls the amount of compression achieved. The underlying encoder uses integer codes as follows:

- 0 – no compression,

- 1 – best speed,

- 9 – best compression.

Note that any value between 1 and 9 is also permitted.

To set the flag that controls whether or not the alpha channel is encoded:

► `public boolean isEncodingAlpha();`

Returns the flag that controls whether or not the alpha channel is included in the encoded image.

► `public void setEncodingAlpha(boolean encodingAlpha);`

Sets the flag that controls whether or not the alpha channel is included in the encoded image.

To encode an image to a byte array:

► `public byte[] encode(BufferedImage bufferedImage) throws IOException;`

Returns a byte array containing an encoded version of the given image. The encoding uses the current `quality` and `encodeAlpha` settings.

To write an encoded version of an image to an output stream:

► `public void encode(BufferedImage bufferedImage, OutputStream outputStream) throws IOException;`

Writes a byte array (containing an encoded version of the given image) to the specified output stream. The encoding uses the current `quality` and `encodeAlpha` settings. Note that the entire image is encoded to a byte array first, before writing the bytes to the output stream—for large images this can use a lot of memory.

### See Also

[SunPNGEncoderAdapter](#).

## 27.7 SunJPEGEncoderAdapter

### 27.7.1 Overview

An encoder for the JPEG image file format that uses Java’s ImageIO framework to perform the encoding. This encoder is only available when JFreeChart is compiled and run using JDK/JRE 1.4.2 or later. The Ant build script excludes it from the build when using JDK 1.3.1, in which case the methods that write charts to JPEG format will throw exceptions. Since JPEG is such a rotten format for charts, this is no great loss.

### 27.7.2 Methods

The quality setting is ignored by this encoder:

► `public float getQuality();`

Returns the quality setting.

► `public void setQuality(float quality);`

Sets the quality setting.

The alpha encoding flag is ignored by this encoder:

```
► public boolean isEncodingAlpha();  
Returns false always.  
  
► public void setEncodingAlpha(boolean encodingAlpha);  
Any value passed to this method is ignored.
```

To encode an image:

```
► public byte[] encode(BufferedImage bufferedImage) throws IOException;  
Returns a byte array containing a version of the given image encoded in JPEG format by Java's  
ImageIO framework.  
  
► public void encode(BufferedImage bufferedImage, OutputStream outputStream) throws IOException;  
Writes an image to the given output stream (in JPEG format) using Java's ImageIO framework.
```

## 27.8 SunPNGEncoderAdapter

### 27.8.1 Overview

An encoder for the PNG image file format that uses Java's ImageIO framework to perform the encoding. This encoder is only available when JFreeChart is compiled and run using JDK/JRE 1.4.2 or later. The Ant build script excludes it from the build when using JDK 1.3.1, in which case the [KeypointPNGEncoderAdapter](#) is used instead.

### 27.8.2 Methods

The quality setting is ignored by this encoder:

```
► public float getQuality();  
Returns 0.0f always, the encoder does not support the quality setting.  
  
► public void setQuality(float quality);  
Any value passed to this method is ignored, the encoder does not support the quality setting.
```

The alpha encoding flag is ignored by this encoder:

```
► public boolean isEncodingAlpha();  
Returns false always.  
  
► public void setEncodingAlpha(boolean encodingAlpha);  
Any value passed to this method is ignored.
```

To encode an image:

```
► public byte[] encode(BufferedImage bufferedImage) throws IOException;  
Returns a byte array containing a version of the given image encoded in PNG format by Java's  
ImageIO framework.  
  
► public void encode(BufferedImage bufferedImage, OutputStream outputStream) throws IOException;  
Writes an image to the given output stream (in PNG format) using Java's ImageIO framework.
```

### See Also

[KeypointPNGEncoderAdapter](#).

# Chapter 28

## Package: org.jfree.chart.entity

### 28.1 Introduction

The `org.jfree.chart.entity` package contains classes that represent entities in a chart. Entities provide information about the physical location of items in a chart that has been drawn, as well as optional data such as tool tip text and URL strings.

### 28.2 Background

Recall that when you render a chart to a `Graphics2D` using the `draw()` method in the `JFreeChart` class, you have the option of supplying a `ChartRenderingInfo` object to collect information about the chart's dimensions. Most of this information is represented in the form of `ChartEntity` objects, stored in an `EntityCollection`.

You can use the entity information in any way you choose. For example, the `ChartPanel` class makes use of the information for:

- displaying tool tips;
- handling chart mouse events.

It is more than likely that other applications for this information will be found.

### 28.3 CategoryItemEntity

#### 28.3.1 Overview

This class is used to convey information about an item within a category plot. The information captured includes the dataset containing the item, the series and category identifying the item, area occupied by the item (at the time the chart was rendered), and the tool tip and URL text (if any) generated for the item.

#### 28.3.2 Constructors

To construct a new instance:

```
➔ public CategoryItemEntity(Shape area, String toolTipText, String urlText,
CategoryDataset dataset, Comparable rowKey, Comparable columnKey); [1.0.6]
Creates a new entity instance.
```

The original constructor has been deprecated as of JFreeChart 1.0.6, and will be removed in a future release.

### 28.3.3 Methods

In addition to the methods inherited from the [ChartEntity](#) class:

- **public CategoryDataset getDataset();**  
Returns a reference to the dataset (never `null`).
- **public void setDataset(CategoryDataset dataset);**  
Sets the dataset reference for this entity (`null` is not permitted).

To identify the data item:

- **public Comparable getRowKey(); [1.0.6]**  
Returns the row key for the data item.
- **public void setRowKey(Comparable rowKey); [1.0.6]**  
Sets the row key for the data item.
- **public Comparable getColumnKey(); [1.0.6]**  
Returns the column key for the data item.
- **public void setColumnKey(Comparable columnKey); [1.0.6]**  
Sets the column key for the data item.

The original index-based methods for identifying the data item have been deprecated:

- **public int getSeries(); [Deprecated, 1.0.6]**  
Returns the index of the series containing the item that this entity represents.
- **public void setSeries(int series); [Deprecated, 1.0.6]**  
Sets the index of the series containing the item that this entity represents.
- **public Object getCategory(); [Deprecated, 1.0.6]**  
Returns the category containing the item that this entity represents. This may be `null`.
- **public void setCategory(Object category); [Deprecated, 1.0.6]**  
Sets the category containing the item that this entity represents.
- **public int getCategoryIndex(); [Deprecated, 1.0.6]**  
Returns the index of the category containing the item that this entity represents.
- **public void setCategoryIndex(int index); [Deprecated, 1.0.6]**  
Sets the index of the category containing the item that this entity represents.

For debugging purposes, this class overrides the `toString()` method:

- **public String toString();**  
Returns a string representation of this `CategoryItemEntity`, typically used when debugging.

### 28.3.4 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

- **public boolean equals(Object obj);**  
Tests this entity for equality with an arbitrary object.

Instances of this class are cloneable ([PublicCloneable](#) is implemented) and serializable.

### 28.3.5 Notes

Most `CategoryItemRenderer` implementations will generate entities using this class, as required.

#### See Also

[ChartEntity](#), [CategoryPlot](#).

## 28.4 ChartEntity

### 28.4.1 Overview

This class is used to convey information about an entity within a chart. The information captured includes the area occupied by the item and the tool tip text generated for the item.

There are a number of subclasses that can be used to provide additional information about a chart entity.

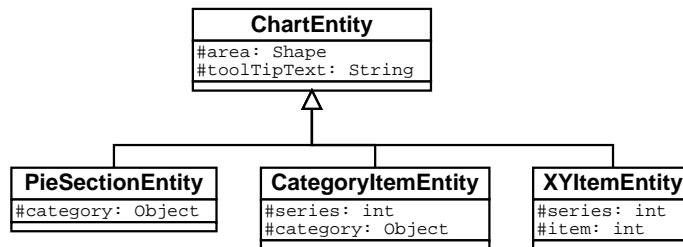


Figure 28.1: Chart entity classes

### 28.4.2 Constructors

To construct a new instance:

```
► public ChartEntity(Shape area, String toolTipText);
Creates a new chart entity object. The area is specified in Java 2D space.
```

Chart entities are created by other classes in the JFreeChart library, you don't usually need to create them yourself.

### 28.4.3 Methods

Accessor methods are implemented for the `area` and `toolTipText` attributes.

To support the generation of HTML image maps, the `getShapeType()` method returns a `String` containing either `RECT` or `POLY`, and the `getShapeCoords()` method returns a `String` containing the coordinates of the shape's outline. See the [ChartUtilities](#) class for more information about HTML image maps.

### 28.4.4 Notes

The `ChartEntity` class *records* where an entity has been drawn using a `Graphics2D` instance. Changing the attributes of an entity won't change what has already been drawn.

#### See Also

[CategoryItemEntity](#), [PieSectionEntity](#), [XYItemEntity](#).

## 28.5 ContourEntity

### 28.5.1 Overview

This class is deprecated as of JFreeChart version 1.0.4.

#### See Also

[ContourPlot](#).

## 28.6 EntityCollection

### 28.6.1 Overview

An interface that defines the API for a collection of *chart entities*. This is used by the `ChartRenderingInfo` class to record where items have been drawn when a chart is rendered using a `Graphics2D` instance.

Each `ChartEntity` can also record tool tip information (for displaying tool tips in a Swing user interface) and/or URL information (for generating HTML image maps).

### 28.6.2 Methods

The interface defines three methods. To clear a collection:

```
► public void clear();  
Clears the collection. All entities in the collection are discarded.
```

To add an entity to a collection:

```
► public void addEntity(ChartEntity entity);  
Adds an entity to the collection.
```

To retrieve an entity based on Java 2D coordinates:

```
► public ChartEntity getEntity(double x, double y);  
Returns an entity whose area contains the specified coordinates. If the coordinates fall within  
the area of multiple entities (the entities overlap) then only one entity is returned.
```

### 28.6.3 Notes

The `StandardEntityCollection` class provides a basic implementation of this interface (but one that won't scale to large numbers of entities).

#### See Also

[ChartEntity](#), [StandardEntityCollection](#).

## 28.7 LegendItemEntity

### 28.7.1 Overview

An entity that records information about a legend item.

### 28.7.2 Constructor

To create a new `LegendItemEntity`:

```
► public LegendItemEntity(Shape area);  
Creates a new legend item with the specified hotspot.
```

### 28.7.3 Methods

The legend item is associated with a dataset and a series key:

```
► public Dataset getDataset(); [1.0.6]  
Returns the dataset for the legend item.  
  
► public void setDataset(Dataset dataset); [1.0.6]  
Sets the dataset for the legend item.  
  
► public Comparable getSeriesKey(); [1.0.6]  
Returns the series key for the legend item.
```

```
► public void setSeriesKey(Comparable key); [1.0.6]
Sets the series key for the legend item.
```

To access the index of the series that the legend item relates to:

```
► public int getSeriesIndex(); [Deprecated, 1.0.6]
Returns the index of the series that the legend item entity relates to.

► public void setSeriesIndex(int index); [Deprecated, 1.0.6]
Sets the index of the series that the legend item entity relates to.
```

## 28.7.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this entity for equality with an arbitrary object.
```

To obtain a clone of the entity:

```
► public Object clone() throws CloneNotSupportedException;
Returns a clone of the entity.
```

## 28.8 PieSectionEntity

### 28.8.1 Overview

This class is used to convey information about an item within a pie plot. The information captured includes the area occupied by the item, the dataset, pie and section indices, and the tool tip and URL text (if any) generated for the item.

### 28.8.2 Constructors

To construct a new instance:

```
► public PieSectionEntity(Shape area, PieDataset dataset, int pieIndex, int sectionIndex,
Comparable sectionKey, String toolTipText, String urlText);
Creates a new entity object.
```

### 28.8.3 Methods

Accessor methods are implemented for the `dataset`, `pieIndex`, `sectionIndex` and `sectionKey` attributes. Other methods are inherited from the `ChartEntity` class.

### 28.8.4 Notes

The `PiePlot` class generates pie section entities as required.

#### See Also

[ChartEntity](#), [PiePlot](#).

## 28.9 StandardEntityCollection

### 28.9.1 Overview

A basic implementation of the `EntityCollection` interface. This class can be used (optionally, by the `ChartRenderingInfo` class) to store a collection of chart entity objects from one rendering of a chart. The entities can be used to support tool-tips, drill-down charts and HTML image maps.

### 28.9.2 Constructor

To create a new collection:

```
► public StandardEntityCollection();
Creates a new (empty) collection.
```

### 28.9.3 Methods

The following methods are supported by this class:

```
► public void add(ChartEntity entity);
Adds a chart entity to the collection. This method throws an IllegalArgumentException if
entity is null.

► public void addAll(EntityCollection collection);
Adds all the entities from the specified collection to this collection. This method throws a
NullPointerException if collection is null.

► public int getEntityCount();
Returns the number of entities stored in the collection.

► public ChartEntity getEntity(int index);
Returns a chart entity from the specified position in the collection.

► public void clear();
Clears all the entities from the collection.

► public ChartEntity getEntity(double x, double y);
Returns an entity from the collection that has a “hot-spot” that contains the specified (x, y)
location (in Java2D space).

► public Collection getEntities();
Returns an unmodifiable collection containing the entities.

► public Iterator iterator();
Returns an iterator for the entities in the collection.
```

### 28.9.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this entity collection for equality with an arbitrary object.

► public Object clone() throws CloneNotSupportedException
Returns a deep clone of this collection (each entity in the collection is cloned).
```

### 28.9.5 Notes

The `getEntity()` method iterates through the entities searching for one that contains the specified coordinates. For charts with a large number of entities, a more efficient approach will be required.<sup>1</sup>

#### See Also

[ChartEntity](#), [EntityCollection](#).

## 28.10 TickLabelEntity

### 28.10.1 Overview

An entity that records information about a tick label.

---

<sup>1</sup>This is on the to-do list but, given the size of the to-do list, I'm hopeful that someone will contribute code to address this.

## 28.11 XYAnnotationEntity

### 28.11.1 Overview

An entity that represents an [XYAnnotation](#).

### 28.11.2 Constructor

To create a new instance:

```
► public XYAnnotationEntity(Shape hotspot, int rendererIndex, String toolTipText,  
String urlText);
```

Creates a new entity with the specified `hotspot`. The renderer index denotes the renderer to which the annotation belongs.

### 28.11.3 Methods

In addition to the methods inherited from [ChartEntity](#), this class defines the following:

```
► public int getRendererIndex();
```

Returns the index of the renderer to which the annotation is assigned.

```
► public void setRendererIndex(int index);
```

Sets the index of the renderer to which the annotation is assigned.

### 28.11.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
```

Tests this entity for equality with an arbitrary object.

## 28.12 XYItemEntity

### 28.12.1 Overview

This class is used to convey information about an item within an XY plot. The information captured includes the area occupied by the item, the tool tip text generated for the item, and the series and item index.

### 28.12.2 Constructors

To construct a new instance:

```
► public XYItemEntity(Shape area, XYDataset dataset, int series, int item, String toolTipText,  
String urlText);
```

Creates a new entity object.

### 28.12.3 Methods

Accessor methods are implemented for the `dataset`, `series` and `item` attributes. Other methods are inherited from the [ChartEntity](#) class.

### 28.12.4 Notes

Most [XYItemRenderer](#) implementations will generate entities using this class, as required.

#### See Also

[ChartEntity](#), [XYPlot](#).

# Chapter 29

## Package: org.jfree.chart.event

### 29.1 Introduction

This package contains classes and interfaces that are used to broadcast and receive events relating to changes in chart properties. By default, some of the classes in the library will automatically register themselves with other classes, so that they receive notification of any changes and can react accordingly. For the most part, you can simply rely on this default behaviour.

### 29.2 AxisChangeEvent

#### 29.2.1 Overview

An event that can be sent to an `AxisChangeListener` to provide information about a change to an axis. Almost every axis update will trigger an `AxisChangeEvent`.

#### 29.2.2 Notes

Some points to note:

- often, the only information provided by the event is that *some* change has been made to the axis (that is, the specific change is not identified);
- when a chart is displayed in a `ChartPanel`, any `AxisChangeEvent` will trigger a chain of events that results in the chart on the panel being repainted.

### 29.3 AxisChangeListener

#### 29.3.1 Overview

An interface through which axis change event notifications are posted.

#### 29.3.2 Methods

The interface defines a single method:

```
↳ public void axisChanged(AxisChangeEvent event);
```

Receives notification of a change to an axis.

### 29.3.3 Notes

Some points to note:

- if a class needs to receive notification of changes to an axis, then it needs to implement this interface and register itself with the axis;
- the plot classes that manage axes (for example, `CategoryPlot` and `XYPlot`) implement this interface to listen for changes to the axes, and typically respond by generating a `PlotChangeEvent`.

## 29.4 ChartChangeEvent

### 29.4.1 Overview

An event that is used to provide information about changes to a chart. You can register an object with a `JFreeChart` instance, provided that the object implements the `ChartChangeListener` interface, and it will receive a notification whenever the chart changes.

### 29.4.2 Constructors

The following constructors are defined:

- ▶ `public ChartChangeEvent(Object source);`  
Creates a new event generated by the given `source`.
- ▶ `public ChartChangeEvent(Object source, JFreeChart chart);`  
Creates a new event generated by the given `source` for the given `chart` (the `source` and `chart` may be the same).
- ▶ `public ChartChangeEvent(Object source, JFreeChart chart, ChartChangeEventType type);`  
Creates a new event with the specified `type`.

### 29.4.3 Methods

The following methods are defined:

- ▶ `public JFreeChart getChart();`  
Returns the chart that the event relates to.
- ▶ `public void setChart(JFreeChart chart);`  
Sets the chart for the event.
- ▶ `public ChartChangeEventType getType();`  
Returns the event type.
- ▶ `public void setType(ChartChangeEventType type);`  
Sets the event type.

### 29.4.4 Notes

The `ChartPanel` class automatically registers itself with the chart it is displaying. When it receives a `ChartChangeEvent`, it repaints the chart.

## 29.5 ChartChangeEvent Type

### 29.5.1 Overview

This class defines the tokens that can be used to specify the “type” for a `ChartChangeEvent`.

The intent behind specifying event types is to allow JFreeChart to react in special ways to particular events. For example, an updated dataset may not require a chart redraw if the data that changed is outside the visible data range. However, there is currently no code in JFreeChart that takes advantage of the event type.

Token:	Description:
<code>ChartChangeEvent.Type.GENERAL</code>	A general event.
<code>ChartChangeEvent.Type.NEW_DATASET</code>	An event that signals that a new dataset has been added to the chart.
<code>ChartChangeEvent.Type.DATASET_UPDATED</code>	An event that signals that a dataset has been updated.

*Table 29.1: ChartChangeEvent tokens*

## 29.6 ChartChangeListener

### 29.6.1 Overview

An interface through which chart change event notifications are posted.

### 29.6.2 Methods

The interface defines a single method:

```
→ public void chartChanged(ChartChangeEvent event);
```

Receives notification of a change to a chart.

### 29.6.3 Notes

Some points to note:

- if a class needs to receive notification of changes to a chart, then it needs to implement this interface and register itself with the chart;
- the `ChartPanel` class implements this interface, and repaints the chart whenever a change event is received.

## 29.7 ChartProgressEvent

### 29.7.1 Overview

An event that contains information about the progress made during the rendering of a chart. Any class that implements the `ChartProgressListener` interface can register with the `JFreeChart` class and receive these events during chart rendering.

### 29.7.2 Constructor

To create a new event:

```
→ public ChartProgressEvent(Object source, JFreeChart chart, int type, int percent);
```

Creates a new event with the given attributes. The `source` may be the chart, or some subcomponent of the chart. The `type` identifies the event type, defined values include `DRAWING_STARTED` and `DRAWING_FINISHED` (others may be added in the future). The `percent` is an estimate of the amount of progress, in the range 0 to 100.

Typically, user code will receive events that have been constructed by `JFreeChart`, and won't need to create new event instances.

### 29.7.3 Methods

Accessor methods are provided for the event's attributes:

```
→ public JFreeChart getChart();
```

Returns the chart that the event relates to.

```
→ public void setChart(JFreeChart chart);
Sets the chart that the event belongs to (this should not be null).

→ public int getType();
Returns the event type, one of DRAWING_STARTED and DRAWING_FINISHED. Additional types may be defined in the future.

→ public void setType(int type);
Sets the drawing type, which should be one of DRAWING_STARTED and DRAWING_FINISHED. Additional types may be defined in the future.
```

This event provides a mechanism for finding out what percentage of the chart rendering has been completed. Unfortunately, this isn't fully implemented, so you cannot rely on it:

```
→ public int getPercent();
Returns the percentage complete for the chart's rendering. This should be a value in the range 0 to 100.

→ public void setPercent(int percent);
Sets the percentage complete for the chart's rendering. This should be a value in the range 0 to 100.
```

#### 29.7.4 Notes

This mechanism is intended to provide the ability to report progress on the rendering of slow drawing charts, but is not yet complete. It still serves a purpose in that it allows code to determine the point at which chart rendering is complete.

### 29.8 ChartProgressListener

#### 29.8.1 Overview

A listener that can receive progress updates from a chart. The listener will receive an event (DRAWING\_STARTED) when the chart drawing commences, and another event (DRAWING\_FINISHED) when the chart drawing is finished.<sup>1</sup>

#### 29.8.2 Method

This interface defines a single method that receives notification (from a `JFreeChart` instance) of the chart drawing progress:

```
→ public void chartProgress(ChartProgressEvent event);
Receives notification of the progress of chart rendering.
```

### 29.9 MarkerChangeEvent

#### 29.9.1 Overview

An event that is used to signal that some change has been made to a `Marker`. In JFreeChart, the plot classes listen for changes to any markers they manage, and notify the chart when such changes occur.

#### 29.9.2 Constructors

There is a single constructor:

```
→ public MarkerChangeEvent(Marker marker); [1.0.3]
Creates a new event with marker as the source. If marker is null, this constructor throws an
IllegalArgumentException.
```

---

<sup>1</sup>Originally it was planned that the listener should receive interim events during chart drawing, but this hasn't been implemented yet.

### 29.9.3 Methods

The following method is defined:

```
➔ public Marker getMarker(); [1.0.3]  
Returns the marker that triggered the event. This will never be null.
```

### 29.9.4 Notes

This class was introduced in version 1.0.3.

#### See Also

[MarkerChangeListener](#).

## 29.10 MarkerChangeListener

### 29.10.1 Overview

The interface through which [MarkerChangeEvent](#) notifications are posted. This interface is implemented by [CategoryPlot](#) and [XYPlot](#).

### 29.10.2 Methods

The interface defines a single method:

```
➔ public void markerChanged(MarkerChangeEvent event); [1.0.3]  
Receives notification of a change to a marker.
```

### 29.10.3 Notes

Some points to note:

- this interface was introduced in JFreeChart version 1.0.3 (prior to this version, markers did not generate change events);
- if a class needs to receive notification of changes to a marker, then it needs to implement this interface and register itself with the legend.

#### See Also

[MarkerChangeEvent](#).

## 29.11 PlotChangeEvent

### 29.11.1 Overview

An event that is used to provide information about changes to a plot. You can register an object with a [Plot](#) instance, provided that the object implements the [PlotChangeListener](#) interface, and it will receive a notification whenever the plot changes.

### 29.11.2 Notes

A [JFreeChart](#) object will automatically register itself with the [Plot](#) that it manages, and receive notification whenever the plot changes. The chart usually responds by raising a [ChartChangeEvent](#), which other listeners may respond to (for example, the [ChartPanel](#) if the chart is displayed in a GUI).

## 29.12 PlotChangeListener

### 29.12.1 Overview

An interface through which plot change event notifications are posted.

### 29.12.2 Methods

The interface defines a single method:

```
→ public void plotChanged(PlotChangeEvent event);
```

Receives notification of a change to a plot.

### 29.12.3 Notes

Some points to note:

- if a class needs to receive notification of changes to a plot, then it needs to implement this interface and register itself with the plot.
- the [JFreeChart](#) class implements this interface and automatically registers itself with the plot it manages.

## 29.13 RendererChangeEvent

### 29.13.1 Overview

An event that is used to provide information about changes to a renderer. If an object needs to receive notification of these events, its class should implement the [RendererChangeListener](#) interface so the object can register itself with the renderer via the `addChangeListener()` method.

### 29.13.2 Usage

Typically, you won't need to use this class directly. By default, JFreeChart's plot classes will automatically register (as a [RendererChangeListener](#)) with each renderer that is assigned to the plot. As a result, (most) changes to a renderer will cause the plot to receive notification of the change. The plot will usually respond by firing a [PlotChangeEvent](#) which, in turn, gets passed on to the chart and results in a [ChartChangeEvent](#) being fired. This chain of events is used to ensure that charts are automatically updated whenever a change is made to any component of the chart.

### 29.13.3 Notes

In the current implementation, the event just signals a change without specifying exactly what changed. A possible future enhancement would be to include information about the nature of the change, so that the listener(s) can decide what action to take in response to the event.

## 29.14 RendererChangeListener

### 29.14.1 Overview

An interface through which renderer change event notifications are posted. The [CategoryPlot](#) and [XYPlot](#) classes implement this interface so they can receive notification of changes to their renderer(s).

## 29.14.2 Methods

The interface defines a single method:

```
► public void rendererChanged(RendererChangeEvent event);
```

Receives notification of a change to a renderer.

## 29.14.3 Notes

If an `Object` needs to receive notification of changes to a renderer, then its class needs to implement this interface so the object can register itself with the renderer.

# 29.15 TitleChangeEvent

## 29.15.1 Overview

An event that is used to provide information about changes to a chart title (any subclass of `Title`).

## 29.15.2 Notes

This event is part of the overall mechanism that JFreeChart uses to automatically update charts whenever changes are made to components of the chart.

### See Also

[Title](#), [TitleChangeListener](#).

# 29.16 TitleChangeListener

## 29.16.1 Overview

An interface through which title change event notifications are posted.

## 29.16.2 Methods

The interface defines a single method:

```
► public void titleChanged(TitleChangeEvent event);
```

Receives notification of a change to a title.

## 29.16.3 Notes

If a class needs to receive notification of changes to a title, then it needs to implement this interface and register itself with the title.

### See Also

[TitleChangeEvent](#).

# Chapter 30

## Package: org.jfree.chart.imageMap

### 30.1 Overview

This package contains classes and interfaces that support the creation of HTML image maps. These image maps can be created using the [ImageMapUtilities](#) class, typically from a servlet.

### 30.2 DynamicDriveToolTipTagFragmentGenerator

#### 30.2.1 Overview

A tool-tip fragment generator that generates tool-tips that are designed to work with the Dynamic Drive DHTML Tip Message library:

<http://www.dynamicdrive.com>

This class implements the [ToolTipTagFragmentGenerator](#) interface.

### 30.3 ImageMapUtilities

#### 30.3.1 Overview

This class contains some utility methods that are useful for creating HTML image maps. A range of demos ([ImageMapDemo1-6.java](#)) are included in the JFreeChart demo collection.

#### 30.3.2 Methods

Several methods provide the ability to write an image map directly to a stream:<sup>1</sup>

```
➔ public static void writeImageMap(PrintWriter writer, String name,  
ChartRenderingInfo info);
```

Writes an image map using `info` as the source of chart entity information. This is equivalent to `writeImageMap(writer, name, info, new StandardToolTipTagFragmentGenerator(),  
new StandardURLTagFragmentGenerator());`

```
➔ public static void writeImageMap(PrintWriter writer, String name,  
ChartRenderingInfo info, boolean useOverLibForToolTips);
```

Writes an image map using `info` as the source of chart entity information. This will use an instance of [OverLIBToolTipTagFragmentGenerator](#) to format the tooltip output, if requested.

---

<sup>1</sup>Note that in the current implementation, the image map is created entirely in memory and then written to the stream, which is not as efficient as it could be.

```
→ public static void writeImageMap(PrintWriter writer, String name,
    ChartRenderingInfo info, ToolTipTagFragmentGenerator toolTipTagFragmentGenerator,
    URLTagFragmentGenerator urlTagFragmentGenerator) throws IOException;
Writes an image map using info as the source of chart entity information. This method first calls getImageMap() to create the image map text, then writes it to write. The tooltip and URL fragment generators provide the option to customize the imagemap output.
```

To create an HTML image map string:

```
→ public static String getImageMap(String name, ChartRenderingInfo info);
Returns an image map based on the chart entity information in info. This is equivalent to
getImageMap(name, info, new StandardToolTipTagFragmentGenerator(), new StandardURL-
TagFragmentGenerator());

→ public static String getImageMap(String name, ChartRenderingInfo info,
    ToolTipTagFragmentGenerator toolTipTagFragmentGenerator,
    URLTagFragmentGenerator urlTagFragmentGenerator);
Returns an HTML image map based on the chart entity information in info.
```

The following method creates standard escape sequences for “unsafe” characters in a string that will be embedded in HTML output:

```
→ public static String htmlEscape(String input); [1.0.9]
Returns a string that corresponds to the input string after replacing certain characters with
standard HTML escape sequences. If input is null, this method throws an IllegalArgumentException.
```

### 30.3.3 Notes

Some points to note:

- tooltip and URL *content* is controlled by generators defined in the packages:
  - `org.jfree.chart.labels.*` for tooltips;
  - `org.jfree.chart.urls.*` for URLs.

...whereas the tooltip and URL fragment generators defined in this package are concerned with variation in the HTML tags that get incorporated into the HTML image map.

## 30.4 OverLIBToolTipTagFragmentGenerator

### 30.4.1 Overview

A tool-tip generator that generates tool-tips for use with the OverLIB library. See this URL for details:

<http://www.bosrup.com/web/overlib/>

This class implements the `ToolTipTagFragmentGenerator` interface.

## 30.5 StandardToolTipTagFragmentGenerator

### 30.5.1 Overview

A tool-tip generator that generates tool-tips using the HTML title attribute. This class implements the `ToolTipTagFragmentGenerator` interface.

## 30.6 StandardURLTagFragmentGenerator

### 30.6.1 Overview

A default implementation of the [URLTagFragmentGenerator](#) interface. Instances of this class are created by some methods in the [ImageMapUtilities](#) class, to create the `href` elements in the HTML image map. You can supply an alternative implementation if you want greater control over the hyperlinks in the image map.

### 30.6.2 Constructor

This class has only the default constructor. Instances are stateless, so there are no attributes to define.

### 30.6.3 Methods

This class implements a single method:

```
► public String generateURLFragment(String urlText);
```

Returns a URL fragment containing the specified URL text. The implementation is very simple:

```
return " href=\"" + urlText + "\"";
```

### 30.6.4 Equals, Cloning and Serialization

This class has no state, and is typically used for a single call to the `getImageMap()` or `writeImageMap()` methods in the [ImageMapUtilities](#) class. Therefore, it does not override the `equals()` method, and is neither cloneable nor serializable.

#### See Also

[URLTagFragmentGenerator](#).

## 30.7 ToolTipTagFragmentGenerator

### 30.7.1 Overview

The interface that must be implemented by a class that generates tooltip tag fragments for an HTML image map.

Classes that implement this interface include:

- [StandardToolTipTagFragmentGenerator](#);
- [DynamicDriveToolTipTagFragmentGenerator](#);
- [OverLIBToolTipTagFragmentGenerator](#);

### 30.7.2 Methods

This interface defines a single method:

```
► public String generateToolTipFragment(String toolTipText);
```

Returns a tooltip fragment based on the supplied tool-tip text.

## 30.8 URLTagFragmentGenerator

### 30.8.1 Overview

The interface that must be implemented by a class that generates the URL tag fragment for an HTML image map. For example, in the following `area` element, the URL fragment is shown underlined:

```
<area shape="rect" coords="553,259,564,288" title="(Series 3, Type 8) = 12"  
alt="" href="bar_chart_detail.jsp?series=Series+3&category=Type+8"/>
```

The URL content is created elsewhere, but this generator is responsible for generating the surrounding text (the `href` tag in this case). The `StandardURLTagFragmentGenerator` class is the only implementation of this interface provided by JFreeChart.

### 30.8.2 Methods

This interface defines a single method:

```
→ public String generateURLFragment(String urlText);  
Returns a URL fragment based on the supplied URL text.
```

### 30.8.3 Notes

You can pass an instance of a class that implements this interface to the `getImageMap()` and `writeImageMap()` methods in the `ImageMapUtilities` class.

# Chapter 31

## Package: org.jfree.chart.labels

### 31.1 Introduction

This package contains interfaces and classes for generating labels for the individual data items in a chart. There are two label types:

- *item labels* – text displayed in, on or near to each data item in a chart;
- *tooltips* – text that is displayed when the mouse pointer “hovers” over a data item in a chart.

Section 11 contains information about using tool tips and section 12 contains information about using item labels.

### 31.2 AbstractCategoryItemLabelGenerator

#### 31.2.1 Overview

An abstract base class for creating item labels for a `CategoryItemRenderer`. Known subclasses include:

- `StandardCategoryToolTipGenerator`;
- `StandardCategoryItemLabelGenerator`.

The generator uses Java’s `MessageFormat` class to construct labels by substituting any or all of the objects listed in table 31.1.

The data value is formatted before it is passed to the `MessageFormat`—you can specify the `NumberFormat` or `DateFormat` that is used to preformat the value via the constructor.

#### 31.2.2 Constructors

Two (protected) constructors are provided, the difference between them is the type of formatter (number or date) for the data values. In both cases, the `labelFormat` parameter determines the overall structure of the generated label—you can use the substitutions listed in table 31.1.

Code:	Description:
{0}	The series name.
{1}	The category label.
{2}	The (preformatted) data value.

Table 31.1: `MessageFormat` substitutions

```

    ➔ protected AbstractCategoryItemLabelGenerator(String labelFormat,
        NumberFormat formatter);
Creates a new generator that formats the data values using the supplied NumberFormat instance.

    ➔ protected AbstractCategoryItemLabelGenerator(String labelFormat,
        DateFormat formatter);
Creates a new generator that formats the data values using the supplied DateFormat instance.

```

## Methods

To generate a label string:

```

    ➔ protected String generateLabelString(CategoryDataset dataset, int row, int column);
Generates a label string. This method first calls the createItemArray() function, then passes
the result to Java's MessageFormat to build the required label.

```

The following function builds the array (`Object[]`) that contains the items that can be substituted by the `MessageFormat` code:

```

    ➔ protected Object[] createItemArray(CategoryDataset dataset, int row, int column);
Returns an array containing three items, the series name, the category label and the formatted
data value.

```

### 31.2.3 Notes

Some points to note:

- the `StandardCategoryToolTipGenerator` and `StandardCategoryItemLabelGenerator` classes are extensions of this class;
- instances of this class are `Cloneable` and `Serializable`.

## 31.3 AbstractPieItemLabelGenerator

### 31.3.1 Overview

A base class used to create label generators for pie charts. Subclasses include:

- `StandardPieSectionLabelGenerator`;
- `StandardPieToolTipGenerator`.

### 31.3.2 Constructor

The following constructor is provided for use by subclasses:

```

    ➔ protected AbstractPieItemLabelGenerator(String labelFormat,
        NumberFormat numberFormat, NumberFormat percentFormat);
Creates a new instance with the specified formatting attributes. The labelFormat is a string
that is used by an internal MessageFormat instance to compose a section label for an item in
the dataset—see the generateSectionLabel() method. If any of the arguments is null, this
constructor will throw an IllegalArgumentException.

```

### 31.3.3 Methods

The following methods are defined:

```

    ➔ public String getLabelFormat();
Returns the formatting string that is used by the internal MessageFormat instance.

```

Token:	Description:
{0}	The series name.
{1}	The (preformatted) x-value.
{2}	The (preformatted) y-value.

Table 31.2: *MessageFormat substitutions*

```
► public NumberFormat getNumberFormat();
Returns the number formatter used to preformat each dataset value before incorporating it
into a section label. Note that the label format string (see getLabelFormat()) may or may not
include the dataset value.

► public NumberFormat getPercentFormat();
Returns the number formatter used to preformat each percentage value1 before incorporating
it into a section label. Note that the label format string (see getLabelFormat()) may or may
not include the percentage value.

► protected Object[] createItemArray(PieDataset dataset, Comparable key);
Creates an array of objects to pass to the internal MessageFormat instance used to create the
section label. The array contains four String objects:


- item 0: key.toString();
- item 1: the dataset value formatted using getNumberFormat(), or null;
- item 2: the dataset value as a percentage formatted using getPercentFormat();
- item 3: the total of all the dataset values, formatted using getNumberFormat().



► protected String generateSectionLabel(PieDataset dataset, Comparable key);
Returns a section label for the specified item in the given dataset. This method is called by
JFreeChart, it typically won't be called by external code.
```

### 31.3.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this generator for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

## 31.4 AbstractXYItemLabelGenerator

### 31.4.1 Overview

An abstract base class that creates item labels (on demand) for an `XYItemRenderer`. Subclasses include:

- `StandardXYToolTipGenerator`; and
- `StandardXYItemLabelGenerator`.

Internally, the generator uses Java's `MessageFormat` class to construct labels by substituting any or all of the tokens listed in table 31.2. The x and y values are formatted before they are passed to `MessageFormat`—you can specify the `NumberFormat` or `DateFormat` that is used to preformat the values via the constructor.

---

<sup>1</sup>The percentage value is the dataset value expressed as a percentage of the sum of all dataset values.

### 31.4.2 Constructors

All the constructors for this class are `protected` and provided for use by subclasses only. The provided constructors give you control over the formatters (number or date) used for the x and y data values. In all cases, the `labelFormat` parameter determines the overall structure of the generated label—you can use the substitutions listed in table 31.2.

- `protected AbstractXYItemLabelGenerator();`  
Creates a new generator that formats the data values using the default number formatter for the current locale.
- `protected AbstractXYItemLabelGenerator(String formatString, NumberFormat xFormat, NumberFormat yFormat);`  
Creates a new generator that formats the data values using the supplied `NumberFormat` instances.
- `protected AbstractXYItemLabelGenerator(String formatString, DateFormat xFormat, NumberFormat yFormat);`  
Creates a new generator that formats the x-values as dates and the y-values as numbers.
- `protected AbstractXYItemLabelGenerator(String formatString, DateFormat xFormat, NumberFormat yFormat); [1.0.4]`  
Creates a new generator that formats the x-values as numbers and the y-values as dates.
- `protected AbstractXYItemLabelGenerator(String formatString, DateFormat xFormat, DateFormat yFormat);`  
Creates a new generator that formats both the x and y values as dates.

### Attributes

To obtain the format string that was specified in the constructor:

- `public String getFormatString();`  
Returns the format string that will be passed to a `MessageFormat` instance when creating the item label.

In the typical case, the x and y values are formatted as numbers:

- `public NumberFormat getXFormat();`  
Returns the formatter for the x-values (never `null`). Note that if the `getXDateFormat()` method returns a non-null formatter, it will be used instead.
- `public NumberFormat getYFormat();`  
Returns the formatter for the y-values (never `null`). Note that if the `getYDateFormat()` method returns a non-null formatter, it will be used instead.

These formatters can be (optionally) overridden by date formatters:

- `public DateFormat getXDateFormat();`  
Returns the (date) formatter for the x-values. If this is `null`, the x-values will be formatted as numbers—see `getXFormat()`.
- `public DateFormat getYDateFormat();`  
Returns the (date) formatter for the y-values. If this is `null`, the y-values will be formatted as numbers—see `getYFormat()`.

### Other Methods

The following methods are called by JFreeChart—you won't normally call them directly from your own code:

- `protected String generateLabelString(XYDataset dataset, int series, int item);`  
Generates a label string. This method first calls the `createItemArray()` method, then passes the result to Java's `MessageFormat` to build the required label.

The following function builds the array (`Object[]`) that contains the items that can be substituted by the `MessageFormat` code:

- `protected Object[] createItemArray(XYDataset dataset, int series, int item);`  
Returns an array containing three items, the series name, the formatted x and y data values.

### 31.4.3 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`

Tests this generator for equality with an arbitrary object. This method returns `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `AbstractXYItemLabelGenerator`;
- `obj` has the same attributes as this generator.

Instances of this class are `Cloneable` and `Serializable`.

### 31.4.4 Notes

Some points to note:

- the `StandardXYToolTipGenerator` and `StandardXYItemLabelGenerator` classes are extensions of this class.

## 31.5 BoxAndWhiskerToolTipGenerator

### 31.5.1 Overview

A *tool tip generator* for a box-and-whisker chart. This is the default generator used by the `BoxAndWhiskerRenderer` class.

## 31.6 BoxAndWhiskerXYToolTipGenerator

### 31.6.1 Overview

A *tool tip generator* for a box-and-whisker chart. This is the default generator used by the `XYBoxAndWhiskerRenderer` class.

## 31.7 CategoryItemLabelGenerator

### 31.7.1 Overview

A *category item label generator* is an object that assumes responsibility for creating the text strings that will be used for item labels in a chart. A generator is assigned to a renderer using the `setItemLabelGenerator()` method in the `CategoryItemRenderer` interface. This interface defines the API through which the renderer will communicate with the generator.

### 31.7.2 Usage

Chapter 12 contains information about using item labels.

### 31.7.3 Methods

The renderer will call this method to obtain an item label:

► `public String generateLabel(CategoryDataset data, int series, int category);`

Returns a string that will be used to label the specified item. Classes that implement this method are permitted to return `null` for the result, in which case no label will be displayed for that item.

Additional methods:

```
→ public String generateRowLabel(CategoryDataset dataset, int row);
  Returns a label for the given row in the dataset.

→ public String generateColumnLabel(CategoryDataset dataset, int column);
  Returns a label for the given column in the dataset.
```

### 31.7.4 Notes

Some points to note:

- the `StandardCategoryItemLabelGenerator` class provides one implementation of this interface, but you can also write your own class that implements this interface, and take complete control over the generation of item labels.

## 31.8 CategorySeriesLabelGenerator

### 31.8.1 Overview

An interface defining the API that a caller (typically a `CategoryItemRenderer`) can use to obtain a label for a series in a dataset. This interface is implemented by the `StandardCategorySeriesLabelGenerator` class.

### 31.8.2 Methods

The renderer will call this method to obtain an item label:

```
→ public String generateLabel(CategoryDataset dataset, int series);
  Returns a string that will be used to label the specified series.
```

### 31.8.3 Notes

Some points to note:

- by convention, all classes that implement this interface should be either:
  - immutable; or
  - implement the `PublicCloneable` interface.

This provides a mechanism for a referring class to determine whether or not it needs to clone the generator, and access to the `clone()` method in the case that the generator is cloneable.

## 31.9 CategoryToolTipGenerator

### 31.9.1 Overview

A *category tool tip generator* is an object that assumes responsibility for creating the text strings that will be used for tooltips in a chart. A generator is assigned to a renderer using the `setToolTipGenerator()` method in the `CategoryItemRenderer` interface. This interface defines the API through which the renderer will communicate with the generator.

### 31.9.2 Methods

The renderer will call this method to obtain the tooltip text for an item:

```
→ public String generateToolTip(CategoryDataset data, int series, int category);
  Returns a string that will be used as the tooltip text for the specified item. If null is returned,
  no tool tip will be displayed.
```

### 31.9.3 Notes

Some points to note:

- the [StandardCategoryToolTipGenerator](#) provides one implementation of this interface, but you can also write your own class that implements this interface, and take complete control over the generation of item labels and tooltips;
- refer to chapter [11](#) for information about using tool tips.

## 31.10 ContourToolTipGenerator

### 31.10.1 Overview

The interface that must be implemented by all contour tool tip generators. When a [ContourPlot](#) requires tooltip text for a data item, it will obtain it via this interface.

*This interface is deprecated as of JFreeChart version 1.0.4.*

### 31.10.2 Methods

The interface defines a single method for obtaining the tooltip text for a data item:

```
➔ public String generateToolTip(ContourDataset data, int item);
```

Returns a string that can be used as the tooltip text for a data item.

#### See Also

[ContourPlot](#).

## 31.11 CustomXYToolTipGenerator

### 31.11.1 Overview

A tool tip generator (for use with an [XYItemRenderer](#)) that returns a predefined tool tip for each data item.

### 31.11.2 Methods

To specify the text to use for the tool tips:

```
➔ public void addToolTipSeries(List toolTips);
```

Adds the list of tool tips (for one series) to internal storage. These tool tips will be returned (without modification) by the generator for each data item.

### 31.11.3 Notes

See section [11](#) for information about using tool tips with JFreeChart.

## 31.12 HighLowItemLabelGenerator

### 31.12.1 Overview

A *label generator* that is intended for use with the [HighLowRenderer](#) class. The generator will only return tooltips for a dataset that is an implementation of the [OHLCDataset](#) interface.

### 31.12.2 Constructors

To create a new label generator:

```
➔ public HighLowItemLabelGenerator(DateFormat dateFormatter, NumberFormat numberFormatter);
```

Creates a new label generator that uses the specified date and number formatters.

### 31.12.3 Methods

The key method constructs a `String` to be used as the tooltip text for a particular data item:

```
➔ public String generateToolTip(XYDataset dataset, int series, int item);
```

Returns a string containing the date, value, high value, low value, open value and close value for the data item. This method will return `null` if the dataset does not implement the `OHLCDataset` interface.

The following method is intended to generate an item label for display in a chart, but since the renderer does not yet support this the method simply returns `null`:

```
➔ public String generateItemLabel(XYDataset dataset, int series, int category);
```

Returns `null`. To be implemented.

### 31.12.4 Notes

See section 11 for an overview of tool tips with JFreeChart.

## 31.13 IntervalCategoryItemLabelGenerator

### 31.13.1 Overview

An *label generator* that can be used with any `CategoryItemRenderer`. This generator will detect if the dataset supplied to the renderer is an implementation of the `IntervalCategoryDataset` interface, and will generate labels that display both the *start value* and the *end value* for each item.

### 31.13.2 Constructors

The default constructor will create a label generator that formats the data values as numbers, using the platform default number format:

```
➔ public IntervalCategoryItemLabelGenerator();
```

Creates a new label generator with a default number formatter.

If you prefer to set the number format yourself, use the following constructor:

```
➔ public IntervalCategoryItemLabelGenerator(NumberFormat formatter);
```

Creates a new label generator with a specific number formatter.

In some cases, the data values in the dataset will represent dates (encoded as milliseconds since midnight, 1-Jan-1970 GMT, as for `java.util.Date`). In this case, you can create a label generator using the following constructor:

```
➔ public IntervalCategoryItemLabelGenerator(DateFormat formatter);
```

Creates a new label generator that formats the start and end data values as dates.

### 31.13.3 Notes

The `createGanttChart()` in the `ChartFactory` class uses this type of label generator (with date formatting).

## 31.14 IntervalCategoryToolTipGenerator

### 31.14.1 Overview

An *tool tip generator* that can be used with any [CategoryItemRenderer](#). This generator will detect if the dataset supplied to the renderer is an implementation of the [IntervalCategoryDataset](#) interface, and will generate labels that display both the *start value* and the *end value* for each item.

### 31.14.2 Constructors

The default constructor will create a label generator that formats the data values as numbers, using the platform default number format:

```
➔ public IntervalCategoryToolTipGenerator();
Creates a new tool tip generator with a default number formatter.
```

If you prefer to set the number format yourself, use the following constructor:

```
➔ public IntervalCategoryToolTipGenerator(NumberFormat formatter);
Creates a new tool tip generator with a specific number formatter.
```

In some cases, the data values in the dataset will represent dates (encoded as milliseconds since midnight, 1-Jan-1970 GMT, as for `java.util.Date`). In this case, you can create a label generator using the following constructor:

```
➔ public IntervalCategoryToolTipGenerator(DateFormat formatter);
Creates a new tool tip generator that formats the start and end data values as dates.
```

### 31.14.3 Notes

The `createGanttChart()` in the [ChartFactory](#) class uses this type of label generator (with date formatting).

## 31.15 ItemLabelAnchor

### 31.15.1 Overview

An *item label anchor* is used by a renderer to calculate a fixed point (the *item label anchor point*) relative to a data item on a chart. This point becomes a reference point that an item label can be aligned to.

This class defines 25 anchors. The numbers 1 to 12 are used and roughly correspond to the positions of the hours on a clock face. In addition, positions are defined relative to an “inside” ring and an “outside” ring - see figure 31.1 for an illustration.

With 12 points on the inside circle, 12 points on the outside circle, plus a “center” anchor point, in all there are 25 possible anchor points.

For some renderers, the circular arrangement of anchor points doesn’t make sense, so the renderer is free to modify the anchor positions (see the [BarRenderer](#) class for an example).

### 31.15.2 Usage

The [ItemLabelPosition](#) class includes an item label anchor as one of the attributes that define the location of item labels drawn by a renderer.

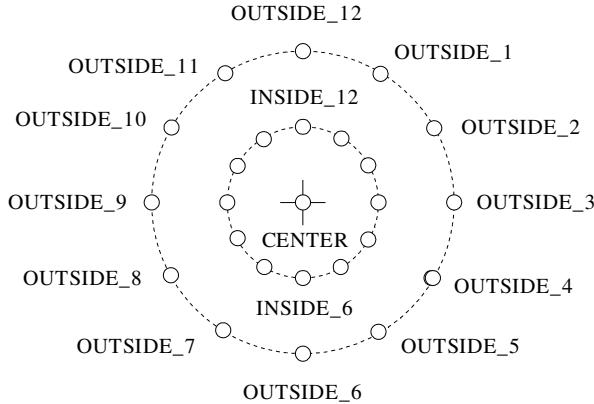


Figure 31.1: The Item Label Anchors

## 31.16 ItemLabelPosition

### 31.16.1 Overview

This class is used to specify the position of item labels on a chart. Four attributes are used to specify the position:

- the *item label anchor* - the renderer will use this to calculate an (x, y) anchor point on the chart near to the data item that the item label corresponds to (see [ItemLabelAnchor](#));
- the *text anchor* - this is a point relative to the item label text which will be aligned with the item label anchor point above;
- the *rotation anchor* - this is another point somewhere on the item label about which the text will be rotated (if there is a rotation);
- the *rotation angle* - this specifies the amount of rotation about the rotation point.

These four attributes provide a lot of scope for placing item labels in interesting ways.

### 31.16.2 Usage

The [AbstractRenderer](#) class provides methods for specifying the item label position for positive and negative data values separately:

```
→ public void setPositiveItemLabelPosition(ItemLabelPosition position);
Sets the item label position for positive data values.

→ public void setNegativeItemLabelPosition(ItemLabelPosition position);
Sets the item label position for negative data values.
```

### 31.16.3 Constructors

This class defines three constructors:

```
→ public ItemLabelPosition();
Creates a default instance. Equivalent to this(ItemLabelAnchor.OUTSIDE12, TextAnchor.BOTTOM_CENTER) —
see below.
```

```
➔ public ItemLabelPosition(ItemLabelAnchor itemLabelAnchor, TextAnchor textAnchor);
Creates a position with no text rotation. Equivalent to this(itemLabelAnchor, textAnchor,
TextAnchor.CENTER, 0.0).

➔ public ItemLabelPosition(ItemLabelAnchor itemLabelAnchor, TextAnchor textAnchor, TextAnchor
rotationAnchor, double angle);
Creates a new ItemLabelPosition instance. None of the arguments can be null.
```

### 31.16.4 Methods

Accessor methods are defined for the attributes specified via the constructor.

```
➔ public ItemLabelAnchor getItemLabelAnchor();
Returns the anchor point for the item label (never null). This defines a location relative to the
data value on the chart to which the item label will be aligned.

➔ public TextAnchor getTextAnchor();
Returns the reference point on the label text that will be aligned to the anchor point on the
chart. This method never returns null.

➔ public TextAnchor getRotationAnchor();
Returns the reference point on the label text about which any rotation will be performed. This
method never returns null.

➔ public double getAngle();
Returns the angle of rotation for the text (in radians).
```

None of the above attributes can be modified post-construction, because instances of this class are immutable.

### 31.16.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
Tests this ItemLabelPosition for equality with an arbitrary object. Returns true if and only if:


- obj is not null;
- obj is an instance of ItemLabelPosition;
- obj has the same attributes as this instance.

```

Instances of this class are immutable, so they don't need to be `Cloneable`. Instances of this class are `Serializable`.

## 31.17 MultipleXYSeriesLabelGenerator

### 31.17.1 Overview

A series label generator that can return multiple labels (separated by a newline character) for a single series. You might use this generator to show legend items in two or more languages, for example.

### 31.17.2 Constructors

This class defines two constructors:

```
➔ public MultipleXYSeriesLabelGenerator();
Equivalent to MultipleXYSeriesLabelGenerator("{0}")—see the next constructor.

➔ public MultipleXYSeriesLabelGenerator(String format);
Creates a new generator where the main label is generated with the specified format string. At
label generation time, any occurrence of {0} in the format string will be replaced by the series
name.
```

### 31.17.3 Methods

The following methods are defined:

- `public void addSeriesLabel(int series, String label);`  
Adds an additional label for a series. This method does NOT fire a change event.
- `public void clearSeriesLabels(int series);`  
Clears the additional labels for the specified series. This method does NOT fire a change event.
- `public String generateLabel(XYDataset dataset, int series);`  
Generates the label string for a series in the specified dataset.
- `protected Object[] createItemArray(XYDataset dataset, int series);`  
Creates and returns an array (of length 1) containing the string representation of the key for the specified series. This method is used internally.

### 31.17.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this generator for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### See Also

[XYSeriesLabelGenerator](#).

## 31.18 PieSectionLabelGenerator

### 31.18.1 Overview

An interface that defines the methods used by a `PiePlot` to request section labels. Two generators can be specified for a `PiePlot`:

- `setLabelGenerator()` – generates the labels displayed directly on the plot;
- `setLegendLabelGenerator()` – generates the labels displayed in the plot's legend (if it has one).

The `StandardPieSectionLabelGenerator` class provides a standard implementation of this interface.

### 31.18.2 Usage

The `PieChartDemo2.java` demo application includes code to customise the section labels, so you can refer to that demo for sample usage.

### 31.18.3 Methods

The `PiePlot` class will call the following method to obtain a section label for each section in a pie chart as it is being drawn:

- `public String generateSectionLabel(PieDataset dataset, Comparable key);`  
Returns a section label for the specified item in the dataset. A class implementing this method can return `null`, in which case no label will be displayed for the pie section.

An alternative method that returns an `AttributedString` is defined, but is currently not used:

- `public AttributedString generateAttributedSectionLabel(PieDataset dataset, Comparable key);`  
Returns an `AttributedString` for the section label for the specified item in the dataset. This method is not used at present—classes implementing this interface can safely return `null` for this method.

### 31.18.4 Notes

Some points to note:

- you can develop your own label generator, register it with a [PiePlot](#), and take full control over the labels that are generated.

#### See Also

[PieToolTipGenerator](#).

## 31.19 PieToolTipGenerator

### 31.19.1 Overview

The interface that must be implemented by a *pie tool tip generator*, a class used to generate tool tips for a pie chart.

### 31.19.2 Methods

The [PiePlot](#) class will call the following method to obtain a tooltip for each section in a pie chart:

```
➔ public String generateToolTip(PieDataset data, Comparable key);
Returns a String that will be used as the tool tip text. This method can return null in which
case no tool tip will be displayed.
```

### 31.19.3 Notes

Some points to note:

- the [StandardPieToolTipGenerator](#) class provides an implementation of this interface;
- you can develop your own tool tip generator, register it with a [PiePlot](#), and take full control over the labels that are generated;
- section 11 contains information about using tool tips with JFreeChart.

## 31.20 StandardCategoryItemLabelGenerator

### 31.20.1 Overview

A generator that can be assigned to a [CategoryItemRenderer](#) for the purpose of generating item labels (this class implements the [CategoryItemLabelGenerator](#) interface). This class is very flexible in the format of the labels it can generate. It uses Java's [MessageFormat](#) class to create a label which can contain any of the items listed in table 31.1. The data value can be formatted using any [NumberFormat](#) instance.

### 31.20.2 Usage

Most often you will assign a generator to a renderer and then never need to refer to it again:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
CategoryItemRenderer renderer = plot.getRenderer();
CategoryItemLabelGenerator generator = new StandardCategoryItemLabelGenerator(
    "{2}", new DecimalFormat("0.00"));
renderer.setItemLabelGenerator(generator);
renderer.setItemLabelsVisible(true);
```

The renderer will call the generator's methods when necessary. See section 12 for more information.

### 31.20.3 Constructors

To create a default generator:

```
↳ public StandardCategoryItemLabelGenerator();
```

Creates a new generator that formats values using the default number format for the user's locale. "{2}" is used as the label format string (that is, just the data value).

To create a generator that formats values as numbers:

```
↳ public StandardCategoryItemLabelGenerator(String labelFormat,
    NumberFormat formatter);
```

Creates a generator that formats values as numbers using the supplied formatter. The `labelFormat` is passed to a `MessageFormat` to control the structure of the generated label, and can use any of the substitutions listed in table 31.1.

To create a generator that formats values as dates (interpreting the numerical value as milliseconds since 1-Jan-1970, in the same way as `java.util.Date`):

```
↳ public StandardCategoryItemLabelGenerator(String labelFormat,
    DateFormat formatter);
```

Creates a generator that formats values as dates using the supplied formatter. The `labelFormat` is passed to a `MessageFormat` to control the structure of the generated label, and can use any of the substitutions listed in table 31.1.

### 31.20.4 Methods

The renderer will call the following method whenever it requires an item label:

```
↳ public String generateLabel(CategoryDataset dataset,
    int series, int category);
```

Generates an item label for the specified data item.

### 31.20.5 Notes

Some points to note:

- instances of this class are cloneable and serializable, and the `PublicCloneable` interface is implemented;
- for a demo, see `ItemLabelDemo3.java` in the JFreeChart demo collection.

## 31.21 StandardCategorySeriesLabelGenerator

### 31.21.1 Overview

A generator that can be assigned to a `CategoryItemRenderer` for the purpose of generating series labels (this class implements the `CategorySeriesLabelGenerator` interface) for the legend. This class uses Java's `MessageFormat` class to substitute the series name into an arbitrary string containing the token {0}.

### 31.21.2 Usage

Most often you will assign a generator to a renderer and then never need to refer to it again:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
AbstractCategoryItemRenderer renderer = (AbstractCategoryItemRenderer) plot.getRenderer();
CategorySeriesLabelGenerator generator = new StandardCategorySeriesLabelGenerator("{0}");
renderer.setLegendItemLabelGenerator(generator);
renderer.setItemLabelsVisible(true);
```

The renderer will call the generator's methods when necessary.

Code:	Description:
{0}	The series key (or name).
{1}	The category.
{2}	The item value.

Table 31.3: *MessageFormat substitutions for StandardCategoryToolTipGenerator*

### 31.21.3 Constructors

To create a default generator:

```
→ public StandardCategorySeriesLabelGenerator();
```

Creates a new generator that uses just the series name as the label.

To create a generator that formats with a custom format string:

```
→ public StandardCategorySeriesLabelGenerator(String labelFormat);
```

Creates a generator with the given format string. The `labelFormat` is passed to a `MessageFormat` to control the structure of the generated label, with {0} being substituted by the series name.

### 31.21.4 Methods

The renderer will call the following method whenever it requires a series label:

```
→ public String generateLabel(CategoryDataset dataset, int series);
```

Generates a series label for the specified series. This method is typically called by JFreeChart, not by external code.

### 31.21.5 Equals, Cloning and Serialization

This class override the equals method:

```
→ public boolean equals(Object obj);
```

Tests this generator for equality with an arbitrary object, returning `true` if and only if:

- `obj` is an instance of `StandardCategorySeriesLabelGenerator`;
- `obj` has the same `formatPattern` string as this generator.

Instances of this class are `Cloneable` and `Serializable`.

## 31.22 StandardCategoryToolTipGenerator

### 31.22.1 Overview

A generator that can be assigned to a `CategoryItemRenderer` for the purpose of generating tooltips. A format string provides the general template for each tool tip item, and Java's `MessageFormat` class is used to substitute actual values from the dataset (the series key/name, the category, and the data value). Table 31.3 lists the items that can be included for substitution.

### 31.22.2 Usage

This class provides an easy way to customise the tool tip text generated by a `CategoryItemRenderer`. This example shows how to create a new tool tip generator, and assign it to the plot's renderer:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setToolTipGenerator(new StandardCategoryToolTipGenerator(
    "The value is {2}, the series is {0} and the category is {1}.",
    NumberFormat.getInstance());
```

Once the generator is set, nothing more needs to be done—the renderer will call the generator's methods when necessary.

### 31.22.3 Constructors

This class has a default constructor:

```
→ public StandardCategoryToolTipGenerator();
```

Creates a new generator that creates tooltips using the format string “{0},{1} = {2}”. The data value is formatted using the default number format for the user’s locale.

To create a generator that formats values as numbers:

```
→ public StandardCategoryToolTipGenerator(String labelFormat, NumberFormat formatter);
```

Creates a generator that creates tooltips using the specified format string and number formatter.

An `IllegalArgumentException` is thrown if either argument is `null`.

To create a generator that formats values as dates (interpreting the numerical value as milliseconds since 1-Jan-1970, in the same way as `java.util.Date`):

```
→ public StandardCategoryToolTipGenerator(String labelFormat, DateFormat formatter);
```

Creates a generator that creates tooltips using the specified format string and date formatter.

In this case, the data value is interpreted as the number of milliseconds since 1-Jan-1970 (as for `java.util.Date`). An `IllegalArgumentException` is thrown if either argument is `null`.

### 31.22.4 Methods

When the renderer requires a tool tip, it will call the following method:

```
→ public String generateToolTip(CategoryDataset dataset,
int series, int category);
```

Generates a tooltip for the specified data item using the format string and number (or date) formatter supplied to the constructor.

### 31.22.5 Notes

Some points to note:

- this class implements the `CategoryToolTipGenerator` and `PublicCloneable` interfaces;
- section 11 contains information about using tool tips with JFreeChart.

## 31.23 StandardContourToolTipGenerator

### 31.23.1 Overview

A default implementation of the `ContourToolTipGenerator` interface.

*This class is deprecated as of JFreeChart version 1.0.4.*

## 31.24 StandardPieSectionLabelGenerator

### 31.24.1 Overview

A generator that is used to create section labels for a `PiePlot`. The generator uses Java’s `MessageFormat` class to construct labels by substituting any or all of the objects listed in table 31.4. The default section label format is “{0}”,<sup>2</sup> which displays the item key as a string.

This class implements the `PieSectionLabelGenerator` interface.

---

<sup>2</sup>This is defined in the `DEFAULT_SECTION_LABEL_FORMAT` constant field.

Code:	Description:
{0}	The item key.
{1}	The item value.
{2}	The item value as a percentage of the total.
{3}	The total of all values in the dataset.

Table 31.4: *MessageFormat* substitutions for the *StandardPieSectionLabelGenerator*.

### 31.24.2 Usage

You can use this class when you want to change the format of the the section labels on a pie chart. For example, to show percentages in the pie section labels:

```
PiePlot plot = (PiePlot) chart.getPlot();
PieSectionLabelGenerator generator = new StandardPieSectionLabelGenerator(
    "{0} = {2}", new DecimalFormat("0"), new DecimalFormat("0.00%"));
plot.setLabelGenerator(generator);
```

### 31.24.3 Constructors

The default constructor uses number and percentage formatters appropriate for the default locale:

► public StandardPieSectionLabelGenerator();  
Creates a default section label generator.

You can create a generator with a specific format string:

► public StandardPieSectionLabelGenerator(String labelFormat);  
Equivalent to `StandardPieSectionLabelGenerator(labelFormat, Locale.getInstance())`—see below.  
► public StandardPieSectionLabelGenerator(Locale locale); [1.0.7]  
Equivalent to `StandardPieSectionLabelGenerator(DEFAULT_SECTION_LABEL_FORMAT, locale)`—see below.  
► public StandardPieSectionLabelGenerator(String labelFormat, Locale locale); [1.0.7]  
Creates a generator using the specified format string. The item value and percentage (if included in the format string) will be formatted using default formatters for the specified locale. If `labelFormat` is `null`, this constructor throws an `IllegalArgumentException`.

The final constructor allows you to specify the item value and percentage formatters:

► public StandardPieSectionLabelGenerator(String labelFormat,
NumberFormat numberFormat, NumberFormat percentFormat)  
Creates a generator using the specified format string, with custom formatters for the item value and item percentage. This constructor throws an `IllegalArgumentException` if any of the arguments is `null`.

### 31.24.4 Methods

To get the label for a section:

► public String generateSectionLabel(PieDataset dataset, Comparable key);  
Returns the section label for the data item with the given `key`. The actual string returned depends on the format string and locale specified in the constructor for this class.

### 31.24.5 Attributed Labels

An option is provided to use `AttributedString` instances as the section labels, however there is currently no mechanism in the `PiePlot` class to display these (so, for now, you can just ignored these methods).

Code:	Description:
{0}	The item key.
{1}	The item value.
{2}	The item value as a percentage of the total.

Table 31.5: *MessageFormat substitutions*

```
➔ public AttributedString generateAttributedSectionLabel(PieDataset dataset, Comparable key);
    Returns the attributed label for the section with the given key. This method can return null.
```

The default implementation of the above method just returns fixed strings that are controlled via the following methods:

```
➔ public AttributedString getAttributedLabel(int series);
    Returns the attributed label (possibly null).
➔ public void setAttributedLabel(int series, AttributedString label);
    Sets the attributed label (null is permitted) for the given section.
```

### 31.24.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
    Tests this label generator for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 31.24.7 Notes

In version 1.0.2, the default section label format changed to show only the section name (this change affects default pie plots, among other things).

## 31.25 StandardToolTipGenerator

### 31.25.1 Overview

A label generator that can be used to generate tool tips for a `PiePlot` (this class implements the `PieToolTipGenerator` interface). The generator uses Java's `MessageFormat` class to construct labels by substituting any or all of the objects listed in table 31.5.

The default tool tip format string is "`{0}: ({1}, {2})`", which displays the item key, followed by the item value and percentage.

### 31.25.2 Usage

You can use this class when you want to change the format of the the tool tips on a pie chart. For example:

```
PiePlot plot = (PiePlot) chart.getPlot();
PieToolTipGenerator generator = new StandardToolTipGenerator(
    "{0} = {2}", new DecimalFormat("0"), new DecimalFormat("0.00%"));
plot.setToolTipGenerator(generator);
```

### 31.25.3 Constructors

The default constructor uses number and percentage formatters appropriate for the default locale:

```
➔ public StandardToolTipGenerator();
Equivalent to StandardToolTipGenerator(DEFAULT_TOOLTIP_FORMAT, Locale.getDefault())—see below.
```

You can create a generator with a specific format string:

```
► public StandardPieToolTipGenerator(String labelFormat);
Equivalent to StandardPieToolTipGenerator(labelFormat, Locale.getDefault())—see below.

► public StandardPieToolTipGenerator(Locale locale); [1.0.7]
Equivalent to StandardPieToolTipGenerator(DEFAULT_TOOLTIP_FORMAT, locale)—see below.

► public StandardPieToolTipGenerator(String labelFormat, Locale locale); [1.0.7]
Creates a generator using the specified format string. The item value and percentage (if included
in the format string) will be formatted using default formatters for the specified locale.
```

The final constructor allows you to specify the item value and percentage formatters:

```
► public StandardPieToolTipGenerator(String labelFormat,
NumberFormat numberFormat, NumberFormat percentFormat)
Creates a generator using the specified format string, with custom formatters for the item value
and item percentage.
```

### 31.25.4 Methods

See [AbstractPieItemLabelGenerator](#).

### 31.25.5 Notes

Some points to note:

- instances of this class are cloneable and serializable;
- section 11 contains information about using tool tips with JFreeChart.

#### See Also

[PieToolTipGenerator](#).

## 31.26 StandardXYItemLabelGenerator

### 31.26.1 Overview

A generator that can be assigned to an [XYItemRenderer](#) for the purpose of generating item labels (this class implements the [XYItemLabelGenerator](#) interface). This class is very flexible in the format of the labels it can generate. It uses Java's [MessageFormat](#) class to create a label that can contain any of the items listed in table 31.2. The x and y values can be formatted using any instance of [NumberFormat](#) or [DateFormat](#).

### 31.26.2 Usage

Most often you will assign a generator to a renderer and then never need to refer to it again:

```
XYPlot plot = (XYPlot) chart.getPlot();
XYItemRenderer renderer = plot.getRenderer();
XYItemLabelGenerator generator = new StandardXYItemLabelGenerator(
    "{2}", new DecimalFormat("0.00"), new DecimalFormat("0.00")
);
renderer.setItemLabelGenerator(generator);
renderer.setItemLabelsVisible(true);
```

The renderer will call the generator's methods when necessary. See section 12 for more information.

### 31.26.3 Constructors

To create a default generator:

```
↳ public StandardXYItemLabelGenerator();
```

Creates a new generator that formats values using the default number format for the user's locale. "{2}" is used as the label format string (that is, just the data value).

To create a generator that formats the x and y values as numbers:

```
↳ public StandardXYItemLabelGenerator(String labelFormat, NumberFormat xFormat,
NumberFormat yFormat);
```

Creates a generator that formats values as numbers using the supplied formatters. The `labelFormat` is passed to a `MessageFormat` to control the structure of the generated label, and can use any of the substitutions listed in table 31.2.

To create a generator that formats the x-values as dates (interpreting the numerical value as milliseconds since 1-Jan-1970, in the same way as `java.util.Date`):

```
↳ public StandardXYItemLabelGenerator(String labelFormat, DateFormat xFormat,
NumberFormat yFormat);
```

Creates a generator that formats values as dates using the supplied formatter. The `labelFormat` is passed to a `MessageFormat` to control the structure of the generated label, and can use any of the substitutions listed in table 31.2.

```
↳ public StandardXYItemLabelGenerator(String formatString, NumberFormat xFormat,
DateFormat yFormat); [1.0.4]
```

Creates a generator that formats the x-values as numbers and the y-values as dates.

```
↳ public StandardXYItemLabelGenerator(String formatString, DateFormat xFormat,
DateFormat yFormat);
```

Creates a generator that formats both the x and y-values as dates.

### 31.26.4 Methods

The renderer will call the following method whenever it requires an item label:

```
↳ public String generateLabel(XYDataset dataset, int series, int item);
```

Generates an item label for the specified data item.

### 31.26.5 Notes

Some points to note:

- instances of this class are cloneable and serializable, and the `PublicCloneable` interface is implemented;

#### See Also

[AbstractXYItemLabelGenerator](#), [StandardXYToolTipGenerator](#).

## 31.27 StandardXYSeriesLabelGenerator

### 31.27.1 Overview

A generator that can be assigned to an `XYItemRenderer` for the purpose of generating series labels for the legend (this class implements the `XYSeriesLabelGenerator` interface). This class is very flexible in the format of the labels it can generate. It uses Java's `MessageFormat` class to create a label, with {0} being substituted with the series name.

### 31.27.2 Constructors

To create a default generator:

```
➔ public StandardXYSeriesLabelGenerator();
```

Creates a new generator that formats values with "{0}" used as the label format string (that is, just the series name).

To create a generator with a custom format string:

```
➔ public StandardXYLabelGenerator(String labelFormat);
```

Creates a generator that formats the series label with the given format string. The `labelFormat` is passed to a `MessageFormat` to control the structure of the generated label, with {0} being substituted with the series name.

### 31.27.3 Methods

The renderer will call the following method whenever it requires an item label:

```
➔ public String generateLabel(XYDataset dataset, int series);
```

Generates a label for the specified series.

### 31.27.4 Notes

Some points to note:

- instances of this class are cloneable and serializable, and the `PublicCloneable` interface is implemented;

## 31.28 StandardXYToolTipGenerator

### 31.28.1 Overview

A generator that can be assigned to an `XYItemRenderer` for the purpose of generating tooltips (this class implements the `XYToolTipGenerator` interface). This class is very flexible in the format of the labels it can generate. It uses Java's `MessageFormat` class to create a label that can contain any of the items listed in table 31.2. The x and y values can be formatted using any instance of `NumberFormat` or `DateFormat`.

### 31.28.2 Usage

You can create a tool tip generator and assign it to a renderer when you wish to control the formatting of the tool tip text. For example:

```
XYPlot plot = (XYPlot) chart.getPlot();
XYItemRenderer renderer = plot.getRenderer();
XYToolTipGenerator generator = new StandardXYToolTipGenerator(
    "{2}", new DecimalFormat("0.00"), new DecimalFormat("0.00")
);
renderer.setToolTipGenerator(generator);
```

The renderer will call the generator's methods when necessary. See section 11 for more information.

For the display of time series data, you will want the x-values to be formatted as dates in the tool tips. You can achieve this by specifying a `DateFormat` instance as the formatter for the x-values, as follows:

```
XYPlot plot = (XYPlot) chart.getPlot();
XYItemRenderer renderer = plot.getRenderer();
XYToolTipGenerator generator = new StandardXYToolTipGenerator(
    "{1}, {2}", new SimpleDateFormat("d-MMM-yyyy"), new DecimalFormat("0.00")
);
renderer.setToolTipGenerator(generator);
```

### 31.28.3 Constructors

To create a default generator:

```
➔ public StandardXYToolTipGenerator();
```

Creates a new generator that formats values using the default number format for the user's locale. "{0}: ({1}, {2})" is used as the label format string (that is, the series name followed by the x and y values).

To create a generator that formats the x and y values as numbers:

```
➔ public StandardXYToolTipGenerator(String labelFormat, NumberFormat xFormat,
NumberFormat yFormat);
```

Creates a generator that formats values as numbers using the supplied formatters. The `labelFormat` is passed to a `MessageFormat` to control the structure of the generated label, and can use any of the substitutions listed in table 31.2.

To create a generator that formats the x-values as dates (interpreting the numerical value as milliseconds since 1-Jan-1970, in the same way as `java.util.Date`):

```
➔ public StandardXYToolTipGenerator(String labelFormat, DateFormat xFormat,
NumberFormat yFormat);
```

Creates a generator that formats values as dates using the supplied formatter. The `labelFormat` is passed to a `MessageFormat` to control the structure of the generated label, and can use any of the substitutions listed in table 31.2.

```
➔ public StandardXYToolTipGenerator(String formatString, NumberFormat xFormat,
DateFormat yFormat); [1.0.4]
```

Creates a generator that formats the x-values as numbers and the y-values as dates.

```
➔ public StandardXYToolTipGenerator(String formatString, DateFormat xFormat,
DateFormat yFormat);
```

Creates a generator that formats both the x and y-values as dates.

### 31.28.4 Methods

The renderer will call the following method whenever it requires an item label:

```
➔ public String generateToolTip(XYDataset dataset, int series, int item);
```

Generates a tool tip for the specified data item.

### 31.28.5 Notes

Some points to note:

- instances of this class are cloneable and serializable, and the `PublicCloneable` interface is implemented;

#### See Also

[AbstractXYItemLabelGenerator](#), [StandardXYItemLabelGenerator](#).

## 31.29 StandardXYZToolTipGenerator

### 31.29.1 Overview

A default implementation of the `XYZToolTipGenerator` interface. This generator is used with the `XYBubbleRenderer` class.

## 31.30 SymbolicXYItemLabelGenerator

### 31.30.1 Overview

An item label generator for use with symbolic plots.

## 31.31 XYItemLabelGenerator

### 31.31.1 Overview

This interface defines the API for an *item label generator*, which is used by a renderer to obtain labels for the items in a dataset. A generator is assigned to a renderer using the `setItemLabelGenerator()` method in the `XYItemRenderer` interface.

Classes that implement this interface include:

- `StandardXYItemLabelGenerator`;
- `SymbolicXYItemLabelGenerator`;
- `BubbleXYItemLabelGenerator`;
- `HighLowItemLabelGenerator`.

### 31.31.2 Usage

Chapter 12 contains information about using item labels.

### 31.31.3 Methods

The renderer will call the following method whenever it requires an item label:

```
➔ public String generateLabel(XYDataset dataset, int series, int item);
```

Returns a string that will be used to label the specified data item. Classes that implement this method are permitted to return `null` for the result, in which case no label will be displayed for that item.

### 31.31.4 Notes

Some points to note:

- the `StandardXYItemLabelGenerator` class provides a very flexible implementation of this interface, which should meet most requirements. However, you can also write your own class that implements this interface, and take complete control over the generation of item labels;

## 31.32 XYSeriesLabelGenerator

### 31.32.1 Overview

An *xy series label generator* is an object that assumes responsibility for generating the text strings that will be used for the series labels in a chart's legend. A generator is assigned to a renderer using the `setLegendItemLabelGenerator()` method in the `XYItemRenderer` interface.

Classes that implement this interface include:

- `StandardXYSeriesLabelGenerator`;
- `MultipleXYSeriesLabelGenerator`.

### 31.32.2 Methods

The renderer will call the following method whenever it requires a series label for the legend:

```
➔ public String generateLabel(XYDataset dataset, int series);  
    Returns a string that will be used to label the specified data series.
```

#### See Also

[StandardXYSeriesLabelGenerator](#).

## 31.33 XYToolTipGenerator

### 31.33.1 Overview

The interface that must be implemented by an *XY tool tip generator*, a class used to generate tool tips for an [XYPlot](#).

### 31.33.2 Methods

The plot will call the following method whenever it requires a tool tip for an item:

```
➔ public String generateToolTip(XYDataset dataset, int series, int item);  
    This method is called whenever the plot needs to generate a tooltip for a data item. It can  
    return an arbitrary string, generally derived from the specified item in the supplied dataset.3
```

### 31.33.3 Notes

Some points to note:

- to “install” a tool tip generator, use the `setToolTipGenerator()` method in the [XYItemRenderer](#) interface.
- [StandardXYToolTipGenerator](#) implements this interface, but you are free to write your own implementation to suit your requirements.

Section 11 contains information about using tool tips with JFreeChart.

## 31.34 XYZToolTipGenerator

### 31.34.1 Overview

A tool tip generator that creates labels for items in an [XYZDataset](#).

### 31.34.2 Methods

This interface adds a single method to the one it inherits from [XYToolTipGenerator](#):

```
➔ public String generateToolTip(XYZDataset dataset, int series, int item);  
    Returns a (possibly null) string as the tool tip text for the specified item within a given series.
```

---

<sup>3</sup>If you are implementing this function with a look up table for tool tips, be aware that some [XYDataset](#) implementations can reorder the data items, so you will need to ensure that your lookup table is kept in sync with the dataset.

### 31.34.3 Notes

Some points to note:

- this interface extends [XYToolTipGenerator](#);
- the [StandardXYZToolTipGenerator](#) class is the only implementation of this interface provided by JFreeChart.

# Chapter 32

## Package: org.jfree.chart.needle

### 32.1 Overview

This package contains classes for drawing needles in a [CompassPlot](#):

- [ArrowNeedle](#) – an arrow needle;
- [LineNeedle](#) – a line needle;
- [LongNeedle](#) – a long needle;
- [PinNeedle](#) – a pin needle;
- [PlumNeedle](#) – a plum needle;
- [PointerNeedle](#) – a pointer needle;
- [ShipNeedle](#) – a ship needle;
- [WindNeedle](#) – a wind needle;

### 32.2 ArrowNeedle

#### 32.2.1 Overview

A class that draws an *arrow needle* for the [CompassPlot](#) class (see figure 32.1).

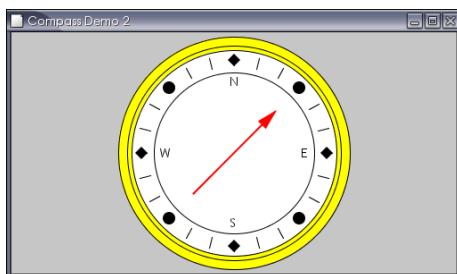


Figure 32.1: An arrow needle

## 32.3 LineNeedle

### 32.3.1 Overview

A class that draws a *line needle* for the `CompassPlot` class (see figure 32.2).

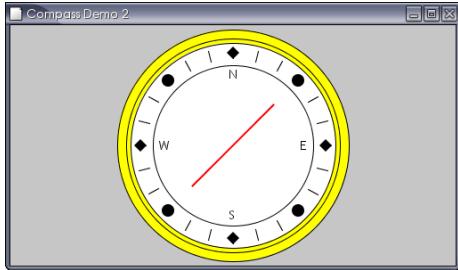


Figure 32.2: A line needle

## 32.4 LongNeedle

### 32.4.1 Overview

A class that draws a *long needle* for the `CompassPlot` class (see figure 32.3).

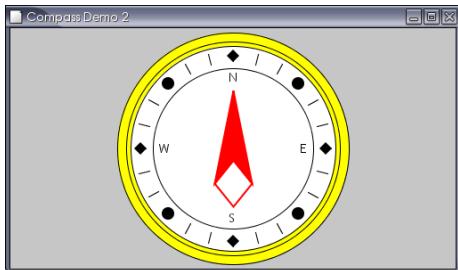


Figure 32.3: A long needle

## 32.5 MeterNeedle

### 32.5.1 Overview

A base class that draws a needle for the `CompassPlot` class. A range of different subclasses implement different types of needles:

- `ArrowNeedle` – an arrow needle;
- `LineNeedle` – a line needle;
- `LongNeedle` – a long needle;
- `PinNeedle` – a pin needle;
- `PlumNeedle` – a plum needle;
- `PointerNeedle` – a pointer needle;

- [ShipNeedle](#) – a ship needle;
- [WindNeedle](#) – a wind needle;

## 32.6 PinNeedle

### 32.6.1 Overview

A class that draws a *pin needle* for the [CompassPlot](#) class (see figure 32.4).

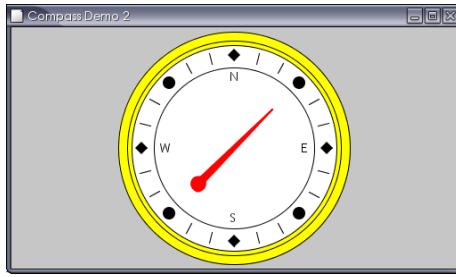


Figure 32.4: A pin needle

## 32.7 PlumNeedle

### 32.7.1 Overview

A class that draws a *plum needle* for the [CompassPlot](#) class (see figure 32.5).

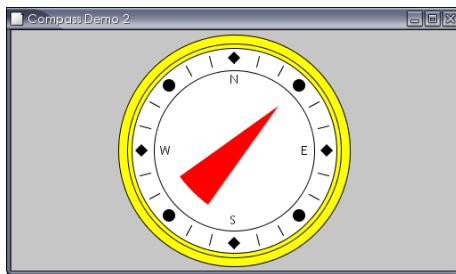


Figure 32.5: A plum needle

## 32.8 PointerNeedle

### 32.8.1 Overview

A class that draws a *pointer needle* for the `CompassPlot` class (see figure 32.6).

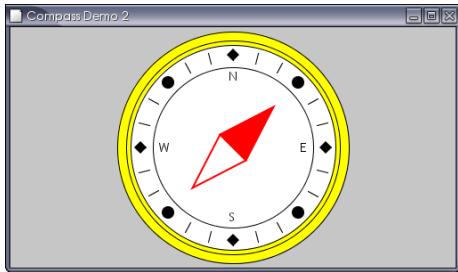


Figure 32.6: A pointer needle

## 32.9 ShipNeedle

### 32.9.1 Overview

A class that draws a *ship needle* for the `CompassPlot` class (see figure 32.7).

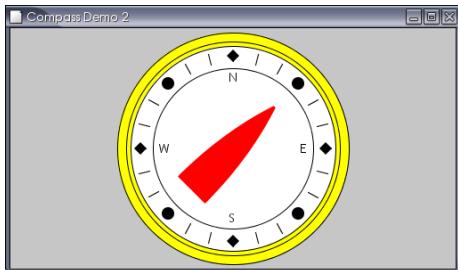


Figure 32.7: A ship needle

## 32.10 WindNeedle

### 32.10.1 Overview

A class that draws a *wind needle* for the `CompassPlot` class (see figure 32.8).

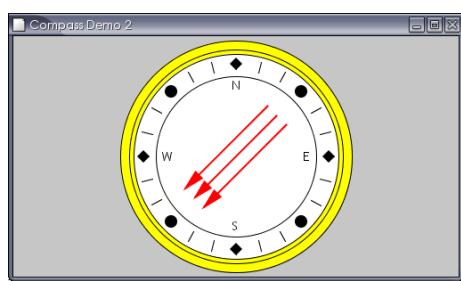


Figure 32.8: A wind needle

# Chapter 33

## Package: org.jfree.chart.plot

### 33.1 Overview

The `org.jfree.chart.plot` package contains:

- the `Plot` base class;
- a range of plot subclasses, including `PiePlot`, `CategoryPlot` and `XYPlot`;
- various support classes and interfaces.

This is an important package, because the `Plot` classes play a key role in controlling the presentation of data with JFreeChart.

### 33.2 CategoryMarker

#### 33.2.1 Overview

A marker that can be used to highlight a category in a `CategoryPlot`. The marker can be drawn as a line (see figure 33.1) or as a rectangle (see figure 33.2). Markers are added to the plot using the `addDomainMarker()` methods in the `CategoryPlot` class.

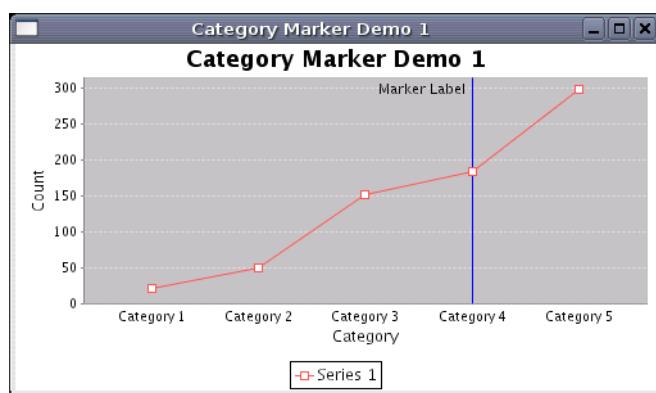


Figure 33.1: A `CategoryMarker` drawn as a line

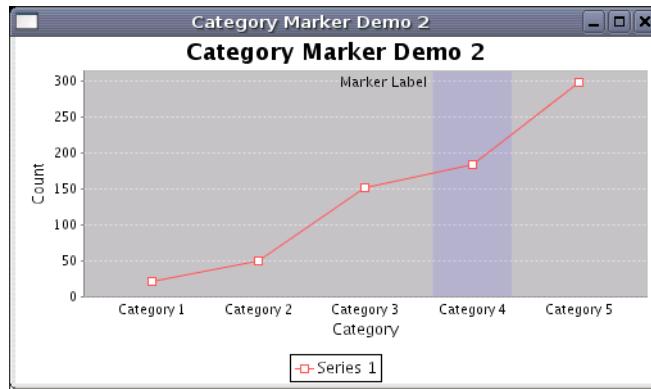


Figure 33.2: A `CategoryMarker` drawn as a rectangle

### 33.2.2 Constructors

To create a new marker, use one of the following constructors:

- `public CategoryMarker(Comparable key);`  
Creates a marker for the category with the specified `key`, using `Color.gray` for the paint and `new BasicStroke(1.0f)` for the stroke.
- `public CategoryMarker(Comparable key, Paint paint, Stroke stroke);`  
Creates a marker for the category with the specified `key`, using the specified `paint` and `stroke`.
- `public CategoryMarker(Comparable key, Paint paint, Stroke stroke, Paint outlinePaint, Stroke outlineStroke, float alpha);`  
Creates a marker for the category with the specified `key`, using the specified `paint` and `stroke`.  
The `alpha` value controls the transparency (0.0 is transparent, 1.0 is opaque).

### 33.2.3 Methods

To get the key that links the marker to a category:

- `public Comparable getKey();`  
Returns the key for the marker.
- `public void setKey(Comparable key); [1.0.3]`  
Sets the key for the marker and sends a `MarkerChangeEvent` to all registered listeners.

A flag controls whether the marker is drawn as a thin line in the center of the category or a rectangle covering the whole width of the category:

- `public boolean getDrawAsLine();`  
Returns `true` if the marker should be drawn as a thin line in the middle of the category, and `false` if the marker should be drawn as a rectangle covering the full width of the category.
- `public void setDrawAsLine(boolean drawAsLine);`  
Sets the flag that controls whether the marker is drawn as a line or a rectangle, and sends a `MarkerChangeEvent` to all registered listeners.

Other methods are inherited from the `Marker` class.

### 33.2.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests the marker for equality with an arbitrary object. This method returns `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `CategoryMarker`;
- `obj` has the same field values as this marker.

Instances of this class are `Cloneable` and `Serializable`.

### 33.2.5 Notes

Some points to note:

- markers are drawn by the `drawDomainMarker()` method in the `AbstractCategoryItemRenderer` class;
- `CategoryMarker` is a subclass of `Marker`;
- demos (`CategoryMarkerDemo1` and `CategoryMarkerDemo2`) illustrating the use of this class are included in the [JFreeChart demo collection](#).

## 33.3 CategoryPlot

### 33.3.1 Overview

A general plotting class that is most commonly used to display bar charts, but also supports line charts, area charts, stacked area charts and more. A *category plot* has:

- one or more *domain axes* (instances of `CategoryAxis`);
- one or more *range axes* (instances of `ValueAxis`);
- one or more *datasets* (these can be instances of any class that implements the `CategoryDataset` interface);
- one or more *renderers* (these can be instances of any class that implements the `CategoryItemRenderer` interface);

The plot can be displayed with a horizontal or vertical orientation (see the `PlotOrientation` class).

### 33.3.2 Attributes

The attributes maintained by the `CategoryPlot` class, which are in addition to those inherited from the `Plot` class, are listed in Table 33.1.

### 33.3.3 Plot Orientation

A `CategoryPlot` can be drawn with one of two orientations:

- *horizontal orientation* – the domain (category) axis will appear at the left or right of the chart, and the range (value) axis will appear at the top or bottom of the chart;
- *vertical orientation* – the domain (category) axis will appear at the top or bottom of the chart and the range (value) axis will appear at the left or right of the chart.

The default orientation is `PlotOrientation.VERTICAL`. To change the plot's orientation, use the following code:

```
plot.setOrientation(PlotOrientation.HORIZONTAL);
```

Note that calling this method will trigger a `PlotChangeEvent` that will result in the chart being redrawn if it is being displayed in a `ChartPanel`.

Attribute:	Description:
<i>orientation</i>	The plot orientation (horizontal or vertical).
<i>axisOffset</i>	The offset between the data area and the axes.
<i>domainAxes</i>	The domain axes (used to display categories).
<i>domainAxisLocations</i>	The locations of the domain axes.
<i>rangeAxes</i>	The range axes (used to display values).
<i>rangeAxisLocations</i>	The locations of the range axes.
<i>datasets</i>	The dataset(s).
<i>renderers</i>	The plot's renderers ("pluggable" objects responsible for drawing individual data items within the plot).
<i>renderingOrder</i>	The order for rendering data items (see <a href="#">DatasetRenderingOrder</a> ).
<i>columnRenderingOrder</i>	Controls the column order in which the data items are rendered.
<i>rowRenderingOrder</i>	Controls the row order in which the data items are rendered.
<i>domainGridlinesVisible</i>	A flag that controls whether gridlines are drawn against the domain axis.
<i>domainGridlinePosition</i>	The position of the gridlines against the domain axis.
<i>domainGridlinePaint</i>	The paint used to draw the domain gridlines.
<i>domainGridlineStroke</i>	The stroke used to draw the domain gridlines.
<i>rangeGridlinesVisible</i>	A flag that controls whether gridlines are drawn against the range axis.
<i>rangeGridlinePaint</i>	The paint used to draw the range gridlines.
<i>rangeGridlineStroke</i>	The stroke used to draw the range gridlines.
<i>foregroundRangeMarkers</i>	A list of markers (constants) to be highlighted on the plot.
<i>backgroundRangeMarkers</i>	A list of markers (constants) to be highlighted on the plot.
<i>weight</i>	The weight for the plot (only used when the plot is a subplot).
<i>fixedDomainAxisSpace</i>	Specifies a fixed amount of space to allocate to the domain axis ( <code>null</code> permitted).
<i>fixedRangeAxisSpace</i>	Specifies a fixed amount of space to allocate to the range axis ( <code>null</code> permitted).

Table 33.1: Attributes for the `CategoryPlot` class

### 33.3.4 Axes

A `CategoryPlot` usually has a single domain axis (an instance of the `CategoryAxis` class) and a single range axis (an instance of the `ValueAxis` class). You can obtain a reference to the primary domain axis with:

```
CategoryAxis domainAxis = plot.getDomainAxis();
```

Similarly, you can obtain a reference to the primary range axis with:

```
ValueAxis rangeAxis = plot.getRangeAxis();
```

The `CategoryPlot` class also has support for multiple axes. You can obtain a reference to any secondary domain axis by specifying the axis index:

```
CategoryAxis domainAxis2 = plot.getDomainAxis(1);
```

Similarly, you can obtain a reference to any secondary range axis by specifying the axis index:

```
ValueAxis rangeAxis2 = plot.getRangeAxis(1);
```

To find the index for an axis:

► `public int getDomainAxisIndex(CategoryAxis axis); [1.0.3]`

Returns the index for the specified axis, or -1 if the axis does not belong to the plot. If `axis` is `null`, this method throws an `IllegalArgumentException`.

► `public int getRangeAxisIndex(ValueAxis axis); [1.0.7]`

Returns the index for the specified axis, or -1 if the axis does not belong to the plot. If `axis` is `null`, this method throws an `IllegalArgumentException`.

The axis classes have many attributes that can be customised to control the appearance of your charts.

### 33.3.5 Axis Offsets

The axes can be offset slightly from the edges of the plot area, if required:

► public `RectangleInsets` `getAxisOffset()`;

Returns the object that controls the offset between the plot area and the axes. The default value is `RectangleInsets.ZERO_INSETS`.

► public void `setAxisOffset(RectangleInsets offset)`;

Sets the object that controls the offset between the plot area and the axes, and sends a `PlotChangeEvent` to all registered listeners. If `offset` is `null`, this method throws an `IllegalArgumentException`.

### 33.3.6 Datasets and Renderers

A `CategoryPlot` can have zero, one or many datasets and each dataset is usually associated with a renderer (the object that is responsible for drawing the visual representation of each item in a dataset). A dataset is an instance of any class that implements the `CategoryDataset` interface and a renderer is an instance of any class that implements the `CategoryItemRenderer` interface.

To get/set a dataset:

► public `CategoryDataset` `getDataset(int index)`;

Returns the dataset at the specified index (possibly `null`).

► public void `setDataset(int index, CategoryDataset dataset)`;

Assigns a dataset to the plot. The new dataset replaces any existing dataset at the specified index. It is permitted to set a dataset to `null` (in that case, no data will be displayed on the chart).

To get/set a renderer:

► public `CategoryItemRenderer` `getRenderer(int index)`;

Returns the renderer at the specified index (possibly `null`).

► public void `setRenderer(int index, CategoryItemRenderer renderer)`;

Sets the renderer at the specified index and sends a `PlotChangeEvent` to all registered listeners. It is permitted to set any renderer to `null`.

### 33.3.7 Dataset Rendering Order

When a plot has more than one dataset, the order in which the datasets are rendered can have an impact on the final appearance of the chart. For example, if one dataset is represented using bars, and the other is represented using lines, you'll normally want the lines to be drawn in front of the bars. By default, the datasets are drawn in reverse order, so that the first dataset you add appears at the front of the chart.

You can control the rendering order using the following methods:

► public `DatasetRenderingOrder` `getDatasetRenderingOrder()`;

Returns the current dataset rendering order (never `null`). The default is `DatasetRenderingOrder.REVERSE`.

► public void `setDatasetRenderingOrder(DatasetRenderingOrder order)`;

Sets the dataset rendering order and sends a `PlotChangeEvent` to all registered listeners. It is not permitted to set the rendering order to `null`.

By default, datasets will be rendered in reverse order so that the first dataset added to the plot appears to be in front of the other datasets.

### 33.3.8 Item Rendering Order

Within each dataset, the data items are rendered by column then by row, in ascending order (by default). In some cases, you might need to change the order in which the items are rendered (it only matters when the renderer draws items in such a way that they can overlap with other items, for example the bars drawn by a `BarRenderer3D`):

```

→ public SortOrder getColumnRenderingOrder();
Returns the column rendering order (the default is SortOrder.ASCENDING). This method never returns null.

→ public void setColumnRenderingOrder(SortOrder order);
Sets the column rendering order and sends a PlotChangeEvent to all registered listeners. This method throws an IllegalArgumentException if order is null.

→ public SortOrder getRowRenderingOrder();
Returns the row rendering order (the default is SortOrder.ASCENDING). This method never returns null.

→ public void setRowRenderingOrder(SortOrder order);
Sets the row rendering order and sends a PlotChangeEvent to all registered listeners. This method throws an IllegalArgumentException if order is null.

```

Note that the above methods *do not* change the position of the items, only the order in which they get drawn.

### 33.3.9 Series Colors

The colors used for the series within the chart are controlled by the plot's *renderer(s)*. You can obtain a reference to the primary renderer and set the series colors using code similar to the following:

```

CategoryPlot plot = (CategoryPlot) chart.getPlot();
CategoryItemRenderer renderer = plot.getRenderer();
renderer.setSeriesPaint(0, new Color(0, 0, 255));
renderer.setSeriesPaint(1, new Color(75, 75, 255));
renderer.setSeriesPaint(2, new Color(150, 150, 255));

```

### 33.3.10 Gridlines

By default, the `CategoryPlot` class will display gridlines against the (primary) range axis, but not the domain axis. However, it is simple to override the default behaviour:

```

CategoryPlot plot = (CategoryPlot) chart.getPlot();
plot.setDomainGridlinesVisible(true);
plot.setRangeGridlinesVisible(true);

```

Note that the domain and range gridlines are controlled independently.

### 33.3.11 Legend Items

The items that appear in the legend for a chart are obtained by a call to the following method *at the time the chart is being drawn*:

```

→ public LegendItemCollection getLegendItems();
Returns the collection of legend items that should be displayed in the legend for this plot.
Note that a new collection is returned each time this method is called—modifying the returned collection has no effect on the legend displayed by the chart.

```

By default, this method will return a collection that contains one item for each series in the dataset(s) belonging to the plot. If this is not the behaviour you require, there are a couple of options for altering the items that will appear in the chart's legend.

First, you can specify a “fixed” set of legend items that will always be displayed, regardless of the contents of the dataset(s):

```

→ public void setFixedLegendItems(LegendItemCollection items);
Sets a “fixed” collection of legend items that will always be used for this plot regardless of the contents of the dataset(s) belonging to the plot. Set this to null if you wish to revert to the default behaviour.

```

A second, but more complex, approach involves subclassing `CategoryPlot` and overriding the `getLegendItems()` method. This gives you complete control over the legend items included for your plot.

### 33.3.12 Fixed Axis Dimensions

The width and height of the axes are normally determined by JFreeChart to allow just the required amount of space, no more and no less. Occasionally, you may want to override this behaviour and specify a fixed amount of space to allocate to each axis. As an example, this can make it easier to align the contents of multiple charts.

```
→ public AxisSpace getFixedDomainAxisSpace();
Returns the fixed dimensions for the domain axis (possibly null).

→ public void setFixedDomainAxisSpace(AxisSpace space);
Sets the fixed dimensions for the domain axis. Set this to null if you prefer JFreeChart to determine this dynamically (the default behaviour).

→ public AxisSpace getFixedRangeAxisSpace();
Returns the fixed dimensions for the range axis (possibly null).

→ public void setFixedRangeAxisSpace(AxisSpace space);
Sets the fixed dimensions for the range axis. Set this to null if you prefer JFreeChart to determine this dynamically (the default behaviour).
```

### 33.3.13 Crosshair

This plot has support for a crosshair against the primary range axis. The following flag controls whether or not the plot displays the crosshair:

```
→ public boolean isRangeCrosshairVisible();
Returns the flag that controls whether or not the plot displays a crosshair against the range axis. The default value is false.

→ public void setRangeCrosshairVisible(boolean flag);
Sets the flag that controls whether or not the plot displays a crosshair against the range axis, and sends a PlotChangeEvent to all registered listeners.
```

The crosshair value is updated when the user clicks on a [ChartPanel](#) that displays this plot. A flag controls whether the crosshair is set to the value corresponding to the mouse click, or the nearest actual data value:

```
→ public boolean isRangeCrosshairLockedOnData();
Returns the flag that controls whether or not the crosshair point locks onto the nearest data value. The default value is true.

→ public void setRangeCrosshairLockedOnData(boolean flag);
Sets the flag that controls whether or not the crosshair point locks onto the nearest data value, and sends a PlotChangeEvent to all registered listeners.
```

To control the current crosshair value:

```
→ public double getRangeCrosshairValue();
Returns the current crosshair value. Note that after a mouse click, this value will change during the chart repaint—to read the latest value, you need to wait until the chart has finished repainting.

→ public void setRangeCrosshairValue(double value);
Equivalent to setRangeCrosshairValue(value, true)—see the next method.

→ public void setRangeCrosshairValue(double value, boolean notify);
Sets the crosshair value and, if requested, sends a PlotChangeEvent to all registered listeners.
```

To control the paint and stroke used to draw the crosshair line:

```
→ public Stroke getRangeCrosshairStroke();
Returns the stroke used to draw the crosshair. The default stroke is a thin dashed line. This method never returns null.

→ public void setRangeCrosshairStroke(Stroke stroke);
Sets the stroke used to draw the crosshair and sends a PlotChangeEvent to all registered listeners.
```

► `public Paint getRangeCrosshairPaint();`  
 Returns the paint used to draw the crosshair. The default value is `Color.blue`. This method never returns `null`.

► `public void setRangeCrosshairPaint(Paint paint);`  
 Sets the paint used to draw the crosshair and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

### 33.3.14 Methods

A zoom method is provided to support the zooming function provided by the `ChartPanel` class:

► `public void zoom(double percent);`  
 Increases or decreases the axis range (about the anchor value) by the specified percentage. If the percentage is zero, then the auto-range calculation is restored for the value axis.

The category axis remains fixed during zooming, only the value axis changes.

To add a range marker to a plot:

► `public void addRangeMarker(Marker marker);`  
 Adds a marker which will be drawn against the range axis.

To add an annotation to a plot:

► `public void addAnnotation(CategoryAnnotation annotation);`  
 Adds an annotation to the plot.

To set the weight for a plot:

► `public void setWeight(int weight);`  
 Sets the weight for a plot. This is used to determine how much space is allocated to the plot when it is used as a subplot within a combined plot.

### 33.3.15 Draw Method

The following method is called by the `JFreeChart` class during chart drawing:

► `public void draw(Graphics2D g2, Rectangle2D plotArea,  
 Point2D anchor, PlotState parentState, PlotRenderingInfo state);`  
 Draws the plot within the specified area.

In typical situations, you won't normally call this method directly.

### 33.3.16 Notes

A number of `CategoryItemRenderer` implementations are included in the JFreeChart distribution.

#### See Also

[CombinedDomainCategoryPlot](#), [CombinedRangeCategoryPlot](#).

## 33.4 ColorPalette

### 33.4.1 Overview

The abstract base class for the color palettes used by the `ContourPlot` class. *This class is deprecated as of version 1.0.4.*

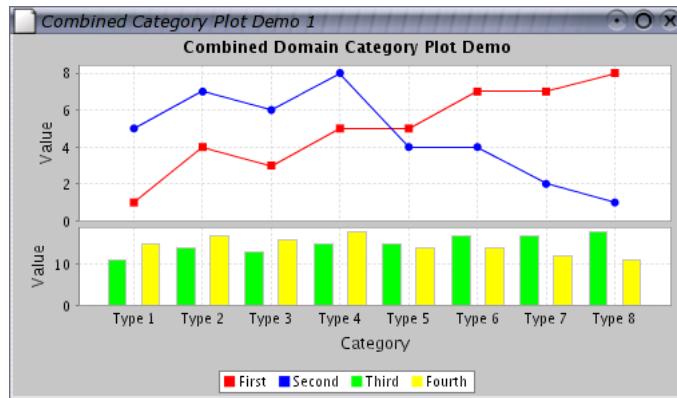


Figure 33.3: A *CombinedDomainCategoryPlot*

## 33.5 CombinedDomainCategoryPlot

### 33.5.1 Overview

A *category plot* that allows multiple subplots to be displayed together using a shared domain axis—see figure 33.3 for an example.

### 33.5.2 Constructors

To create a new parent plot:

```
→ public CombinedDomainCategoryPlot();
Creates a new parent plot that uses a default CategoryAxis for the shared domain axis.

→ public CombinedDomainCategoryPlot(CategoryAxis domainAxis);
Creates a new parent plot with the specified domain axis (null not permitted).
```

After creating a new parent plot, you need to add some subplots.

### 33.5.3 Adding and Removing Subplots

To add a subplot to a combined plot:

```
→ public void add(CategoryPlot subplot);
Adds a subplot to the combined plot, with a weight of 1, and sends a PlotChangeEvent to all registered listeners. Adding a null subplot is not permitted. The subplot's domain axis will be set to null. You must ensure that the subplot has a non-null range axis. See the following method for a description of the weight.

→ public void add(CategoryPlot subplot, int weight);
Adds a subplot to the combined plot, with the specified weight, and sends a PlotChangeEvent to all registered listeners. Adding a null subplot is not permitted. The subplot's domain axis will be set to null. You must ensure that the subplot has a non-null range axis.
```

The weight determines how much of the plot area is assigned to the subplot. For example, if you add three subplots with weights of 1, 2 and 4, the relative amount of space assigned to each plot is  $1/7$ ,  $2/7$  and  $4/7$  (where the 7 is the sum of the individual weights).

To remove a subplot:

```
→ public void remove(CategoryPlot subplot);
Removes the specified subplot and sends a PlotChangeEvent to all registered listeners.
```

To get a list of the subplots:

```
→ public List getSubplots();
Returns an unmodifiable list of the subplots.
```

### 33.5.4 Draw Method

The following method is called by the `JFreeChart` class during chart drawing:

```
➔ public void draw(Graphics2D g2, Rectangle2D plotArea,
    Point2D anchor, PlotState parentState, PlotRenderingInfo state);
    Draws the plot within the specified area.
```

In typical situations, you won't normally call this method directly.

### 33.5.5 Notes

The `CombinedCategoryPlotDemo1.java` file (included in the JFreeChart demo collection) provides an example of this type of plot.

#### See Also

[CombinedRangeCategoryPlot](#).

## 33.6 CombinedDomainXYPlot

### 33.6.1 Overview

A subclass of `XYPlot` that allows you to combine multiple plots on one chart, where the subplots share the domain axis, and maintain their own range axes.

Figure 33.4 illustrates the relationship between the `CombinedDomainXYPlot` and its subplots.

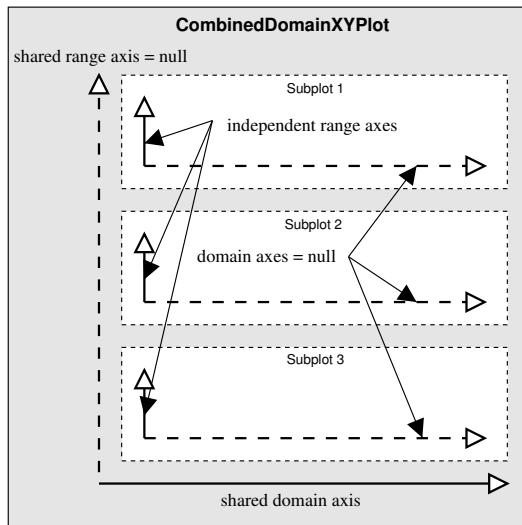


Figure 33.4: `CombinedDomainXYPlot` axes

The `CombinedXYPlotDemo1` class (included in the JFreeChart demo collection) provides an example of this type of plot.

### 33.6.2 Constructors

The default constructor creates a plot with no subplots (initially) and a `NumberAxis` for the shared domain axis:

```
➔ public CombinedDomainXYPlot();
    Creates a new parent plot.
```

More commonly, you will supply the shared domain axis:

```
► public CombinedDomainXYPlot(ValueAxis domainAxis);
Creates a new parent plot using the specified domainAxis (null permitted).
```

After creating the parent plot, you need to add subplots.

### 33.6.3 Methods

To add a subplot to a combined plot:

```
► public void add(XYPlot subplot);
Adds a subplot to the combined plot, with a weight of 1, and sends a PlotChangeEvent to all registered listeners. Adding a null subplot is not permitted. The subplot's domain axis will be set to null. You must ensure that the subplot has a non-null range axis. See the next method for a description of the weight.
```

```
► public void add(XYPlot subplot, int weight);
Adds a subplot to the combined plot, with the specified weight, and sends a PlotChangeEvent to all registered listeners. Adding a null subplot is not permitted. The subplot's domain axis will be set to null. You must ensure that the subplot has a non-null range axis.
```

The weight determines how much of the plot area is assigned to the subplot. For example, if you add three subplots with weights of 1, 2 and 4, the relative amount of space assigned to each plot is 1/7, 2/7 and 4/7 (where the 7 is the sum of the individual weights).

To remove a subplot:

```
► public void remove(XYPlot subplot);
Removes the specified subplot and sends a PlotChangeEvent to all registered listeners.
```

### 33.6.4 The Plot Orientation

To set the plot orientation:

```
► public void setOrientation(PlotOrientation orientation);
Sets the orientation of this plot and all its subplots.
```

### 33.6.5 The Gap Between Subplots

To control the amount of space between the subplots:

```
► public double getGap();
Returns the gap between subplots, in Java2D units.
```

```
► public void setGap(double gap);
Sets the gap (in points) between the subplots and sends a PlotChangeEvent to all registered listeners.
```

### 33.6.6 Notes

Some points to note:

- the dataset for this class should be set to null (only the subplots display data);
- the subplots managed by this class should have one axis set to null (the shared axis is maintained by this class);
- you do not need to set a renderer for the plot, since each subplot maintains its own renderer;
- a demonstration of this type of plot is described in section ??.

**See Also**[XYPlot](#).

## 33.7 CombinedRangeCategoryPlot

### 33.7.1 Overview

A *category plot* that allows multiple subplots to be displayed together using a shared range axis—see figure 33.5 for an example.

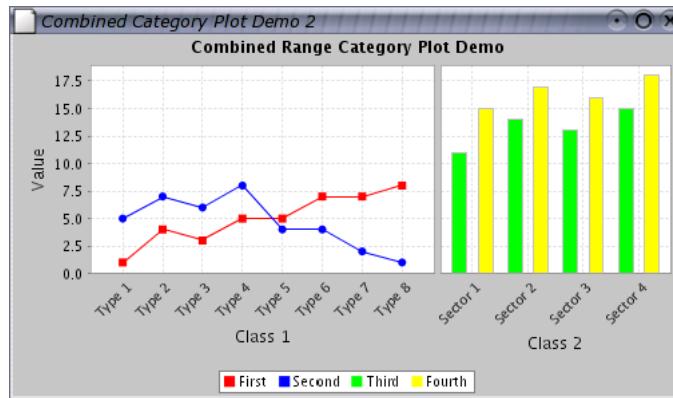


Figure 33.5: A CombinedRangeCategoryPlot

### 33.7.2 Constructors

To create a new parent plot:

```
→ public CombinedRangeCategoryPlot();
Creates a new parent plot that uses a default NumberAxis for the shared range axis.

→ public CombinedRangeCategoryPlot(ValueAxis rangeAxis);
Creates a new parent plot with the specified range axis (null not permitted).
```

After creating a new parent plot, you need to add some subplots.

### 33.7.3 Adding and Removing Subplots

To add a subplot to a combined plot:

```
→ public void add(CategoryPlot subplot);
Adds a subplot to the combined plot, with a weight of 1, and sends a PlotChangeEvent to all registered listeners. Adding a null subplot is not permitted. You must ensure that the subplot's domain axis is not null. The subplot's range axis will be set to null.

→ public void add(CategoryPlot subplot, int weight);
Adds a subplot to the combined plot, with the specified weight, and sends a PlotChangeEvent to all registered listeners. Adding a null subplot is not permitted. You must ensure that the subplot's domain axis is not null. The subplot's range axis will be set to null.
```

The weight determines how much of the plot area is assigned to the subplot. For example, if you add three subplots with weights of 1, 2 and 4, the relative amount of space assigned to each plot is 1/7, 2/7 and 4/7 (where the 7 is the sum of the individual weights).

To remove a subplot:

```
→ public void remove(CategoryPlot subplot);
    Removes the specified subplot and sends a PlotChangeEvent to all registered listeners.
```

To get a list of the subplots:

```
→ public List<CategoryPlot> getSubplots();
    Returns an unmodifiable list of the subplots.
```

### 33.7.4 Draw Method

The following method is called by the `JFreeChart` class during chart drawing:

```
→ public void draw(Graphics2D g2, Rectangle2D plotArea,
    Point2D anchor, PlotState parentState, PlotRenderingInfo state);
    Draws the plot within the specified area.
```

In typical situations, you won't normally call this method directly.

### 33.7.5 Notes

The `CombinedCategoryPlotDemo2.java` file (included in the JFreeChart demo collection) provides an example of this type of plot.

## 33.8 CombinedRangeXYPlot

### 33.8.1 Overview

A subclass of `XYPlot` that allows you to combine multiple plots on one chart, where the subplots share a single range axis, and maintain their own domain axes.

Figure 33.6 illustrates the relationship between the `CombinedRangeXYPlot` and its subplots).

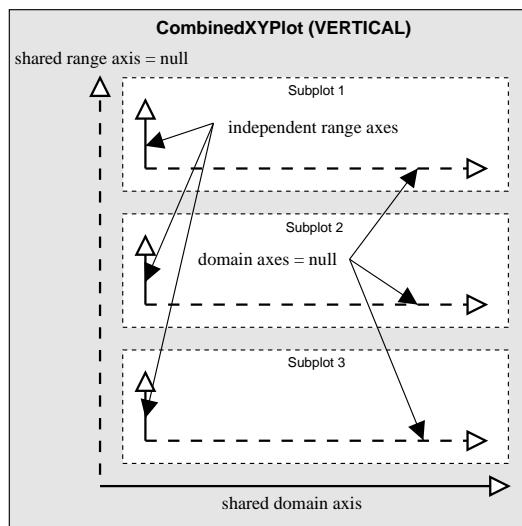


Figure 33.6: `CombinedRangeXYPlot` axes

The `CombinedRangeXYPlotDemo` class provides an example of this type of plot.

### 33.8.2 Methods

There are two methods for adding a subplot to a combined plot:

```
➔ public void add(XYPlot subplot);
```

Adds a subplot to the combined plot, with a weight of 1. Adding a `null` subplot is not permitted. You must ensure that the subplot has a non-`null` domain axis. The subplot's range axis will be set to `null`.

```
➔ public void add(XYPlot subplot, int weight);
```

Adds a subplot to the combined plot, with the specified weight. Adding a `null` subplot is not permitted. You must ensure that the subplot has a non-`null` domain axis. The subplot's range axis will be set to `null`.

The weight determines how much of the plot area is assigned to the subplot. For example, if you add three subplots with weights of 1, 2 and 4, the relative amount of space assigned to each plot is 1/7, 2/7 and 4/7 (where the 7 is the sum of the individual weights).

To remove a subplot:

```
➔ public void remove(XYPlot subplot);
```

Removes the specified subplot and sends a `PlotChangeEvent` to all registered listeners. If `subplot` is `null`, this method throws an `IllegalArgumentException`.

To control the amount of space between the subplots:

```
➔ public void setGap(double gap);
```

Sets the gap (in points) between the subplots.

### 33.8.3 Notes

Some points to note:

- the dataset for this class should be set to `null` (only the subplots display data);
- the subplots managed by this class should have one axis set to `null` (the shared axis is maintained by this class);
- you do not need to set a renderer for the plot, since each subplot maintains its own renderer;
- each subplot uses its own series colors. You should modify the default colors to ensure that the items for each subplot are uniquely colored;
- a demonstration of this type of plot is described in section 14.5.

## 33.9 CompassPlot

### 33.9.1 Overview

A *compass plot* presents directional data in the form of a compass dial—an example is shown in figure 33.7.

### 33.9.2 Constructors

To create a new instance:

```
➔ public CompassPlot();
```

Creates a new `CompassPlot` instance using an instance of `DefaultValueDataset` for the current value.

```
➔ public CompassPlot(ValueDataset dataset);
```

Creates a new `CompassPlot` instance using the specified dataset (which may be `null`).

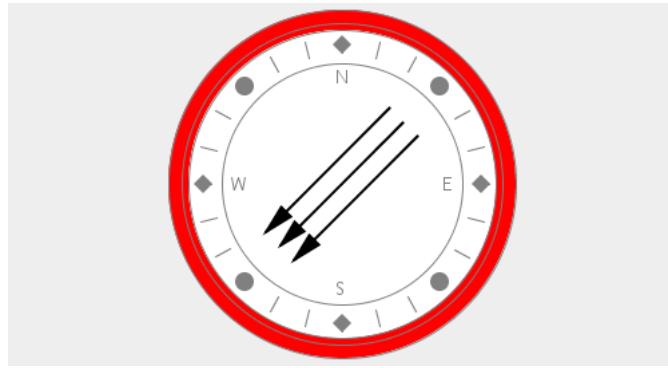


Figure 33.7: A chart that uses `CompassPlot`

### 33.9.3 General Attributes

To control the background color of the interior of the compass:

► `public Paint getRoseCenterPaint();`

Returns the paint (never null) used to fill the interior of the compass dial. The default is `Color.white`.

► `public void setRoseCenterPaint(Paint paint);`

Sets the paint used to fill the interior of the compass dial and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the color of the compass border:

► `public Paint getRosePaint();`

Returns the paint (never null) used to fill the outer border of the compass dial. The default is `Color.yellow`.

► `public void setRosePaint(Paint paint);`

Sets the paint used to fill the outer border of the compass dial and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

The compass border is outlined with a highlight color:

► `public Paint getRoseHighlightPaint();`

Returns the paint (never null) used to draw the edges of the outer border of the compass dial. The default is `Color.black`.

► `public void setRoseHighlightPaint(Paint paint);`

Sets the paint used to draw the edges of the outer border of the compass dial and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

### 33.9.4 The Plot Border

To control whether or not the plot's border/background is drawn:

► `public boolean getDrawBorder();`

Returns the flag that controls whether or not the plot's border is drawn. The default value is `false`.

► `public void setDrawBorder(boolean status);`

Sets the flag that controls whether or not the plot's border is drawn. No change event is generated (this may change in the future).

### 33.9.5 Datasets

A typical plot will use a single dataset (an instance of `ValueDataset`), but it is also possible to display the values from multiple datasets.

- ▶ `public ValueDataset[] getDatasets();`  
Returns an array containing references to this plot's dataset(s).
- ▶ `public void addDataset(ValueDataset dataset);`  
Equivalent to `addDataset(dataset, null)`—see the next method description.
- ▶ `public void addDataset(ValueDataset dataset, MeterNeedle needle);`  
Adds a dataset (and corresponding needle) to the plot.

### 33.9.6 Needles

The value from each dataset is displayed using a needle. To customise the appearance of the needle(s), use the following methods:

- ▶ `public void setSeriesNeedle(int type);`  
Equivalent to `setSeriesNeedle(0, type)`;—see the next method.
- ▶ `public void setSeriesNeedle(int index, int type);`  
Sets the type of needle for the dataset specified by `index`. Recognised types are:
  - 0 – an instance of `ArrowNeedle`;
  - 1 – an instance of `LineNeedle`;
  - 2 – an instance of `LongNeedle`;
  - 3 – an instance of `PinNeedle`;
  - 4 – an instance of `PlumNeedle`;
  - 5 – an instance of `PointerNeedle`;
  - 6 – an instance of `ShipNeedle`;
  - 7 – an instance of `WindNeedle`;
  - 8 – an instance of `ArrowNeedle`;
  - 9 – an instance of `MiddlePinNeedle`;

Note the duplication in items 0 and 8.

To customise the needle used to display the value from a dataset:

- ▶ `public void setSeriesNeedle(int index, MeterNeedle needle);`  
Sets the needle for the dataset specified by `index` and sends a `PlotChangeEvent` to all registered listeners.
- ▶ `public void setSeriesPaint(int series, Paint paint);`  
Sets the fill paint for the needle associated with the dataset specified by the `series` index.
- ▶ `public void setSeriesOutlinePaint(int series, Paint p);`  
Sets the outline paint for the needle associated with the dataset specified by the `series` index.
- ▶ `public void setSeriesOutlineStroke(int series, Stroke stroke);`  
Sets the outline stroke for the needle associated with the dataset specified by the `series` index.

### 33.9.7 General Methods

The compass plot does not display a legend, so the `getLegendItems()` method is overridden to return `null`:

- ▶ `public LegendItemCollection getLegendItems();`  
Returns `null`, so no legend items are displayed.

The scale for a full revolution of the compass is controlled by the following methods:

```
► public double getRevolutionDistance();
Returns the length of a full revolution for the compass. The default value is 360.0, because the
compass displays degree values.

► public void setRevolutionDistance(double size);
Sets the length of a full revolution for the compass.
```

### 33.9.8 Labels

Several label attributes are provided by this class, but never used (deprecate?). The label type is controlled via the following methods:

```
► public int getLabelType();
Returns the label type, which is one of:


- CompassPlot.NO_LABELS;
- CompassPlot.VALUE_LABELS;

```

The default value is `CompassPlot.NO_LABELS`. This attribute is not currently used.

```
► public void setLabelType(int type);
Sets the label type and sends a PlotChangeEvent to all registered listeners.
```

The label font:

```
► public Font getLabelFont();
Returns the label font. The default value is null. This attribute is not currently used.

► public void setLabelFont(Font font);
Sets the label font and sends a PlotChangeEvent to all registered listeners.
```

### 33.9.9 Draw Method

The following method is called by the `JFreeChart` class during chart drawing:

```
► public void draw(Graphics2D g2, Rectangle2D area,
Point2D anchor, PlotState parentState, PlotRenderingInfo info);
Draws the plot within the specified area.
```

In typical situations, you won't normally call this method directly.

### 33.9.10 Notes

Some points to note:

- there is a demonstration `CompassDemo1.java` application included in the JFreeChart demo collection.

## 33.10 ContourPlot

### 33.10.1 Overview

A custom plot that displays ( $x$ ,  $y$ ,  $z$ ) data in the form of a 2D contour plot. *This class is deprecated as of version 1.0.4.*

### 33.10.2 Draw Method

The following method is called by the `JFreeChart` class during chart drawing:

```
► public void draw(Graphics2D g2, Rectangle2D plotArea,
Point2D anchor, PlotState parentState, PlotRenderingInfo state);
Draws the plot within the specified area.
```

In typical situations, you won't normally call this method directly.

## 33.11 ContourPlotUtilities

### 33.11.1 Overview

A class that contains static utility methods used by the contour plot implementation. *This class is deprecated as of version 1.0.4.*

## 33.12 ContourValuePlot

### 33.12.1 Overview

An interface used by the contour plot implementation. *This interface is deprecated as of version 1.0.4.*

## 33.13 CrosshairState

### 33.13.1 Overview

This class maintains information about the crosshairs on a plot, as the plot is being rendered. Crosshairs will often need to “lock on” to the data point nearest to the anchor point (which is usually set by a mouse click). This class keeps track of the data item that is “closest” (either in screen space or in data space) to the anchor point.

### 33.13.2 Constructors

The default constructor:

► `public CrosshairState();`

Creates a new instance where distance is calculated in screen space.

► `public CrosshairState(boolean calculateDistanceInDataSpace);`

Creates a new instance where you can select to measure distance in data space or screen space.

### 33.13.3 Methods

The following method is called as a plot is being rendered:

► `public void updateCrosshairPoint(double candidateX, double candidateY);`

Considers the candidate point and updates the crosshair point if the candidate is the “closest” to the anchor point.

## 33.14 DatasetRenderingOrder

### 33.14.1 Overview

This class defines the tokens that can be used to specify the dataset rendering order in a [CategoryPlot](#) or an [XYPlot](#). There are two tokens defined, as listed in table 33.2.

Token:	Description:
<code>DatasetRenderingOrder.FORWARD</code>	The primary dataset is rendered first, so that it appears to be “underneath” the other datasets.
<code>DatasetRenderingOrder.REVERSE</code>	The primary dataset is rendered last, so it appears to be “on top” of the other datasets.

Table 33.2: *DatasetRenderingOrder* tokens

The default setting is `DatasetRenderingOrder.REVERSE`—this ensures that the primary dataset appears “on top” of the secondary datasets.

### 33.14.2 Usage

To change the rendering order, use the following code:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
plot.setDatasetRenderingOrder(DatasetRenderingOrder.FORWARD);
```

### 33.14.3 Notes

Some points to note:

- an example (`OverlaidBarChartDemo1.java`) is included in the JFreeChart demo collection.

## 33.15 DefaultDrawingSupplier

### 33.15.1 Overview

A default class used to provide a sequence of unique `Paint`, `Stroke` and `Shape` objects to be used by renderers when drawing charts (this class implements the `DrawingSupplier` interface).

### 33.15.2 Usage

Every `Plot` class is initialised with an instance of this class as its drawing supplier, and it is unlikely that you would need to use this class directly. However, you *might* create your own class that implements the `DrawingSupplier` interface, and register it with the plot, as a way of overriding the default series colours, line styles and shapes.

### 33.15.3 Constructors

The default constructor creates a drawing supplier with default sequences:

► `public DefaultDrawingSupplier();`

Creates a new drawing supplier with default sequences:

- the paint sequence is obtained from the `createDefaultPaintArray()` method in the `ChartColor` class;
- the fill paint sequence contains just one colour (`Color.white`);
- the outline paint sequence contains just one colour (`Color.lightGray`);
- the stroke sequence contains just one stroke (`BasicStroke(1.0f, BasicStroke.CAP_SQUARE, BasicStroke.JOIN_BEVEL)`);
- the outline stroke sequence contains just one stroke (`BasicStroke(1.0f, BasicStroke.CAP_SQUARE, BasicStroke.JOIN_BEVEL)`);
- the shape sequence contains 10 shapes defined by the `createStandardSeriesShapes()` method.

The alternate constructor allows you to supply your own sequences:

► `public DefaultDrawingSupplier(Paint[] paintSequence, Paint[] outlinePaintSequence, Stroke[] strokeSequence, Stroke[] outlineStrokeSequence, Shape[] shapeSequence);`

Creates a new drawing supplier with the specified sequences. None of the arrays should be `null`, nor should they contain `null` items.<sup>1</sup>

---

<sup>1</sup>This is not currently enforced.

### 33.15.4 Methods

To get the next paint in the sequence:

```
➔ public Paint getNextPaint();
```

Returns the next item in the paint sequence. This method should never return `null`.

To get the next fill paint in the sequence:

```
➔ public Paint getNextFillPaint(); [1.0.6]
```

Returns the next item in the fill paint sequence. This method should never return `null`. This method was first introduced in JFreeChart 1.0.6.

To get the next outline paint in the sequence:

```
➔ public Paint getNextOutlinePaint();
```

Returns the next item in the outline paint sequence. This method should never return `null`.

To get the next stroke in the sequence:

```
➔ public Stroke getNextStroke();
```

Returns the next item in the stroke sequence. This method should never return `null`.

To get the next outline stroke in the sequence:

```
➔ public Stroke getNextOutlineStroke();
```

Returns the next item in the outline stroke sequence. This method should never return `null`.

To get the next shape in the sequence:

```
➔ public Shape getNextShape();
```

Returns the next shape in the outline stroke sequence. This method should never return `null`.

The following method defines the default shape sequence for JFreeChart:

```
➔ public static Shape[] createStandardSeriesShapes();
```

Returns an array containing ten “standard” shapes.

### 33.15.5 Equals, Cloning and Serialization

To test for equality with an arbitrary object:

```
➔ public boolean equals(Object obj);
```

Tests this drawing supplier for equality with an arbitrary object (`null` permitted)

Instances of this class are cloneable (the `PublicCloneable` interface is implemented) and serializable.

### 33.15.6 Notes

This class provides a default implementation of the `DrawingSupplier` interface.

## 33.16 DialShape

### 33.16.1 Overview

This class defines the tokens that can be used to specify the dial shape in a `MeterPlot`. There are three tokens defined, as listed in table 33.3.

Token:	Description:
<code>DialShape.CIRCLE</code>	A circle.
<code>DialShape.CHORD</code>	A chord.
<code>DialShape.PIE</code>	A pie.

Table 33.3: *DialShape* tokens

The result of applying each shape to a `MeterPlot` is illustrated in figure 33.8.



Figure 33.8: DialShape examples

### 33.16.2 Usage

The `MeterPlot` class has a method named `setDialShape()` that accepts the tokens defined by this class, for example

```
plot.setDialShape(DialShape.CHORD);
```

## 33.17 DrawingSupplier

### 33.17.1 Overview

A *drawing supplier* provides a limitless (but ultimately repeating) sequence of `Paint`, `Stroke` and `Shape` objects that can be used by renderers when drawing charts. All `Plot` classes will have a default drawing supplier. This provides a single source for colors and line styles, which is particularly useful for avoiding duplicates when a plot has multiple renderers.

You can register your own drawing supplier with a plot if you want to modify the default behaviour. If you do this, you need to call the plot's `setDrawingSupplier()` method before the chart is first drawn (the reason being that the plot's renderer(s) will cache the values returned by the drawing supplier the first time a chart is drawn—subsequent changes to the drawing supplier will have no effect on the values already cached).

In version 1.0.6, a new method (`getNextFillPaint()`) was added to this interface (breaking backwards compatibility for those that implement their own custom drawing suppliers).

### 33.17.2 Interface Methods

This interface defines the following methods:

► `public Paint getNextPaint();`

Returns the next `Paint` object in the sequence (never `null`). These are usually used as the default series colors in charts.

► `public Paint getNextOutlinePaint();`

Returns the next outline `Paint` object in the sequence (never `null`).

► `public Paint getNextFillPaint(); [1.0.6]`

Returns the next fill `Paint` object in the sequence (never `null`). This method was added to the interface at version 1.0.6.

► `public Stroke getNextStroke();`

Returns the next `Stroke` object in the sequence (never `null`). These are usually used as the default series line style in charts.

```
► public Stroke getNextOutlineStroke();
Returns the next outline Stroke object in the sequence (never null).

► public Shape getNextShape();
Returns the next Shape object in the sequence (never null). The shapes returned by this method
should be centered on (0, 0) in Java2D coordinates.
```

### 33.17.3 Notes

Some points to note:

- the `DefaultDrawingSupplier` class provides an implementation of this interface;
- if you write your own implementation of this interface, you should ensure that it implements the `PublicCloneable` interface and is serializable. Otherwise, plots that use your implementation will no longer be cloneable or serializable.

## 33.18 FastScatterPlot

### 33.18.1 Overview

A custom plot that aims to be fast rather than flexible. It displays a single data series in a scatter plot format (that is, a dot to represent each data item). A couple of techniques are used to make this plot type faster than the other plot types provided by JFreeChart:

- data is obtained directly from an array rather than via the `XYDataset` interface;
- the plot draws each point directly rather than using a plug-in renderer.

### 33.18.2 Constructors

This class has two constructors:

```
► public FastScatterPlot();
Creates a new plot with no data, and axes labelled "X" and "Y".

► public FastScatterPlot(float[][] data, ValueAxis domainAxis, ValueAxis rangeAxis);
Creates a new plot with the specified data and axes. For a description of the data array
format, see section 33.18.3. If domainAxis or rangeAxis is null, this constructor throws an
IllegalArgumentException.
```

### 33.18.3 The Data

The data for this plot is supplied as an array containing two equal-length subarrays—one for the x-values and one for the y-values. The following sample code illustrates the creation of a small data array in the correct format:

```
float[] x = new float[] { 1.0f, 2.0f, 5.0f, 6.0f };
float[] y = new float[] { 7.3f, 2.7f, 8.9f, 1.0f };
float[][] data = new float[][] { x, y };
```

To get/set the data array for the plot:

```
► public float[][] getData();
Returns a reference to the data array used by this plot (this might be null). Note that if you
update the values in the array directly, the chart will NOT be automatically repainted (as there
is no mechanism to notify the chart that the dataset has been updated).

► public void setData(float[][] data);
Sets the data array for the plot, replacing any existing data, and sends a PlotChangeEvent to
all registered listeners. If you set this to null, no data is displayed in the plot.
```

### 33.18.4 The Axes

This plot supports a single domain (x) axis and a single range (y) axis. To control the domain axis:

- **public ValueAxis getDomainAxis();**  
Returns the domain axis for the plot (never `null`).
- **public void setDomainAxis(ValueAxis axis); [1.0.3]**  
Sets the domain axis for the plot, and sends a `PlotChangeEvent` to all registered listeners. If `axis` is `null`, this method throws an `IllegalArgumentException`.

To control the range axis:

- **public ValueAxis getRangeAxis();**  
Returns the range axis for the plot (never `null`).
- **public void setRangeAxis(ValueAxis axis); [1.0.3]**  
Sets the range axis for the plot, and sends a `PlotChangeEvent` to all registered listeners. If `axis` is `null`, this method throws an `IllegalArgumentException`.

A utility method returns the range of data values for either of the plot's axes:

- **public Range getDataRange(ValueAxis axis);**  
Returns the range of values in the current dataset for the specified axis. If `axis` does not belong to the plot, this method returns `null`.

### 33.18.5 Rendering

For efficiency, this plot renders the data items directly, rather than via a plug-in renderer. You can modify the color used to draw the dots for each data item:

- **public Paint getPaint();**  
Returns the paint used to draw the dot for each data item (never `null`). The default value is `Color.RED`.
- **public void setPaint(Paint paint);**  
Sets the paint used to draw the dot for each data item, and sends a `PlotChangeEvent` to all registered listeners.

Only one rendering color is defined, because this plot can only display data for a single series.

### 33.18.6 Gridlines

You can display gridlines against the domain axis and/or the range axis. For both sets of gridlines, you can control:

- the visibility of the gridlines (whether or not they are displayed at all);
- the color of the gridlines;
- the line style for the gridlines;

For the domain axis gridlines:

- **public boolean isDomainGridlinesVisible();**  
Returns `true` if gridlines are drawn for the domain axis, and `false` otherwise. The default value is `true`.
- **public void setDomainGridlinesVisible(boolean visible);**  
Sets a flag that controls whether or not the gridlines are displayed and sends a `PlotChangeEvent` to all registered listeners.
- **public Paint getDomainGridlinePaint();**  
Returns the paint used to draw the domain axis gridlines (never `null`). The default value is `Color.lightGray`.

```
→ public void setDomainGridlinePaint(Paint paint);
Sets the paint used to draw the domain axis gridlines and sends a PlotChangeEvent to all registered listeners. If paint is null, this method throws an IllegalArgumentException.
```

```
→ public Stroke getDomainGridlineStroke();
Returns the stroke used to draw the domain axis gridlines (never null). The default stroke is a thin dashed line.
```

```
→ public void setDomainGridlineStroke(Stroke stroke);
Sets the stroke used to draw the domain axis gridlines and sends a PlotChangeEvent to all registered listeners. If stroke is null, this method throws an IllegalArgumentException.
```

Similarly, for the range axis gridlines:

```
→ public boolean isRangeGridlinesVisible();
Returns true if gridlines are drawn for the range axis, and false otherwise. The default value is true.
```

```
→ public void setRangeGridlinesVisible(boolean visible);
Sets a flag that controls whether or not the gridlines are displayed and sends a PlotChangeEvent to all registered listeners.
```

```
→ public Paint getRangeGridlinePaint();
Returns the paint used to draw the range axis gridlines (never null). The default value is Color.lightGray.
```

```
→ public void setRangeGridlinePaint(Paint paint);
Sets the Paint used for the range gridlines and sends a PlotChangeEvent to all registered listeners.
```

```
→ public Stroke getRangeGridlineStroke();
Returns the stroke used to draw the range axis gridlines (never null). The default stroke is a thin dashed line.
```

```
→ public void setRangeGridlineStroke(Stroke stroke);
Sets the Stroke used for the range gridlines and sends a PlotChangeEvent to all registered listeners. If stroke is null, this method throws an IllegalArgumentException.
```

### 33.18.7 Zooming

To support the zooming operations that can be invoked from the `ChartPanel` class, this plot implements the `Zoomable` interface:

```
→ public boolean isDomainZoomable();
Always returns true, because the plot supports zooming along the domain axis.
```

```
→ public boolean isRangeZoomable();
Always returns true, because the plot supports zooming along the range axis.
```

To find the current orientation of the plot:

```
→ public PlotOrientation getOrientation();
Returns the plot orientation, which is always VERTICAL for this class.
```

To invoke zooming on the domain axis:

```
→ public void zoomDomainAxes(double factor, PlotRenderingInfo info, Point2D source);
Zooms the domain axis in or out by the specified factor.
```

```
→ public void zoomDomainAxes(double lowerPercent, double upperPercent, PlotRenderingInfo info,
Point2D source);
Zooms the domain axis to the specified lower and upper bounds.
```

To invoke zooming on the range axis:

```
→ public void zoomRangeAxes(double factor, PlotRenderingInfo info, Point2D source);
Zooms the range axis in or out by the specified factor.
```

```
→ public void zoomRangeAxes(double lowerPercent, double upperPercent, PlotRenderingInfo info,
Point2D source);
Zooms the range axis to the specified lower and upper bounds.
```

### 33.18.8 Other Methods

A string describing the plot type is returned by the following method:

```
→ public String getPlotType();
```

Returns a string describing the plot type. The string is localised, and intended for display in a user interface (such as a plot property editor).

The following method is called by the `JFreeChart` class during chart drawing:

```
→ public void draw(Graphics2D g2, Rectangle2D plotArea,
```

```
Point2D anchor, PlotState parentState, PlotRenderingInfo state);
```

Draws the plot within the specified area. In typical situations, you won't normally call this method directly.

```
→ public void render(Graphics2D g2, Rectangle2D dataArea, PlotRenderingInfo info, CrosshairState crosshairState);
```

Called by the `draw()` method, this method renders the data values.

### 33.18.9 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj);
```

Tests this plot for equality with an arbitrary object. Returns `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `FastScatterPlot`;
- `obj` has the same attributes as this plot (including the data values).

Instances of this class are cloneable and serializable.

### 33.18.10 Notes

Some points to note:

- this plot does not support multiple datasets or axes;
- you cannot specify the orientation of the plot (it is always `PlotOrientation.VERTICAL`);
- this plot has no support for tooltips;
- a demo (`FastScatterPlotDemo.java`) is included in the JFreeChart demo collection.

## 33.19 GreyPalette

### 33.19.1 Overview

A grey palette (extends `ColorPalette`) used by the `ContourPlot` class. *This class is deprecated as of version 1.0.4.*

## 33.20 IntervalMarker

### 33.20.1 Overview

An `IntervalMarker` is used to highlight a (fixed) range of values against the domain or range axis for a `CategoryPlot` or an `XYPLOT`. This class extends the `Marker` class.

### 33.20.2 Usage

You can add a marker to an `XYPlot` using the `addDomainMarker()` or `addRangeMarker()` methods. Similarly, you can add a range marker to a `CategoryPlot` using the `addRangeMarker()` method.

There is a demo application (`DifferenceChartDemo2.java`) included in the JFreeChart demo collection that illustrates the use of this class.

### 33.20.3 Constructors

This class defines several constructors:

- `public IntervalMarker(double start, double end);`  
Creates a new instance for the specified range (in data space).
- `public IntervalMarker(double start, double end, Paint paint); [1.0.9]`  
Equivalent to `IntervalMarker(start, end, paint, new BasicStroke(0.5f), null, null, 0.8f)`—see the next constructor.
- `public IntervalMarker(double start, double end, Paint paint, Stroke stroke, Paint outlinePaint, Stroke outlineStroke, float alpha);`  
Creates a new instance for the specified range (in data space) and with the given attributes.

### 33.20.4 General Attributes

In addition to the methods inherited from `Marker`, this class defines:

- `public double getStartValue();`  
Returns the lower bound of the interval.
- `public void setStartValue(double value); [1.0.3]`  
Sets the lower bound of the interval to be highlighted, and sends a `MarkerChangeEvent` to all registered listeners.
- `public double getEndValue();`  
Returns the upper value in the interval.
- `public void setEndValue(double value); [1.0.3]`  
Sets the upper bound of the interval to be highlighted, and sends a `MarkerChangeEvent` to all registered listeners.

### 33.20.5 Gradient Paint Support

The marker supports the use of `GradientPaint` via a transformer that can dynamically update the coordinates of the gradient to match the interval area:

- `public GradientPaintTransformer getGradientPaintTransformer();`  
Returns the transformer applied to any `GradientPaint` instances used by the marker. This method can return `null`, in which case any `GradientPaint` instance is used without transformation.
- `public void setGradientPaintTransformer(GradientPaintTransformer transformer);`  
Sets the transformer that will be applied to any `GradientPaint` instances used by the marker and sends a `MarkerChangeEvent` to all registered listeners. If `transformer` is `null`, any `GradientPaint` instance will be used without transformation.

### 33.20.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this instance for equality with an arbitrary object (which may be `null`).

Instances of this class are `Cloneable` and `Serializable`.

### 33.20.7 Notes

Markers don't have any code to draw themselves, this function is delegated to the renderer classes.

#### See Also

[ValueMarker](#).

## 33.21 Marker

### 33.21.1 Overview

The base class for markers that can be added to a [CategoryPlot](#) or an [XYPlot](#). Markers are used to highlight particular values or value ranges against either the domain or range axes. Markers can be displayed with or without labels. This abstract base class has three subclasses, as listed in Table 33.4.

Class:	Description:
<a href="#">CategoryMarker</a>	A marker that highlights a category on the domain axis of a <a href="#">CategoryPlot</a> .
<a href="#">ValueMarker</a>	A marker that highlights a single value on a numerical or date axis.
<a href="#">IntervalMarker</a>	A marker that highlights a range of values.

Table 33.4: Subclasses of *Marker*

### 33.21.2 Usage

Demo applications ([MarkerDemo1](#) and [MarkerDemo2](#)) illustrating the use of markers are included in the [JFreeChart demo collection](#).

### 33.21.3 Constructors

Several constructors are provided for the use of subclasses—they are protected, so you cannot call them directly:

```
► protected Marker();
Creates a new marker with default attributes. This is equivalent to new Marker(Color.gray).

► protected Marker(Paint paint);
Creates a new marker with the specified paint (null is not permitted). The other fields take
the following default values:


- stroke: new BasicStroke(0.5f);
- outlinePaint: Color.gray;
- outlineStroke: new BasicStroke(0.5f);
- alpha: 0.80f;



► protected Marker(Paint paint, Stroke stroke, Paint outlinePaint, Stroke outlineStroke,
float alpha);
Creates a new marker with the specified attributes. The paint and stroke arguments cannot
be null. The alpha argument should be in the range 0.0 to 1.0.
```

The label attributes, which cannot be specified in these constructors, take the following default values:

Attribute:	Default Value:
labelFont	<code>new Font("SansSerif", Font.PLAIN, 9);</code>
labelPaint	<code>Color.black;</code>
labelAnchor	<code>RectangleAnchor.TOP_LEFT;</code>
labelOffset	<code>new RectangleInsets(3.0, 3.0, 3.0, 3.0);</code>
labelOffsetType	<code>LengthAdjustmentType.CONTRACT;</code>
labelTextAnchor	<code>TextAnchor.CENTER;</code>

*Table 33.5: Attribute Default Values*

### 33.21.4 General Attributes

This section describes the general attributes that control the appearance of markers. Label attributes are covered in the next section.

To control the paint used to draw the marker:

- `public Paint getPaint();`  
Returns the paint used to draw the marker (never `null`). The default value is `Color.gray`.
- `public void setPaint(Paint paint);`  
Sets the paint used to draw the marker (`null` is not permitted) and sends a `MarkerChangeEvent` to all registered listeners.

To control the stroke used to draw markers that are rendered as lines:

- `public Stroke getStroke();`  
Returns the stroke used to draw the marker, if it is drawn as a line (never `null`). The default value is `BasicStroke(0.5f)`. If the marker is a rectangular region, the outline is drawn using `getOutlineStroke()`, so this attribute is not used in that case.
- `public void setStroke(Stroke stroke);`  
Sets the stroke used to draw the marker when it is drawn as a line (`null` is not permitted) and sends a `MarkerChangeEvent` to all registered listeners.

To control the paint used to draw marker outlines:

- `public Paint getOutlinePaint();`  
Returns the paint used to draw the marker outline (possibly `null`). The default value is `Color.gray`. This field is not used when the marker is drawn as a line.
- `public void setOutlinePaint(Paint paint);`  
Sets the paint used to draw the marker outline when it is drawn as a shape (typically a rectangle), rather than a line (set this to `null` if you do not want the outline drawn). After changing the value, this method sends a `MarkerChangeEvent` to all registered listeners.

To control the stroke used to draw marker outlines:

- `public Stroke getOutlineStroke();`  
Returns the stroke used to draw the marker outline (possibly `null`). The default value is `BasicStroke(0.5f)`. This is not used when the marker is drawn as a line.
- `public void setOutlineStroke(Stroke stroke);`  
Sets the stroke used to draw the marker outline when it is drawn as a shape (typically a rectangle), rather than a line (set this to `null` if you do not want the outline drawn). After changing the value, this method sends a `MarkerChangeEvent` to all registered listeners.

To control the alpha transparency of the marker:

- `public float getAlpha();`  
Returns the alpha transparency for the marker (a value in the range `0.0f` to `1.0f`). `0.0f` is completely transparent and `1.0f` is completely opaque.
- `public void setAlpha(float alpha);`  
Sets the alpha transparency that should be used to draw the marker. This is a value in the range `0.0f` (completely transparent) to `1.0f` (completely opaque). After changing the value, this method sends a `MarkerChangeEvent` to all registered listeners.

### 33.21.5 Label Attributes

Labels can be drawn on or near markers. This section describes the attributes that control the appearance and position of the label.

These methods control the label text, font and color:

► `public String getLabel();`

Returns the label text (which may be `null`). If the label string is `null` (the default), the marker will be drawn without a label.

► `public void setLabel(String label);`

Sets the label text (`null` is permitted) and sends a `MarkerChangeEvent` to all registered listeners.

► `public Font getLabelFont();`

Returns the font used to display the label (never `null`). The default value is `Font("SansSerif", Font.PLAIN, 9)`.

► `public void setLabelFont(Font font);`

Sets the font used to display the label (`null` is not permitted) and sends a `MarkerChangeEvent` to all registered listeners.

► `public Paint getLabelPaint();`

Returns the paint used to display the label text (never `null`). The default value is `Color.black`.

► `public void setLabelPaint(Paint paint);`

Sets the paint used to display the label text (`null` is not permitted) and sends a `MarkerChangeEvent` to all registered listeners.

The remaining methods control the position of the label relative to the marker bounds when it is drawn on the plot:

► `public RectangleAnchor getLabelAnchor();`

Returns the attribute that defines the anchor point, relative to the marker bounds, that the label will be aligned to. The actual point is offset slightly from the marker bounds—see the `getLabelOffset()` method.

► `public void setLabelAnchor(RectangleAnchor anchor);`

Sets the point on the marker bounds that is used for alignment of the label, then sends a `MarkerChangeEvent` to all registered listeners. This anchor (after being adjusted by the label offsets) determines a fixed point on the chart that the marker label can be aligned to.

Figure 33.9 illustrates how the marker label anchor position is calculated relative to the marker's bounds. One of the nine potential anchors is selected via the `setLabelAnchor()` method, and the margin between the marker's bounds and the potential anchor points is determined by the `getLabelOffset()` and `getLabelOffsetType()` methods.

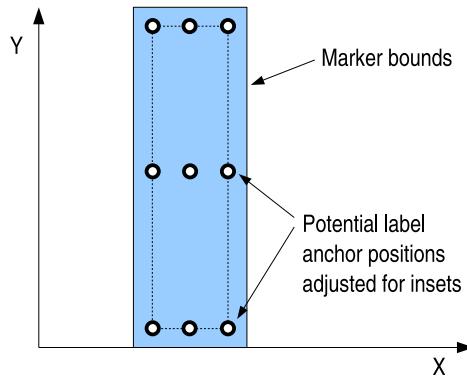


Figure 33.9: Marker insets and the label anchor

```
► public RectangleInsets getLabelOffset();
```

Returns the label offsets (never `null`). The default value is `RectangleInsets(3.0, 3.0, 3.0, 3.0)` (that is, the anchor points lie on a rectangle three Java2D units inside (or outside) the marker's bounding rectangle).

```
► public void setLabelOffset(RectangleInsets offset);
```

Sets the offset between the marker's bounds and the label anchor points (`null` is not permitted), then sends a `MarkerChangeEvent` to all registered listeners.

```
► public LengthAdjustmentType getLabelOffsetType();
```

Returns the label offset type, typically either `CONTRACT` (the default) or `EXPAND`. This controls how the insets returned by `getLabelOffset()` are applied to calculate the label anchor position—figure 33.9 illustrates the `CONTRACT` option, while figure 33.10 illustrates the `EXPAND` option.

```
► public void setLabelOffsetType(LengthAdjustmentType adj);
```

Sets the label offset type, which should be either `CONTRACT` or `EXPAND` (`null` is not permitted), then sends a `MarkerChangeEvent` to all registered listeners.

Figure 33.10 illustrates how the label anchor points can be positioned outside the marker's bounds (by setting the label offset type to `EXPAND`).

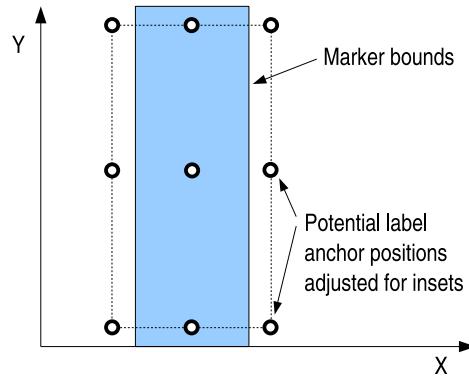


Figure 33.10: Marker insets and the label anchor

To set the point on the label that is aligned to the label anchor:<sup>2</sup>

```
► public TextAnchor getLabelTextAnchor();
```

Returns the point on the label bounds that is aligned to the label anchor point (the default is `TextAnchor.CENTER`). This method never returns `null`.

```
► public void setLabelTextAnchor(TextAnchor anchor);
```

Sets the point on the label that is aligned to the fixed point on the chart determined by the `getLabelAnchor()` method, then sends a `MarkerChangeEvent` to all registered listeners.

Note that if the marker denotes a single value, the bounding rectangle may have zero width.

### 33.21.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
```

Tests this marker for equality with an arbitrary object. This method returns `true` if and only if:

- `obj` is not `null`;

<sup>2</sup>Try running `DrawStringDemo.java`, in the JCommon distribution, to get an understanding of how the `TextAnchor` setting controls basic string alignment.

- `obj` is an instance of `Marker`;
- `obj` has the same attribute values as this marker.

Instances of this class are cloneable and serializable (in order that charts that have markers can be cloneable and serializable).

### 33.21.7 Notes

Some points to note:

- markers are drawn on the chart by the plot's main renderer—see the default drawing methods defined in `AbstractCategoryItemRenderer` and `AbstractXYItemRenderer`;
- prior to version 1.0.3, there was no change notification mechanism for markers, so charts were not updated automatically when marker attributes changed (one way to trigger a repaint of the chart, at least for charts displayed in a `ChartPanel`, was to call `chart.setNotify(true)`). From version 1.0.3 onwards, there is a change notification mechanism, so the repaint will occur automatically.

## 33.22 MeterInterval

### 33.22.1 Overview

Represents a range of values on a `MeterPlot` that should be highlighted for some reason. For example, on a temperature dial you might show intervals for “normal”, “high” and “extreme”.

### 33.22.2 Constructors

To create a new interval:

- ➔ `public MeterInterval(String label, Range range);`  
Creates a new interval with the specified label and range. The default outline paint for the interval is `Color.yellow`, the default outline stroke is `BasicStroke(2.0f)`, and the default background paint is `null`.
- ➔ `public MeterInterval(String label, Range range, Paint outlinePaint, Stroke outlineStroke, Paint backgroundPaint);`  
Creates a new interval with the specified label and range. The label is typically displayed in the plot's legend (if visible). The range is highlighted by filling the background with `backgroundPaint`. If `label` or `range` is `null`, this method throws an `IllegalArgumentException`. All other arguments can be `null`.

### 33.22.3 Methods

To get the label for the interval:

- ➔ `public String getLabel();`  
The label for the interval. This will normally be displayed in the plot's legend.
- ➔ `public Range getRange();`  
Returns the value range for the interval.
- ➔ `public Paint getBackgroundPaint();`  
Returns the paint used to fill the background for the interval.
- ➔ `public Paint getOutlinePaint();`  
Returns the paint used to draw the outline for the interval.
- ➔ `public Stroke getOutlineStroke();`  
Returns the stroke used to draw the outline for the interval.

### 33.22.4 Equals, Cloning and Serialization

To test for equality with an arbitrary object:

```
➔ public boolean equals(Object obj);
Tests the interval for equality with an arbitrary object.
```

Instances of this class are immutable, so it is not necessary to clone them. Serialization is supported.

### 33.22.5 Notes

Some points to note:

- this class is immutable—you cannot change the interval's range or other attributes.

## 33.23 MeterPlot

### 33.23.1 Overview

A plot that displays a single value in a dial presentation. The current value is represented by a needle in the dial, and is also displayed in the center of the dial in text format.

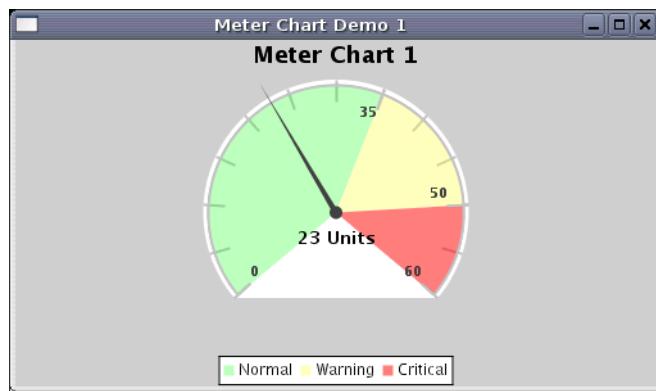


Figure 33.11: A meter chart

Specific intervals in the dial can be highlighted by adding `MeterInterval` instances to the plot.

A new class (`DialPlot`) is now included in the `experimental` directory of the JFreeChart distribution—it is intended that `DialPlot` will eventually replace `MeterPlot`.

### 33.23.2 Constructors

To create a new `MeterPlot`:

```
➔ public MeterPlot();
Creates a new plot with a default range of 0 to 100 and no dataset.

➔ public MeterPlot(ValueDataset dataset);
Creates a dial with default settings, using the supplied dataset.
```

The plot can be customised after it is created, if the default values are not suitable.

### 33.23.3 The Dataset

A `MeterPlot` displays a single value, but still uses a dataset to represent the value rather than relying on a “value” field in the plot. This maintains the separation between the data and the “view”, and consistency with other plot types in JFreeChart.

To access the current dataset:

```
➔ public ValueDataset getDataset();
Returns the current dataset (possibly null).

➔ public void setDataset(ValueDataset dataset);
Sets the dataset for the plot (null permitted) and sends a PlotChangeEvent to all registered
listeners. If the dataset is set to null, no value will be displayed on the dial.
```

To update the displayed value in the chart, call the `setValue()` method in the dataset. This will trigger a `DatasetChangeEvent` which will be picked up by the chart (and cause the chart to be repainted if it is displayed in a `ChartPanel`).

### 33.23.4 The Current Value Display

A needle is used to indicate the current value on the dial. To change the color of the needle:

```
➔ public Paint getNeedlePaint();
Returns the paint used to display the needle on the dial. The default is Color.green.

➔ public void setNeedlePaint(Paint paint);
Sets the color of the needle on the dial and sends a PlotChangeEvent to all registered listeners.
An IllegalArgumentException is thrown if paint is null.
```

The current value is also displayed (near the center of the dial) in text format, with the units appended. To change the font used to display the current value:

```
➔ public Font getValueFont();
Returns the font used to display the current value in the middle of the plot (never null).

➔ public void setValueFont(Font font);
Sets the font used to display the current value and sends a PlotChangeEvent to all registered
listeners. An IllegalArgumentException is thrown if font is null.
```

To change the color used to display the current value:

```
➔ public Paint getValuePaint();
Returns the paint used to display the current value (never null).

➔ public void setValuePaint(Paint paint);
Sets the paint used to display the current value and sends a PlotChangeEvent to all registered
listeners. An IllegalArgumentException is thrown if paint is null.
```

To change the “units” for the value:

```
➔ public String getUnits();
Returns a string describing the units for the dial (possibly null). This is displayed after the
value in the middle of the dial.

➔ public void setUnits(String units);
Sets the unit description for the plot and sends a PlotChangeEvent to all registered listeners. If
this is set to null, then no units are displayed with the meter value.
```

### 33.23.5 The Dial Range, Shape and Background

The range of values that can be displayed on the dial is configurable using the following methods:

```
➔ public Range getRange();
Returns the range of data values on the dial (never null). The default is 0 to 100.
```

```
► public void setRange(Range range);
```

Sets the range of data values on the dial and sends a `PlotChangeEvent` to all registered listeners. An `IllegalArgumentException` is thrown if `range` is `null`. If the current value in the plot's dataset falls outside this range, no needle will be displayed.

To control the shape of the dial:

```
► public DialShape getDialShape();
```

Returns the dial shape (never `null`). The default is `DialShape.CIRCLE`.

```
► public void setDialShape(DialShape shape);
```

Sets the dial shape and sends a `PlotChangeEvent` to all registered listeners. An `IllegalArgumentException` is thrown if `shape` is `null`. Refer to the description of the `DialShape` class for a sample of the available shapes.

The angle spanned by the dial is configurable with the following methods:

```
► public int getMeterAngle();
```

Returns the angle (in degrees) of the full range of the dial. The default value is 270 degrees.

```
► public void setMeterAngle(int angle);
```

Sets the angle (in degrees) of the full range of the dial. This is required to be in the range 1 to 360 degrees.

To change the background color of the dial:

```
► public Paint getDialBackgroundPaint();
```

Returns the paint used for the dial background (never `null`). The default is `Color.black`.

```
► public void setDialBackgroundPaint(Paint paint);
```

Sets the color of the dial background. If you set this to `null`, no background is painted.

To control the outline paint for the dial:

```
► public Paint getDialOutlinePaint();
```

Returns the paint used to draw the dial outline (possibly `null`).

```
► public void setDialOutlinePaint(Paint paint);
```

Sets the paint used to draw the dial outline and sends a `PlotChangeEvent` to all registered listeners.

The dial can be drawn with or without a border:

```
► public boolean getDrawBorder();
```

Returns the flag that controls whether or not a border is drawn around the dial.

```
► public void setDrawBorder(boolean draw);
```

Sets the flag that controls whether or not a border is drawn around the dial and sends a `PlotChangeEvent` to all registered listeners.

### 33.23.6 Tick Labels

Labels are drawn for the first and last ticks only (this is a limitation that needs to be addressed):

```
► public boolean getTickLabelsVisible();
```

Returns `true` if the tick labels should be displayed, and `false` otherwise.

```
► public void setTickLabelsVisible(boolean visible);
```

Sets the flag that controls whether or not tick labels are visible, and sends a `PlotChangeEvent` to all registered listeners.

The font for the labels is controlled with the following methods:

```
► public Font getTickLabelFont();
```

Returns the font used to display the tick labels.

```
► public void setTickLabelFont(Font font);
```

Sets the font used to display the tick labels and sends a `PlotChangeEvent` to all registered listeners.

The paint for the labels is controlled with the following methods:

- `public Paint getTickLabelPaint();`  
Returns the paint used to display the tick labels.
- `public void setTickLabelPaint(Paint paint);`  
Sets the paint used to display the tick labels and sends a `PlotChangeEvent` to all registered listeners.

The formatter for the labels is controlled with the following methods:

- `public NumberFormat getTickLabelFormat();`  
Returns the formatter used to convert the tick values to strings for display.
- `public void setTickLabelFormat(NumberFormat format);`  
Sets the formatter used to convert the tick values to strings for display and sends a `PlotChangeEvent` to all registered listeners.

### 33.23.7 Intervals

It is possible to highlight certain data ranges by adding one or more `MeterInterval` instances to the plot.

- `public List getIntervals();`  
Returns an unmodifiable list of the intervals for the plot. The list may be empty.
- `public void addInterval(MeterInterval interval);`  
Adds an interval to the plot.
- `public void clearIntervals();`  
Removes all intervals from the plot and sends a `PlotChangeEvent` to all registered listeners.

The sample chart in figure 33.11 contains three intervals labelled “Normal”, “Warning” and “Critical”.

### 33.23.8 Legend Items

This plot utilises the legend to display descriptions for the `MeterInterval` instances (if any) that have been added to the plot. The following method returns the required items:

- `public LegendItemCollection getLegendItems();`  
Returns a collection of legend items for the plot. For this plot, there is one item for each `MeterInterval` that has been added to the plot.

You can override this method to customise the legend display.

### 33.23.9 Drawing Methods

This class has several drawing methods that are used internally. In some cases, you can override these methods to change the appearance of the plot:

- `public void draw(Graphics2D g2, Rectangle2D plotArea, Point2D anchor, PlotState parentState, PlotRenderingInfo state);`  
Draws the plot within the specified area. This method is called by the `JFreeChart` class.
- `protected void drawArc(Graphics2D g2, Rectangle2D area, double minValue, double maxValue, Paint paint, Stroke stroke);`  
Draws an arc between the specified values.
- `protected void fillArc(Graphics2D g2, Rectangle2D area, double minValue, double maxValue, Paint paint, boolean dial);`  
Fills the background area between the specified values.
- `protected void drawArcForInterval(Graphics2D g2, Rectangle2D meterArea, MeterInterval interval);`  
Draws an interval arc.

```
► protected void drawTick(Graphics2D g2, Rectangle2D meterArea, double value);
Draws a tick, with no label, for the given value.

► protected void drawTick(Graphics2D g2, Rectangle2D meterArea, double value, boolean label);
Draws a tick, with or without a corresponding label, for the given value.

► protected void drawValueLabel(Graphics2D g2, Rectangle2D area);
Draws the text in the middle of the dial that displays the current value.
```

### 33.23.10 Other Methods

To obtain a short description of the plot type:

```
► public String getPlotType();
Returns a short localised string representing the plot type.
```

To convert a data value to an angle:

```
► public double valueToAngle(double value);
Returns the angle in degrees corresponding to the given data value.
```

The zooming method is overridden to do nothing, zooming is not supported by this plot:

```
► public void zoom(double percent);
This method is overridden to do nothing.
```

### 33.23.11 Equals, Cloning and Serialization

To test the plot for equality with an arbitrary object:

```
► public boolean equals(Object obj);
Tests the plot for equality with an arbitrary object. The plot is equal to obj if and only if:
```

- `obj` is not `null`;
- `obj` is an instance of `MeterPlot`;
- this plot and `obj` have the same field values (not including the dataset, which is ignored for the purposes of equality testing).

This class is both cloneable and serializable.

### 33.23.12 Notes

Some points to note:

- the original version of this class was contributed by Hari;
- the `MeterChartDemo1` and `MeterChartDemo2` classes in the JFreeChart demo collection provide a working example of this class.
- the `DialPlot` class provides a newer and more flexible alternative to this class.

#### See Also

[ValueDataset](#), [DialShape](#), [MeterInterval](#).

## 33.24 MultiplePiePlot

### 33.24.1 Overview

A specialised plot that displays data from a `CategoryDataset` in the form of multiple pie charts. Figure 33.12 shows an example.

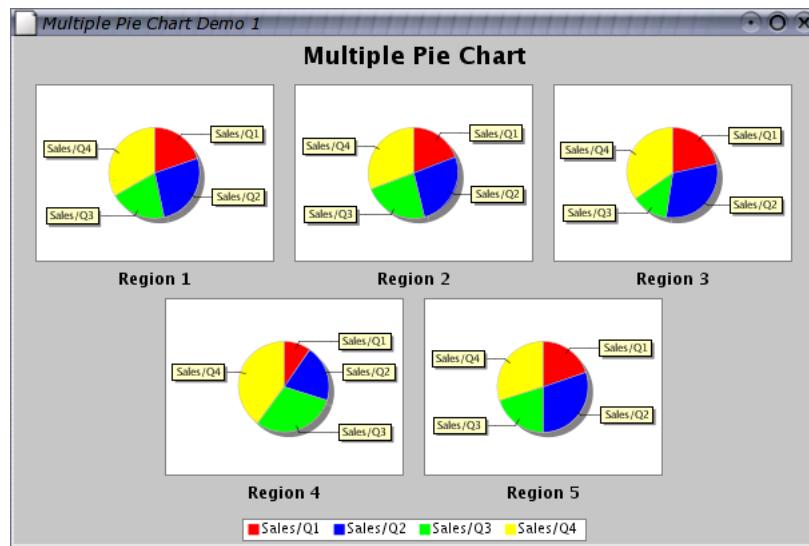


Figure 33.12: A multiple pie chart

### 33.24.2 Constructors

There are two constructors for this class:

► `public MultiplePiePlot();`

Creates a new plot with a `null` dataset.

► `public MultiplePiePlot(CategoryDataset dataset);`

Creates a new plot with the specified dataset (which can be `null`). Data for the individual pie charts is extracted from the dataset by column (you can change this using the `setDataExtractOrder()` method).

### 33.24.3 Methods

This plot uses a single chart instance to draw the multiple pie charts:

► `public JFreeChart getPieChart();`

Returns the chart that is used to render each pie chart in the plot. Any changes you make to this chart will be reflected in the appearance of all the pie charts.

► `public void setPieChart(JFreeChart pieChart);`

Sets the chart that is used to render each of the pie charts in the plot. The `getPlot()` method for this chart MUST return a `PiePlot` instance (this includes `PiePlot3D` and `RingPlot`, since these are subclasses of `PiePlot`). It is advisable to use a chart that does not include a legend.

To access the current dataset for the plot:

► `public CategoryDataset getDataset();`

Returns the current dataset, which may be `null`.

► `public void setDataset(CategoryDataset dataset);`

Sets the dataset for the plot and sends a `PlotChangeEvent` to all registered listeners. The plot will register itself with the new dataset so that it receives notification of any changes to the dataset (and also will unregister from the old dataset).

An important factor determining the appearance of this plot is the order in which the data is extracted for the pie charts:

► `public TableOrder getDataExtractOrder();`

Returns a key that determines how data is extracted (by column or by row) to form the individual pie charts. The default is `TableOrder.BY_COLUMN`.

```
► public void setDataExtractOrder(TableOrder order);
Sets the order of data extraction to one of TableOrder.BY_COLUMN or TableOrder.BY_ROW. In the first case, the number of pie charts displayed will be equal to the number of columns in the dataset, and in the second case it will be equal to the number of rows in the dataset.
```

A lower limit can be specified and will be used to aggregate small data values:

```
► public double getLimit();
Returns the smallest value that will be displayed in its own pie section (the default is 0.0). All sections with values less than this will be aggregated into a single section.

► public void setLimit(double limit);
Sets the smallest value that will be displayed in its own pie section and sends a PlotChangeEvent to all registered listeners.
```

The key used for aggregated data items can be accessed with the following methods:

```
► public Comparable getAggregatedItemsKey(); [1.0.2]
Returns the key used for aggregated items (never null). The default is Other—this can be changed by calling the setAggregatedItemsKey() method.

► public void setAggregatedItemsKey(Comparable key); [1.0.2]
Sets the key that is used for aggregated items and sends a PlotChangeEvent to all registered listeners. This method throws an IllegalArgumentException if key is null.
```

To determine the paint used to display the pie section for aggregated items:

```
► public Paint getAggregatedItemsPaint(); [1.0.2]
Returns the paint used to display the pie section for aggregated items. The default value is Color.lightGray and this field cannot be set to null.

► public void setAggregatedItemsPaint(Paint paint); [1.0.2]
Sets the paint used to display the pie section for aggregated items and sends a PlotChangeEvent to all registered listeners. This method throws an IllegalArgumentException if paint is null.
```

### 33.24.4 Miscellaneous Methods

The plot type is described by the following method:

```
► public String getPlotType();
Returns the string Multiple Pie Plot.
```

The legend items are created by the following method (which you are free to override):

```
► public LegendItemCollection getLegendItems();
Returns the legend items for the plot. Depending on the data extract order, this will be the column keys or the row keys from the dataset.
```

The following method is called by the `JFreeChart` class during chart drawing:

```
► public void draw(Graphics2D g2, Rectangle2D plotArea,
Point2D anchor, PlotState parentState, PlotRenderingInfo state);
Draws the plot within the specified area.
```

In typical situations, you won't normally call this method directly.

### 33.24.5 Equals, Cloning and Serialization

The equals method is overridden:

```
► public boolean equals(Object obj);
Tests this plot for equality with an arbitrary object. This method returns true if and only if:
```

- `obj` is not `null`;
- `obj` is an instance of `MultiplePiePlot`;
- both plots have the same attributes (excluding the dataset and the registered listeners).

Instances of this class are cloneable and serializable.

### 33.24.6 Notes

Some points to note:

- several demo applications (`MultiplePieChartDemo1-4.java`) are included in the JFreeChart demo distribution.
- the `createMultiplePieChart()` and `createMultiplePieChart3D()` methods in the `ChartFactory` class create charts using this plot.

## 33.25 PieLabelDistributor

### 33.25.1 Overview

The `PiePlot` class uses this class to arrange section labels so that they do not overlap one another. This is largely an implementation detail—you won’t need to use this class directly.

## 33.26 PieLabelRecord

### 33.26.1 Overview

A temporary holder of information about the label for one section of a `PiePlot`. Instances of this class are used by the `PieLabelDistributor` class. Typically, you won’t use this class directly.

## 33.27 PiePlot

### 33.27.1 Overview

The `PiePlot` class draws pie charts using data obtained through the `PieDataset` interface. A sample chart is shown in figure 33.13.

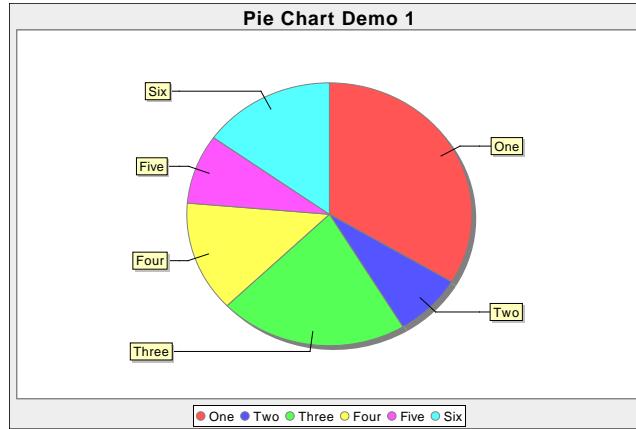


Figure 33.13: A simple pie chart (see `PieChartDemo1.java`)

Refer to chapter 5 for a general overview of the pie chart support in JFreeChart. The `PiePlot` class extends `Plot`. A subclass (`PiePlot3D`) that draws plots with a 3D effect is also available.

### 33.27.2 Constructors

To construct a pie plot:

```
➔ public PiePlot(PieDataset dataset);
```

Creates a pie plot for the given dataset (`null` is permitted). All plot attributes are initialised with default values—these can be changed at any time.

This class also has a default constructor:

```
➔ public PiePlot();
```

Creates a new plot with no dataset.

### 33.27.3 Attributes

The attributes maintained by the `PiePlot` class, which are in addition to those inherited from the `Plot` class, are listed in table 33.6.

Attribute:	Description:
<code>interiorGap</code>	The amount of space to leave blank around the outside of the pie, expressed as a percentage of the chart height and width. Extra space is added for the labels.
<code>circular</code>	A flag that controls whether the pie chart is constrained to be circular, or allowed to take on an elliptical shape to fit the available space.
<code>startAngle</code>	The angle of the first pie section, expressed in degrees (0 degrees is three o'clock, 90 degrees is twelve o'clock, 180 degrees is nine o'clock and 270 degrees is six o'clock).
<code>direction</code>	Pie sections can be ordered in a clockwise ( <code>Rotation.CLOCKWISE</code> ) or anticlockwise ( <code>Rotation.ANTI_CLOCKWISE</code> ) direction.
<code>sectionPaint</code>	The paint used for all sections (usually <code>null</code> ). <i>Deprecated as of 1.0.6.</i>
<code>sectionPaintList</code>	The paint used for each section, unless overridden by <code>sectionPaint</code> .
<code>baseSectionPaint</code>	The default paint, used when no other setting is specified.
<code>sectionOutlinesVisible</code>	A flag that controls whether or not section outlines are drawn.
<code>sectionOutlinePaint</code>	The outline paint used for all sections (usually <code>null</code> ). <i>Deprecated as of 1.0.6.</i>
<code>sectionOutlinePaintList</code>	The outline paint used for each section.
<code>baseSectionOutlinePaint</code>	The default outline paint, used when no other setting is specified.
<code>sectionOutlineStroke</code>	The outline stroke used for all sections (usually <code>null</code> ). <i>Deprecated as of 1.0.6.</i>
<code>sectionOutlineStrokeList</code>	The outline stroke used for each section.
<code>baseSectionOutlineStroke</code>	The default outline stroke, used when no other setting is specified.
<code>shadowPaint</code>	The shadow paint.
<code>shadowXOffset</code>	The x-offset for the shadow effect.
<code>shadowYOffset</code>	The y-offset for the shadow effect.
<code>explodePercentages</code>	The amount (percentage) to “explode” each pie section.
<code>labelGenerator</code>	The section label generator, an instance of <code>PieSectionLabelGenerator</code> .
<code>labelFont</code>	The font for the section labels.
<code>labelPaint</code>	The colour for the section labels.
<code>labelBackgroundPaint</code>	The background colour for the section labels.
<code>maximumLabelWidth</code>	The maximum label width as a percentage of the plot width.
<code>labelGap</code>	The gap for the section labels.
<code>labelLinkMargin</code>	The label link margin.
<code>labelLinkPaint</code>	The <code>Paint</code> used for the lines that connect the pie sections with their corresponding labels.
<code>labelLinkStroke</code>	The <code>Stroke</code> used for the lines that connect the pie sections to their corresponding labels.
<code>toolTipGenerator</code>	A plug-in tool tip generator.
<code>urlGenerator</code>	A plug-in URL generator (for image map generation).
<code>pieIndex</code>	The index for this plot (only used by the <code>MultiplePiePlot</code> class).

Table 33.6: Attributes for the `PiePlot` class

The following default values are used where necessary:

Name:	Value:
DEFAULT_INTERIOR_GAP	0.1 (10 percent)
DEFAULT_START_ANGLE	90.0
DEFAULT_LABEL_FONT	new Font("SansSerif", Font.PLAIN, 10);
DEFAULT_LABEL_PAINT	Color.black;
DEFAULT_LABEL_BACKGROUND_PAINT	new Color(255, 255, 192);
DEFAULT_LABEL_GAP	0.10 (10 percent)

### 33.27.4 The Plot Border

The `PiePlot` draws a border around the outside of the plot, with the chart title and legend appearing outside this border. This border is common to all plot types, but is especially noticeable in pie charts (and other charts that don't have axes). You can change the appearance of this border (or hide it completely) using methods inherited from the `Plot` class—refer to section 33.30.6 for details.

### 33.27.5 The Dataset

To access the dataset being used by the plot:

```
➔ public PieDataset getDataset();
    Returns the current dataset (possibly null).
➔ public void setDataset(PieDataset dataset);
    Replaces the dataset being used by the plot (this triggers a DatasetChangeEvent).
```

### 33.27.6 General Methods

To control whether the pie chart is circular or elliptical:

```
➔ public boolean isCircular();
    Returns the flag that controls whether or not the pie chart is constrained to be circular in
    appearance. The default value is true.
➔ public void setCircular(boolean flag);
    Equivalent to setCircular(flag, true)—see next method.
➔ public void setCircular(boolean circular, boolean notify);
    Sets a flag that controls whether the pie chart is circular or elliptical in shape, and sends a
    PlotChangeEvent to all registered listeners.
```

To control the position of the first section in the chart:

```
➔ public double getStartAngle();
    Returns the angle (in degrees) at which the first pie section starts. Zero is at 3 o'clock, and as
    the angle increases it proceeds anticlockwise around the chart (so that 90 degrees, the current
    default, is at 12 o'clock). This is the same encoding used by Java's Arc2D class.
➔ public void setStartAngle(double angle);
    Sets the angle (in degrees) at which the first section starts, and sends a PlotChangeEvent to all
    registered listeners.
```

To control the direction (clockwise or anticlockwise) of the sections in the pie chart:

```
➔ public Rotation getDirection();
    Returns the direction in which the pie sections are drawn. The default value is Rotation.CLOCKWISE.
➔ public void setDirection(Rotation direction);
    Sets the direction of the sections in the pie chart. Use one of the constants Rotation.CLOCKWISE
    (the default) and Rotation.ANTICLOCKWISE.
```

To control the amount of space around the pie chart:

```
➔ public double getInteriorGap();
    Returns the gap around the interior of the pie plot (the region where the labels are drawn) as
    a percentage of the plot width and height. The default value is 0.08.
➔ public void setInteriorGapPercent(double percent);
    Sets the amount of space to leave inside the plot area and sends a PlotChangeEvent to all
    registered listeners.
```

### 33.27.7 Section Paint

The paint used to fill each section in the pie chart is, by default, auto-populated from the plot's [DrawingSupplier](#). However, you can easily customise the colours using the methods described below.

To control the paint associated with a section:

- `public Paint getSectionPaint(Comparable key); [1.0.3]`  
Returns the paint associated with the specified section, which may be `null`.
- `public void setSectionPaint(Comparable key, Paint paint); [1.0.3]`  
Sets the paint to use for the specified section and sends a [PlotChangeEvent](#) to all registered listeners.

The base section paint is a default that is used when no other setting is available:

- `public Paint getBaseSectionPaint();`  
Returns the base section paint, which is never `null`. The default value is `Color.gray`.
- `public void setBaseSectionPaint(Paint paint);`  
Sets the base section paint (`null` is not permitted) and sends a [PlotChangeEvent](#) to all registered listeners.

An override setting is available, but deprecated as of JFreeChart 1.0.6 because it is more or less redundant:

- `public Paint getSectionPaint(); [Deprecated, 1.0.6]`  
Returns the paint that should be used for ALL sections in the `PiePlot`. The default value is `null`.
- `public void setSectionPaint(Paint paint); [Deprecated, 1.0.6]`  
Sets the paint that applies to ALL sections in the `PiePlot` and sends a [PlotChangeEvent](#) to all registered listeners.

The `PiePlot` drawing code makes use of the following utility methods:

- `protected Paint lookupSectionPaint(Comparable key); [1.0.3]`  
Returns the paint associated with the given key, or `null`.
- `protected Paint lookupSectionPaint(Comparable key, boolean autoPopulate); [1.0.3]`  
Returns the paint associated with the given key. If `autoPopulate` is `true` and there is currently no paint defined, a new paint is fetched from the plot's [DrawingSupplier](#).

### 33.27.8 Section Outlines

The sections in a pie plot can be drawn with or without an outline:

- `public boolean getSectionOutlinesVisible();`  
Returns `true` if section outlines should be drawn for the plot, and `false` otherwise. The default value is `true`.
- `public void setSectionOutlinesVisible(boolean visible);`  
Sets the flag that controls whether or not section outlines are drawn, and sends a [PlotChangeEvent](#) to all registered listeners.

The paint and stroke attributes used to draw the section outlines are specified via the following methods:

- `public Paint getSectionOutlinePaint(); [Deprecated, 1.0.6]`  
Returns the override value for the section outline paint. This defaults to `null`, which means the `getSectionOutlinePaint(int)` method will be called instead.
- `public void setSectionOutlinePaint(Paint paint); [Deprecated, 1.0.6]`  
Sets the override value for the section outline paint and sends a [PlotChangeEvent](#) to all registered listeners. Most of the time, you should leave this set to `null` so that the per-series and base-level settings are exposed.

```
→ public Paint getSectionOutlinePaint(int section);
Returns the outline paint to use for the specified section. If this is null, the plot will use the value returned by getBaseSectionOutlinePaint() instead.

→ public void setSectionOutlinePaint(int section, Paint paint);
Sets the paint used to outline a particular section in the chart and sends a PlotChangeEvent to all registered listeners.

→ public Paint getBaseSectionOutlinePaint();
Returns the default section outline paint, which is used when no other non-null setting is specified. The default value is Color.gray.

→ public void setBaseSectionOutlinePaint(Paint paint);
Sets the default section outline paint (null is not permitted) and sends a PlotChangeEvent to all registered listeners.
```

Similar methods are defined for the outline stroke:

```
→ public Stroke getSectionOutlineStroke(); [Deprecated, 1.0.6]
Returns the override value for the section outline stroke. This defaults to null, which means the getSectionOutlineStroke(int) method will be called instead.

→ public void setSectionOutlineStroke(Stroke stroke); [Deprecated, 1.0.6]
Sets the override value for the section outline stroke and sends a PlotChangeEvent to all registered listeners. Most of the time, you should leave this set to null so that the per-series and base-level settings are exposed.

→ public Stroke getSectionOutlineStroke(int section);
Returns the outline stroke to use for the specified section. If this is null, the plot will use the value returned by getBaseSectionOutlineStroke() instead.

→ public void setSectionOutlineStroke(int section, Stroke stroke);
Sets the stroke used to outline a particular section in the chart and sends a PlotChangeEvent to all registered listeners.

→ public Stroke getBaseSectionOutlineStroke();
Returns the default section outline stroke, which is used when no other non-null setting is specified. The default value is BasicStroke(0.5f).

→ public void setBaseSectionOutlineStroke(Stroke stroke);
Sets the default section outline stroke (null is not permitted) and sends a PlotChangeEvent to all registered listeners.
```

### 33.27.9 Shadow Effect

The pie plot will draw a “shadow” effect. To control the paint used to draw the shadow:

```
→ public Paint getShadowPaint();
Returns the paint used to draw the shadow for each pie section. If this is null, no shadow is drawn. The default value is Color.gray.

→ public void setShadowPaint(Paint paint);
Sets the paint used to draw the “shadow” effect. If you set this to null, no shadow effect will be drawn.
```

The x-offset for the shadow effect is controlled with the following methods:

```
→ public double getShadowXOffset();
Returns the x-offset for the shadow effect, in Java2D units. The default value is 4.0f.

→ public void setShadowXOffset(double offset);
Sets the x-offset (in Java2D units) for the shadow effect, and sends a PlotChangeEvent to all registered listeners.
```

The y-offset for the shadow effect is controlled by these methods:

```
→ public double getShadowYOffset();
Returns the y-offset for the shadow effect, in Java2D units. The default value is 4.0f.

→ public void setShadowYOffset(double offset);
Sets the y-offset (in Java2D units) for the shadow effect, and sends a PlotChangeEvent to all registered listeners.
```

### 33.27.10 Exploded Sections

It is possible to “explode” sections of the pie chart.

► public double getExplodePercent(Comparable key);

Returns the amount by which a specific section in the pie plot is offset, as a percentage of the radius of the pie.

► public void setExplodePercent(Comparable key, double percent);

Sets the amount by which a specific section of the pie plot is offset, expressed as a percentage of the radius of the pie, then sends a `PlotChangeEvent` to all registered listeners.

The following utility method is used internally:

► public double getMaximumExplodePercent();

Returns the maximum offset for the pie plot.

The `PieChartDemo2` application (included in the JFreeChart demo collection) provides a demo.

### 33.27.11 Section Labels

Section labels are generated by a user-definable generator object:

► public PieSectionLabelGenerator getLabelGenerator();

Returns the section label generator for the plot (possibly `null`). The default value is an instance of `StandardPieSectionLabelGenerator`.

► public void setLabelGenerator(PieSectionLabelGenerator generator);

Sets the label generator for the plot and sends a `PlotChangeEvent` to all registered listeners. If you set this to `null`, no section labels will be displayed on the plot.

For example, to display percentage values for the pie sections:

```
PiePlot plot = (PiePlot) chart.getPlot();
PieSectionLabelGenerator generator = new StandardPieSectionLabelGenerator(
    "{0} = {2}", new DecimalFormat("0"), new DecimalFormat("0.00%"));
plot.setLabelGenerator(generator);
```

To control the font used to display the section labels:

► public Font getLabelFont();

Returns the font (never `null`) used to display the section labels. The default value is `Font("SansSerif", Font.PLAIN, 10)`.

► public void setLabelFont(Font font);

Sets the font used to display the section labels, and sends a `PlotChangeEvent` to all registered listeners. If `font` is `null`, this method throws an `IllegalArgumentException`.

To control the colour of the section labels:

► public Paint getLabelPaint();

Returns the colour used to display the section labels (never `null`).

► public void setLabelPaint(Paint paint);

Sets the colour used to display the section labels and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the background colour of the section labels:

► public Paint getLabelBackgroundPaint();

Returns the colour used to fill the section label boxes—if this is `null`, the boxes are transparent (the plot background colour will show through).

► public void setLabelBackgroundPaint(Paint paint);

Sets the colour used to fill the section label boxes and sends a `PlotChangeEvent` to all registered listeners. If you set this to `null`, the label boxes will be transparent (the plot background colour will show through).

To control the outline colour of the section label boxes:

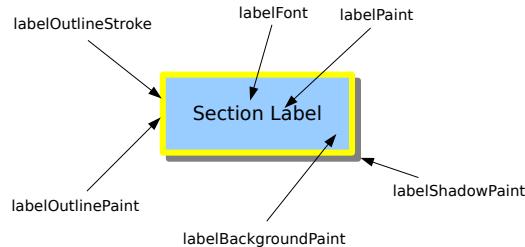


Figure 33.14: *PieSectionLabels.pdf*

► `public Paint getLabelOutlinePaint();`

Returns the colour used to draw the outline around the section labels. If this is `null`, no outline is drawn.

► `public void setLabelOutlinePaint(Paint paint);`

Sets the colour used to draw the outline around the section labels and sends a `PlotChangeEvent` to all registered listeners. If you set this to `null`, the label boxes will not have an outline drawn.

To control the outline stroke for the section label boxes:

► `public Stroke getLabelOutlineStroke();`

Returns the stroke used to draw the outline around the section labels. If this is `null`, no outline box is drawn.

► `public void setLabelOutlineStroke(Stroke stroke);`

Sets the stroke used to draw outlines around the section labels, and sends a `PlotChangeEvent` to all registered listeners. If `stroke` is `null`, no outlines will be drawn.

To control the shadow paint for the section labels:

► `public Paint getLabelShadowPaint();`

Returns the paint used to draw the shadows beneath the section label boxes. If this is `null`, no shadow is drawn. The default value is `DEFAULT_LABEL_SHADOW_PAINT` (`Color(151, 151, 151, 128)`).

► `public void setLabelShadowPaint(Paint paint);`

Sets the paint used to draw the shadows beneath the section label boxes, and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, no shadow will be drawn.

To control the padding for the section labels:

► public `RectangleInsets` `getLabelPadding()`; [1.0.7]

Returns the padding (never `null` for the section labels). This is the whitespace around the text and inside the outline.

► public void `setLabelPadding(RectangleInsets padding)`; [1.0.7]

Sets the padding for the section labels and sends a `PlotChangeEvent` to all registered listeners. If `padding` is `null`, this method throws an `IllegalArgumentException`.

The following methods allow you to plug in an alternative pie label distributor:

► public `AbstractPieLabelDistributor` `getLabelDistributor()`; [1.0.6]

Returns the object (never `null`) responsible for distributing the section labels to avoid overlapping.

► public void `setLabelDistributor(AbstractPieLabelDistributor distributor)`; [1.0.6]

Sets the object responsible for distributing the section labels and sends a `PlotChangeEvent` to all registered listeners. If `distributor` is `null`, this method throws an `IllegalArgumentException`.

### 33.27.12 Simple Label Positioning

A new “simple” labelling option has been introduced in JFreeChart 1.0.7, where the labels are drawn roughly in the center of each pie section. No attempt is made to avoid overlapping labels in the case that several small pie sections are displayed alongside each other, so use this option only for cases where that is not a concern.

► public boolean `getSimpleLabels()`; [1.0.7]

Returns the flag that controls whether the pie plot shows section labels in the “simple” format. The default value is `false`.

► public void `setSimpleLabels(boolean simple)`; [1.0.7]

Sets the flag that controls whether the pie plot shows section labels in the “simple” format, and sends a `PlotChangeEvent` to all registered listeners.

The position of the simple labels is controlled with an offset attribute:

► public `RectangleInsets` `getSimpleLabelOffset()`; [1.0.7]

Returns the insets used to calculate the simple label anchor points relative to the pie plot’s bounding rectangle. The default value is `RectangleInsets(UnitType.RELATIVE, 0.18, 0.18, 0.18, 0.18)`.

► public void `setSimpleLabelOffset(RectangleInsets offset)`; [1.0.7]

Sets the insets used to calculate the simple label anchor points relative to the pie plot’s bounding rectangle, and sends a `PlotChangeEvent` to all registered listeners.

### 33.27.13 Label Links

With regular section labels, a linking line is drawn to connect the pie section with its corresponding section label. To control whether or not these linking lines are drawn:

► public boolean `getLabelLinksVisible()`;

Returns the flag that controls whether or not the section label linking lines are visible. The default value is `true`.

► public void `setLabelLinksVisible(boolean visible)`;

Sets the flag that controls whether or not the section label linking lines are visible, and sends a `PlotChangeEvent` to all registered listeners.

To control the colour of the linking lines:

► public `Paint` `getLabelLinkPaint()`;

Returns the label link paint (never `null`). The default value is `Color.black`.

```
► public void setLabelLinkPaint(Paint paint);
Sets the Paint used for the lines connecting the pie sections to their corresponding labels and sends a PlotChangeEvent to all registered listeners. If paint is null, this method throws an IllegalArgumentException.
```

To control the line style for the linking lines:

```
► public Stroke getLabelLinkStroke();
Returns the stroke used to draw the label linking lines (never null). The default value is BasicStroke\(0.5f\).

► public void setLabelLinkStroke(Stroke stroke);
Sets the Stroke used for the lines connecting the pie sections to their corresponding labels and sends a PlotChangeEvent to all registered listeners. If stroke is null, this method throws an IllegalArgumentException.
```

The overall space allocated to the section labels and their linking lines is configurable using the following methods:

```
► public double getLabelGap();
Returns the gap between the edge of the pie and the label areas at the left and right side of the pie, as a percentage of the overall plot width. The default value is 0.025 (two-and-a-half percent).

► public void setLabelGap(double gap);
Sets the label gap and sends a PlotChangeEvent to all registered listeners.

► public double getMaximumLabelWidth();
Returns the maximum label width as a percentage of the plot width. The default value is 0.14 (fourteen percent).

► public void setMaximumLabelWidth(double width);
Sets the maximum label width (as a percentage of the plot width) and sends a PlotChangeEvent to all registered listeners.
```

The label linking line has a bend or “elbow” at a point slightly outside of the pie chart. The distance of the point from the edge of the pie chart is expressed as a percentage of the pie radius, and is referred to as the label link margin:

```
► public double getLabelLinkMargin();
Returns the label link margin, expressed as a percentage of the radius of the pie chart. The default value is 0.025 (two-and-a-half percent).

► public void setLabelLinkMargin(double margin);
Sets the label link margin, and sends a PlotChangeEvent to all registered listeners.
```

### 33.27.14 Legend Items

The legend for a pie chart is recreated each time the plot is drawn. The [JFreeChart](#) class will call the `getLegendItems()` method to create a collection of items for display in the legend. The methods below provide various configuration options for the generated legend items.

The text for each legend item is created by a label generator that can be modified:

```
► public PieSectionLabelGenerator getLegendLabelGenerator();
Returns the generator that derives the labels for the items in the legend (never null). The default value is a default instance of StandardPieSectionLabelGenerator.

► public void setLegendLabelGenerator(PieSectionLabelGenerator generator);
Sets the generator that derives the labels for the items in the legend and sends a PlotChangeEvent to all registered listeners. If generator is null, this method throws an IllegalArgumentException.
```

The shape displayed for each legend item is controlled via the following methods:

```
► public Shape getLegendItemShape();
Returns the shape displayed with each legend item (never null). The default shape is a circle with radius 4.0 (in Java2D units).
```

```
► public void setLegendItemShape(Shape shape);  
Sets the shape to be displayed with each legend item and sends a PlotChangeEvent to all registered listeners. If shape is null, this method throws an IllegalArgumentException.
```

A generator can be used to create the tool tip text for each item in the legend:

```
► public PieSectionLabelGenerator getLegendLabelToolTipGenerator();  
Returns the generator that creates the tool tips for each item in the legend. The default value is null.  
  
► public void setLegendLabelToolTipGenerator(PieSectionLabelGenerator generator);  
Sets the generator that creates the tool tips for each item in the legend and sends a PlotChangeEvent to all registered listeners. If generator is null, no tool tips will be displayed for the legend items.
```

Similarly, a generator is used to create URLs for each item in the legend (for use when creating HTML image maps):

```
► public PieURLGenerator getLegendLabelURLGenerator(); [1.0.4]  
Returns the generator that creates the URL for each item in the legend. These are only used when creating HTML image maps. The default value is null.  
  
► public void setLegendLabelURLGenerator(PieURLGenerator generator); [1.0.4]  
Sets the generator that creates the URL for each item in the legend and sends a PlotChangeEvent to all registered listeners. If generator is null, no URLs will be generated.
```

### 33.27.15 Tool Tips

If you are displaying your pie chart in a `ChartPanel`, or writing and you want to customise the tooltip text, you can register your own tool tip generator with the plot:

```
► public PieToolTipGenerator getToolTipGenerator();  
Returns the tool tip generator for the plot. The default value is null.3  
  
► public void setToolTipGenerator(PieToolTipGenerator generator);  
Registers a tool tip generator with the pie plot and sends a PlotChangeEvent to all registered listeners. You can set this to null if you do not require tooltips.
```

### 33.27.16 URLs

If you create an HTML image map for a pie chart, it is possible to assign a URL to each pie section in the chart:

```
► public PieURLGenerator getURLGenerator();  
Returns the current URL generator (possibly null) for the plot.  
  
► public void setURLGenerator(PieURLGenerator generator);  
Sets the URL generator for the plot and sends a PlotChangeEvent to all registered listeners.
```

### 33.27.17 Handling for Null and Zero Values

A couple of flags in the `PiePlot` class control how zero and `null` values in the dataset are treated by the plot:

```
► public boolean getIgnoreNullValues();  
The default value is false.  
  
► public void setIgnoreNullValues(boolean flag);  
Sets the flag that controls whether or not null values in the dataset are ignored, then sends a PlotChangeEvent to all registered listeners.  
  
► public boolean getIgnoreZeroValues();  
The default value is false.  
  
► public void setIgnoreZeroValues(boolean flag);  
Sets the flag that controls whether or not zero values in the dataset are ignored, then sends a PlotChangeEvent to all registered listeners.
```

---

<sup>3</sup>Although if you use the `ChartFactory` class to create a pie chart, it may install a tool tip generator for you.

### 33.27.18 Draw Method

The following method is called by the `JFreeChart` class during chart drawing:

```
► public void draw(Graphics2D g2, Rectangle2D area, Point2D anchor,
PlotState parentState, PlotRenderingInfo state);
Draws the plot within the specified area.
```

In typical situations, you won't normally call this method directly.

### 33.27.19 Other Methods

```
► protected Comparable getSectionKey(int section); [1.0.3]
Returns the key for the specified section.
```

An index can be assigned to a `PiePlot`—this is used by the `MultiplePiePlot` class as a way of identifying subplots, and the index is picked up the `StandardPieURLGenerator` class when generating URLs for HTML image maps:

```
► public int getPieIndex();
Returns the index that has been assigned to the plot.

► public void setPieIndex(int index);
Sets the index for the plot. This method generates no notification event.
```

As a workaround for JRE bug 4836495, a minimum arc angle is maintained by the plot—pie segments with an angle less than this value are not drawn:

```
► public double getMinimumArcAngleToDraw();
Returns the minimum angle for drawing the arc segment for a pie section. The default value is 0.00001.

► public void setMinimumArcAngleToDraw(double angle);
Sets the minimum angle for drawing a pie segment. Any segment with an angle less than this value will not be drawn. This is a workaround for a JRE bug:

• see http://bugs.sun.com/bugdatabase/view\_bug.do?bug\_id=4836495
```

### 33.27.20 Notes

Some points to note:

- chapter 5 provides a general overview of the pie chart support in JFreeChart;
- there are several methods in the `ChartFactory` class that will construct a default pie chart for you;
- the `DatasetUtilities` class has methods for creating a `PieDataset` from a `CategoryDataset`;
- the `PieChartDemo1` class in the `org.jfree.chart.demo` package provides a simple pie chart demonstration (plus, there are more demos included in the JFreeChart demo collection).
- the default section label format changed between version 1.0.1 and 1.0.2 of JFreeChart—in later versions, only the section key (and not the value) is displayed;
- some label layout bug fixes included from version 1.0.8 onwards have resulted in the chart dimensions changing—you may need to adjust your code for this.

#### See Also

`PieDataset`, `PieSectionLabelGenerator`, `PieToolTipGenerator`, `Plot`.

## 33.28 PiePlot3D

### 33.28.1 Overview

An extension of the `PiePlot` class that draws pie charts with a 3D effect—see figure 33.15 for an example. The `ChartFactory` class has a `createPieChart3D()` method that you can use to create a ready-made `JFreeChart` instance containing a `PiePlot3D`.

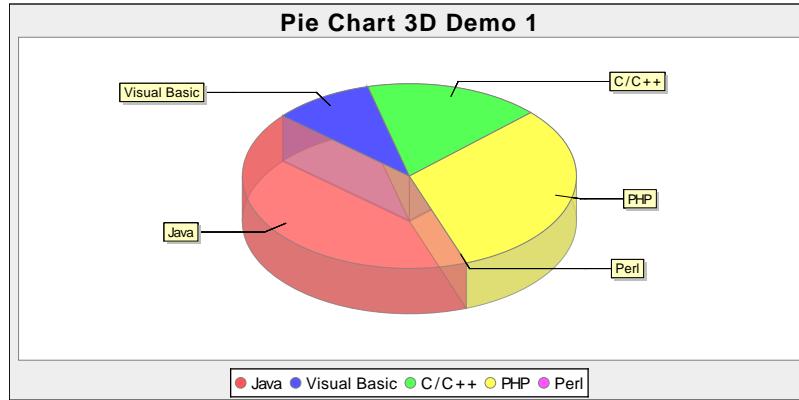


Figure 33.15: A pie chart with 3D effect (see `PieChart3DDemo1.java`)

### 33.28.2 Limitations

To avoid over-selling this plot type, let's point out that it has some limitations:

- the 3D effect is drawn using plain 2D graphics primitives—there is no 3D engine performing a perspective transformation, no ability to control the viewing angle, etc. The effect looks fine when the plot is wider than it is tall, but resizing the chart can result in the perspective looking wrong;
- the plot does not support the “exploded” sections that the regular `PiePlot` class supports;

For now, we suggest that you avoid this class unless you can live with its limitations. In the future, we hope to reimplement this class using a proper 3D graphics engine.

### 33.28.3 Constructor

To create a new instance:

► `public PiePlot3D();`  
Equivalent to `PiePlot3D(null)`—see the next constructor.

► `public PiePlot3D(PieDataset dataset);`  
Creates a new plot with the specified dataset (`null` is permitted).

### 33.28.4 General Attributes

This class inherits most of its attributes from the `PiePlot` class.

The depth factor specifies the size of the 3D effect as a percentage of the height of the plot area:

► `public double getDepthFactor();`  
Returns the depth factor for the 3D effect, as a percentage of the height of the pie plot. The default value is 0.12 (twelve percent).

► `public void setDepthFactor(double factor);`  
 Sets the depth factor (as a percentage of the height of the pie plot) and sends a `PlotChangeEvent` to all registered listeners.

A new flag has been introduced in JFreeChart 1.0.7 to control whether or not the sides of the pie plot are drawn in a darker colour—this only works when the `sectionPaint` is an instance of `java.awt.Color`:

► `public boolean getDarkerSides(); [1.0.7]`  
 Returns the flag that controls whether or not the sides of the pie plot are drawn using a darker colour. The default is `false` (to preserve the existing behaviour).

► `public void setDarkerSides(boolean darker); [1.0.7]`  
 Sets the flag that controls whether or not the sides of the pie plot are drawn using a darker colour, and sends a `PlotChangeEvent` to all registered listeners.

### 33.28.5 Other Methods

The other methods are intended for use by JFreeChart—you won’t normally call these methods directly:

► `public String getPlotType();`  
 Returns a localised string describing the plot type. This can be used for display to end-users (for example, in the property editors).

The `JFreeChart` class will call the following method to draw the plot:

► `public void draw(...);`  
 Draws the plot within a specified area.

► `protected void drawSide(...);`  
 Draws the sides for one segment.

### 33.28.6 Equals, Cloning and Serialization

This class overrides the `equals()` method.<sup>4</sup>

► `public boolean equals(Object obj);`  
 Tests this plot for equality with an arbitrary object. If `obj` is `null`, this method returns `false`.

Instances of this class are `Cloneable` and `Serializable`.

### 33.28.7 Notes

Some points to note:

- the translucent appearance of the sample chart is achieved by setting the plot’s foreground alpha to `0.5f`—see the `setForegroundAlpha()` method in the `Plot` class;
- some demos (`PieChart3DDemo1-3.java`) are included in the JFreeChart demo collection.
- additional information is provided in section 5.8;

#### See Also

[PiePlot](#).

---

<sup>4</sup> At least, as of version 1.0.5 it does!

## 33.29 PiePlotState

### 33.29.1 Overview

A class that records temporary state information during the drawing of a pie chart. This allows one instance of a [PiePlot](#) to be drawn to multiple targets simultaneously (for example, a chart might be drawn on the screen at the same time it is being saved to a file).

## 33.30 Plot

### 33.30.1 Overview

An abstract base class that controls the visual representation of data in a chart. The [JFreeChart](#) class maintains a reference to a [Plot](#), and will provide it with an area in which to draw itself (after allocating space for the chart titles and legend).

A range of subclasses are used to create different types of charts:

- [CategoryPlot](#) – for bar charts and other plots where one axis displays categories and the other axis displays values;
- [MeterPlot](#) – dials, thermometers and other plots that display a single value;
- [PiePlot](#) – for pie charts;
- [XYPlot](#) – for line charts, scatter plots, time series charts and other plots where both axes display numerical (or date) values;

Figure 33.16 illustrates the plot class hierarchy.

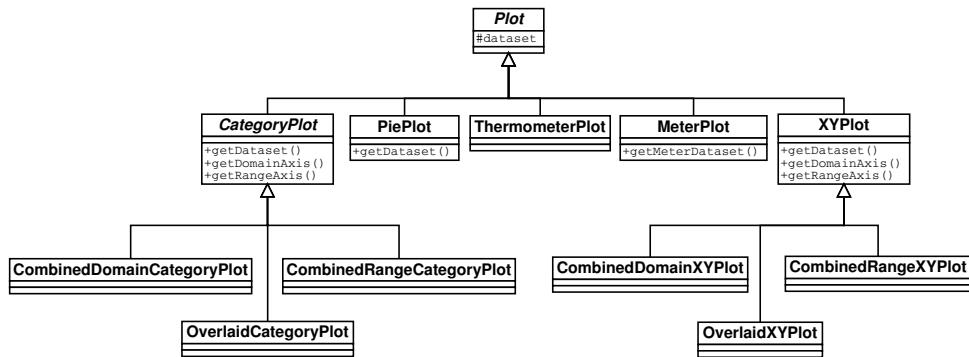


Figure 33.16: Plot classes

When a chart is drawn, the [JFreeChart](#) class first draws the title (or titles) and legend. Next, the plot is given an area (the *plot area*) into which it must draw a representation of its dataset. This function is implemented in the `draw()` method, each subclass of [Plot](#) takes a slightly different approach.

### 33.30.2 Constructors

This class is abstract, so the constructor is `protected`. You cannot create an instance of this class directly, you must use a subclass.

Attribute:	Description:
parent	The parent plot (possibly <code>null</code> ).
datasetGroup	The dataset group (not used).
insets	The amount of space to leave around the outside of the plot.
outlineStroke	The <code>Stroke</code> used to draw an outline around the plot area.
outlinePaint	The <code>Paint</code> used to draw an outline around the plot area.
backgroundPaint	The <code>Paint</code> used to draw the background of the plot area.
backgroundImage	An image that is displayed in the background of the plot (can be <code>null</code> ).
backgroundImageAlignment	The image alignment.
backgroundAlpha	The alpha transparency value used when coloring the plot's background, and also when drawing the background image (if there is one).
foregroundAlpha	The alpha transparency used to draw items in the plot's foreground.
noDataMessage	A string that is displayed by some plots when there is no data to display.
noDataMessageFont	The <code>Font</code> used to display the “no data” message.
noDataMessagePaint	The <code>Paint</code> used to display the “no data” message.
drawingSupplier	The drawing supplier (provides default colors and line strokes).

Table 33.7: Attributes for the `Plot` class

### 33.30.3 Attributes

This class maintains the attributes listed in table 33.7.

All subclasses will inherit these core attributes.

### 33.30.4 Usage

To customise the appearance of a plot, you first obtain a reference to the plot as follows:

```
Plot plot = chart.getPlot();
```

With this reference, you can change the appearance of the plot by modifying it's attributes. For example:

```
plot.setBackgroundPaint(Color.lightGray);
plot.setNoDataMessage("There is no data.");
```

Very often, you will find it necessary to cast the `Plot` object to a specific subclass so that you can access attributes that are defined by the subclass. Refer to the usage notes for each subclass for more details.

### 33.30.5 The Plot Type

The following method returns a string indicating the plot type:

```
➔ public abstract String getPlotType();
```

Returns a string indicating the plot type. This method must never return `null`. The string can be localised (that is, a different string may be returned for each locale).

The method is abstract, and subclasses are required to implement it.

### 33.30.6 The Plot Border

An border is drawn around the outside of most plot types. You can change the appearance of the border by modifying the `Paint` and `Stroke` used to draw it (or set either to `null` to hide the border completely).

```
➔ public Paint getOutlinePaint();
```

Returns the paint used to draw the plot outline. The default value is `Color.GRAY`.

```
→ public void setOutlinePaint(Paint paint);
Sets the paint used to draw the plot outline and sends a PlotChangeEvent to all registered listeners. If you set this to null, no outline will be drawn.

→ public Stroke getOutlineStroke();
Returns the stroke used to draw the plot outline. The default value is BasicStroke(0.5f).

→ public void setOutlineStroke(Stroke stroke);
Sets the stroke used to draw the plot outline and sends a PlotChangeEvent to all registered listeners. If you set this to null, no outline will be drawn.
```

For the `CategoryPlot` and `XYPLOT` classes, the outline is drawn around all four sides of the data area. Note that each of the plot's axes may also draw a line along the edge of the data area—see the `axisLineVisible` attribute defined in the `Axis` class.

### 33.30.7 The Plot Background

The *background area* for a plot is the area inside the plot's axes (if the plot has axes)—it does not include the chart titles, the legend or the axis labels.

The plot's background is filled with the `backgroundPaint`:

```
→ public Paint getBackgroundPaint();
Returns the paint used to fill the plot's background, or null if the plot has a transparent background. The default value is Color.WHITE.

→ public void setBackgroundPaint(Paint paint);
Sets the background paint for the plot and sends a PlotChangeEvent to all registered listeners. You can set this attribute to null for a transparent plot background (in this case, the chart's background is visible—see the JFreeChart class for more information).
```

The alpha transparency for the plot's background is configurable:

```
→ public float getBackgroundAlpha();
Returns the alpha transparency for the plot's background. The default value is 1.0f (opaque).

→ public void setBackgroundAlpha(float alpha);
Sets the alpha transparency for the plot's background and sends a PlotChangeEvent to all registered listeners. The range of values permitted is 0.0f (fully transparent) through to 1.0f (fully opaque). If the background is at all transparent, you will be able to see the chart's background showing through.
```

You can also add an image to the background area:

```
→ public Image getBackgroundImage();
Returns the background image for the plot. The default value is null.

→ public void setBackgroundImage(Image image);
Sets the background image for the plot5 and sends a PlotChangeEvent to all registered listeners. If image is null, no background image will be drawn.
```

A number of options are provided for aligning the background image:

```
→ public int getBackgroundImageAlignment();
Returns the alignment type for the background image. The default values is Align.FIT (which means the image is stretched to fit the available space).
```

To modify the alignment, use the following method:

```
→ public void setBackgroundImageAlignment(int alignment);
Sets the alignment for the background image and sends a PlotChangeEvent to all registered listeners. For the alignment argument, use one of the predefined constants in the Align class from the JCommon class library: CENTER, TOP, BOTTOM, LEFT, RIGHT, TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, BOTTOM_RIGHT, FIT_HORIZONTAL, FIT_VERTICAL and FIT (stretches to fill the entire area).
```

<sup>5</sup>Take care that the image supplied is actually loaded into memory. The `createImage()` method in Java's `Toolkit` class will load images asynchronously, which can result in a chart being drawn before the background image is available—see section 20.4 for more information.

The background image can be drawn using an alpha-transparency, you can set this as follows:

```
► public float getBackgroundImageAlpha();
Returns the alpha transparency for drawing the background image.

► public void setBackgroundImageAlpha(float alpha);
Sets the alpha transparency for drawing the background image, and sends a PlotChangeEvent
to all registered listeners.
```

There are similar methods in the `JFreeChart` class that allow you to control the background area for the chart (which encompasses the entire chart area).

### 33.30.8 The Plot Insets

The space around the outside of the plot is controlled by the plot insets:

```
► public RectangleInsets getInsets();
Returns the insets for the plot (never null). The default value is RectangleInsets(4.0, 8.0,
4.0, 8.0).

► public void setInsets(RectangleInsets insets);
Equivalent to setInsets(insets, true)—see next method.

► public void setInsets(RectangleInsets insets, boolean notify);
Sets the insets for the plot and, if notify is true, sends a PlotChangeEvent to all registered
listeners. If insets is null, this method throws an IllegalArgumentException.
```

### 33.30.9 The Drawing Supplier

The “drawing supplier” is a plug-in object responsible for providing a never-ending sequence of `Paint` and `Stroke` objects for the plot and its renderers. A default instance is installed for every plot automatically, but you can provide a custom supplier if you need to:

```
► public DrawingSupplier getDrawingSupplier();
Returns the drawing supplier for the plot (or the plot’s parent if this is a subplot).

► public void setDrawingSupplier(DrawingSupplier supplier);
Sets the drawing supplier and sends a PlotChangeEvent to all registered listeners. A null supplier
is not permitted.
```

In cases where there is a hierarchy of plots, the intention is that the drawing supplier of the root plot is shared by all the subplots—this ensures that colors are not duplicated by subplots.

### 33.30.10 The Parent Plot

Some plot classes (such as the `CombinedDomainXYPlot` class) manage a number of child plots. A child plot can access its parent plot via the following method:

```
► public Plot getParent();
Returns the parent plot, or null if this plot has no parent.

► public void setParent(Plot parent);
Sets the parent plot for this plot. This method is intended for use by JFreeChart, you shouldn’t
need to call it directly. In fact, setting the parent plot incorrectly will corrupt your chart.
```

If a plot has a parent, then it is sometimes referred to as a *subplot*. The following method can be used to determine if a plot is a subplot:

```
► public boolean isSubplot();
Returns true if this plot is a subplot.
```

To determine the plot at the root of a hierarchy of plots, use the following method:

```
► public Plot getRootPlot()
Returns this plot if there is no parent plot, otherwise returns the root plot for this plot’s parent.
```

### 33.30.11 The Dataset Group

The `datasetGroup` attribute is not currently used. It was originally intended to provide a single object on which to synchronise access to multiple datasets, for those plots that use more than one dataset. However, this has never been implemented.

```
► public DatasetGroup getDatasetGroup();
Returns the dataset group for the plot. This is not used.

► protected void setDatasetGroup(DatasetGroup group);
Sets the dataset group for the plot. This is not used.
```

### 33.30.12 The “No Data” Message

Some plots will display a message when no data is available for plotting. The message itself is a simple string, controlled via the following methods:

```
► public String getNoDataMessage();
Returns the string displayed by some plots when no data is available. The default value is null.

► public void setNoDataMessage(String message);
Sets the string displayed by some plots when no data is available and sends a PlotChangeEvent to all registered listeners.
```

To control the font:

```
► public Font getNoDataMessageFont();
Returns the font used to display the “no data” message (never null). The default value is Font("SansSerif", Font.PLAIN, 12).

► public void setNoDataMessageFont(Font font);
Sets the font used to display the “no data” message and sends a PlotChangeEvent to all registered listeners. If font is null, this method throws an IllegalArgumentException.
```

To control the paint:

```
► public Paint getNoDataMessagePaint();
Returns the paint used to display the “no data” message (never null). The default value is Color.BLACK.

► public void setNoDataMessagePaint(Paint paint);
Sets the paint used to display the “no data” message and sends a PlotChangeEvent to all registered listeners. If paint is null, this method throws an IllegalArgumentException.
```

### 33.30.13 Legend Items

The following method returns a new collection of legend items for the plot:

```
► public LegendItemCollection getLegendItems();
This implementation returns null—subclasses should override this method.
```

### 33.30.14 Other Methods

The `JFreeChart` class expects every plot to implement the `draw()` method, and uses this to draw the plot in a specific area via a `Graphics2D` instance. You won’t normally need to call this method yourself:

```
► public abstract void draw(Graphics2D g2, Rectangle2D area,
Point2D anchor, PlotState parentState, PlotRenderingInfo info);
Draws the chart using the supplied Graphics2D. The plot should be drawn within the plotArea.
```

If you wish to record details of the items drawn within the plot, you need to supply a `ChartRenderingInfo` object. Once the drawing is complete, this object will contain a lot of information about the plot. If you don’t want this information, pass in `null`.

The following method (only used by plots that have axes) receives notification of axis changes:

```
► public void axisChanged(AxisChangeEvent event);  
Fires a PlotChangeEvent.
```

The following method receives notification whenever a [Marker](#) managed by the plot is modified:

```
► public void markerChanged(MarkerChangeEvent event); [1.0.3]  
Called whenever a marker managed by this plot is changed. In response, this method fires a  
PlotChangeEvent which, by default, will be received by the chart that owns this plot.
```

### 33.30.15 Notes

Refer to specific subclasses for information about setting the colors, shapes and line styles for data drawn by the plot.

## 33.31 PlotOrientation

### 33.31.1 Overview

Used to represent the orientation of a plot (in particular, the [CategoryPlot](#) and [XYPlot](#) classes). There are two values, as listed in table 33.8.

Class:	Description:
<code>PlotOrientation.HORIZONTAL</code>	A “horizontal” orientation.
<code>PlotOrientation.VERTICAL</code>	A “vertical” orientation.

*Table 33.8: Plot orientation values*

The orientation corresponds to the “direction” of the range axis. So, for example, a bar chart with a vertical orientation will display vertical bars, while a bar chart with a horizontal orientation will display horizontal bars.

### 33.31.2 Notes

Some points to note:

- for interesting effects, in addition to changing the orientation of a chart you can:
  - change the location of the chart’s axes—see the `setAxisLocation()` methods in the plot classes;
  - invert the scale of the axes—see the `setInverted(boolean)` method in the axis classes.
- two demos (`PlotOrientationDemo1.java` and `PlotOrientationDemo2.java`) are included in the JFreeChart demo collection.

## 33.32 PlotRenderingInfo

### 33.32.1 Overview

This class is used to record information about the individual elements in a single rendering of a plot. Typically, you will obtain one of these from the [ChartRenderingInfo](#) class—it is not a class that you are likely to need to instantiate yourself.

### 33.32.2 Constructor

To create a new instance:

► public PlotRenderingInfo(ChartRenderingInfo owner);

Creates a new instance belonging to the specified `ChartRenderingInfo` instance. You can supply a `null` owner (JFreeChart does this internally when a temporary instance is required).

### 33.32.3 Methods

To find the owner of this instance:

► public ChartRenderingInfo getOwner();

Returns the `ChartRenderingInfo` instance that owns this `PlotRenderingInfo` instance. This may be `null`.

To access the plot area:

► public Rectangle2D getPlotArea();

Returns the area (in Java2D space) in which the plot is drawn.

► public void setPlotArea(Rectangle2D area);

Sets the area (in Java2D space) into which the plot has been drawn. This method is used internally by JFreeChart, you shouldn't need to call it yourself (unless you are developing your own plot class).

To access the data area:

► public Rectangle2D getDataArea();

Returns the area (in Java2D space) in which the data is drawn (that is, the area "inside" the axes).

► public void setDataArea(Rectangle2D area);

Sets the area (in Java2D space) into which the data items are drawn. This method is used internally by JFreeChart, you shouldn't need to call it yourself (unless you are developing your own plot class).

### 33.32.4 Subplot Info

Some plots (such as `CombinedDomainXYPlot`) manage a number of subplots, and there will be a `PlotRenderingInfo` instance for each of these subplots. The following methods provide access to these instances:

► public int getSubplotCount();

Returns the number of subplots (possibly zero).

► public void addSubplotInfo(PlotRenderingInfo info);

Adds a `PlotRenderingInfo` instance for a subplot.

► public PlotRenderingInfo getSubplotInfo(int index);

Returns a `PlotRenderingInfo` instance for the specified subplot.

► public int getSubplotIndex(Point2D source);

Returns the index of the subplot (if any) at the specified location in Java2D space. If `source` is `null`, this method throws an `IllegalArgumentException`.

### 33.32.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);

Tests this instance for equality with an arbitrary object.

► public Object clone() throws CloneNotSupportedException;

Returns a clone of this instance. This is a deep clone except for the `owner` field, which is simply copied as a reference.

## 33.33 PlotState

### 33.33.1 Overview

A class that records temporary state information during the drawing of a chart. This allows a single chart instance to be drawn to multiple targets simultaneously (for example, a chart might be drawn on the screen at the same time it is being saved to a file).

## 33.34 PlotUtilities

### 33.34.1 Overview

A class containing static utility methods related to plots. This class was first introduced in JFreeChart version 1.0.7.

### 33.34.2 Methods

To determine if a plot contains no data:

```
➔ public static boolean isEmptyOrNull(XYPlot plot); [1.0.7]
```

Returns `true` if the datasets for the specified plot are either empty or `null`, and `false` otherwise.

**See Also:**

[DatasetUtilities](#)

## 33.35 PolarPlot

### 33.35.1 Overview

A plot that is used to display data from an `XYDataset` using polar coordinates—see figure 33.17 for an example.

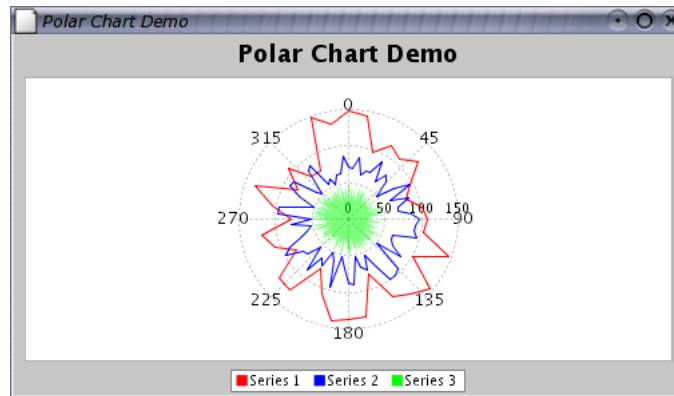


Figure 33.17: A polar chart

The items in the plot are drawn by a [PolarItemRenderer](#).

### 33.35.2 Usage

A demo application (`PolarChartDemo1.java`) is included in the [JFreeChart demo collection](#).

### 33.35.3 Constructors

To create a new plot:

```
↳ public PolarPlot();
```

Creates a new plot with no dataset, axis or renderer. If you use this constructor, you will need to supply a plot, dataset and renderer separately.

```
↳ public PolarPlot(XYDataset dataset, ValueAxis radiusAxis,
PolarItemRenderer renderer);
```

Creates a new polar plot with the specified dataset, axis and renderer. The x-values in the dataset should be in the range 0-360 degrees. The axis is typically an instance of `NumberAxis` and the renderer is typically an instance of `DefaultPolarItemRenderer`.

Note that a convenience method (`createPolarChart()`) for creating charts based on this plot is provided in the `ChartFactory` class.

### 33.35.4 Axis, Dataset and Renderer

This plot supports a single axis (the range or y-axis), dataset and renderer. To access the axis:

```
↳ public ValueAxis getAxis();
```

Returns the axis that provides the value scale for the plot.

```
↳ public void setAxis(ValueAxis axis);
```

Sets the axis that provides the value scale for the plot and sends a `PlotChangeEvent` to all registered listeners. The axis will extend from the center of the plot towards the right hand side of the chart.

To access the dataset:

```
↳ public XYDataset getDataset();
```

Returns the dataset for the plot (possibly `null`). Note that this plot only allows for a single dataset, unlike some other plots in JFreeChart.

```
↳ public void setDataset(XYDataset dataset);
```

Sets the dataset for the plot (`null` is permitted). This method sends a `DatasetChangeEvent` to the plot, which in turn generates a `PlotChangeEvent` for all registered listeners.

To access the renderer:

```
↳ public PolarItemRenderer getRenderer();
```

Returns the current renderer. If the renderer is `null`, no data will be displayed.

```
↳ public void setRenderer(PolarItemRenderer renderer);
```

Sets the renderer and sends a `PlotChangeEvent` to all registered listeners. If you set the renderer to `null`, no data will be displayed.

### 33.35.5 Angle Gridlines

The “angle gridlines” are the (optional) lines radiating out from the center of the chart. These are hard-coded (at present) to appear at 45 degree intervals. You can control whether or not the labels for the gridlines are visible with the following methods:

```
↳ public boolean isAngleLabelsVisible();
```

Returns the flag that controls whether or not the angle labels are visible.

```
↳ public void setAngleLabelsVisible(boolean visible);
```

Sets the flag that controls whether or not the angle labels are visible and sends a `PlotChangeEvent` to all registered listeners. If the new flag value is the same as the old flag value, this method does nothing.

The font used to display the labels:

```
↳ public Font getAngleLabelFont();
```

Returns the font for the angle labels (never `null`).

► **public void setAngleLabelFont(Font font);**  
 Sets the font for the angle labels and sends a [PlotChangeEvent](#) to all registered listeners. An exception is thrown if `font` is `null`.

The (foreground) paint used to display the labels:

► **public Paint getAngleLabelPaint();**  
 Returns the paint for the angle labels (never `null`).

► **public void setAngleLabelPaint(Paint paint);**  
 Sets the paint for the angle labels and sends a [PlotChangeEvent](#) to all registered listeners. An exception is thrown if `paint` is `null`.

To control whether or not the angle gridlines are displayed:

► **public boolean isAngleGridlinesVisible();**  
 Returns the flag that controls whether or not the angle gridlines are displayed.

► **public void setAngleGridlinesVisible(boolean visible);**  
 Sets the flag that controls whether or not the angle gridlines are visible and sends a [PlotChangeEvent](#) to all registered listeners. If the new flag value is the same as the old flag value, this method does nothing.

The stroke and paint used for the gridlines is controlled with the following methods:

► **public Stroke getAngleGridlineStroke();**  
 Returns the stroke used to display the angle gridlines (never `null`).

► **public void setAngleGridlineStroke(Stroke stroke);**  
 Sets the stroke used to display the angle gridlines and sends a [PlotChangeEvent](#) to all registered listeners. An exception is thrown if `stroke` is `null`.

► **public Paint getAngleGridlinePaint();**  
 Returns the paint used to display the angle gridlines (never `null`).

► **public void setAngleGridlinePaint(Paint paint);**  
 Sets the paint used to display the angle gridlines and sends a [PlotChangeEvent](#) to all registered listeners. An exception is thrown if `paint` is `null`.

### 33.35.6 Radius Gridlines

The “radius gridlines” are drawn as circles at a regular interval that is controlled by the size of the tick unit on the plot’s axis.

► **public boolean isRadiusGridlinesVisible();**  
 Returns the flag that controls whether or not the radius gridlines are drawn.

► **public void setRadiusGridlinesVisible(boolean visible);**  
 Sets the flag that controls whether or not the radius gridlines are visible and sends a [PlotChangeEvent](#) to all registered listeners. If the new flag value is the same as the old flag value, this method does nothing.

► **public Stroke getRadiusGridlineStroke();**  
 Returns the radius gridline stroke (never `null`).

► **public void setRadiusGridlineStroke(Stroke stroke);**  
 Sets the stroke used to draw the radius gridlines and sends a [PlotChangeEvent](#) to all registered listeners. An exception is thrown if `stroke` is `null`.

► **public Paint getRadiusGridlinePaint();**  
 Returns the radius gridline paint (never `null`).

► **public void setRadiusGridlinePaint(Paint paint);**  
 Sets the paint used to draw the radius gridlines and sends a [PlotChangeEvent](#) to all registered listeners. An exception is thrown if `paint` is `null`.

### 33.35.7 Corner Text Items

This plot provides an option to add one or more short text items (called “corner text items”) to the lower right corner of the plot:

```
→ public void addCornerTextItem(String text);
    Adds a small text item to be displayed at the bottom right corner of the plot and sends a
    PlotChangeEvent to all registered listeners. An exception is thrown if text is null.

→ public void removeCornerTextItem(String text);
    Removes the specified item from the list of corner text items (if the item is not in the list, this
    method does nothing) and sends a PlotChangeEvent to all registered listeners.

→ public void clearCornerTextItems();
    Removes all corner text items from the list and sends a PlotChangeEvent to all registered listeners.
```

### 33.35.8 Drawing Methods

The following method is called by the `JFreeChart` class during chart drawing:

```
→ public void draw(Graphics2D g2, Rectangle2D plotArea,
    Point2D anchor, PlotState parentState, PlotRenderingInfo state);
    Draws the plot within the specified area.
```

In typical situations, you won’t normally call this method directly. Likewise for these other drawing methods:

```
→ protected void drawCornerTextItems(Graphics2D g2, Rectangle2D area);
    Draws the corner text items (in the lower right corner of the plot).

→ protected AxisState drawAxis(Graphics2D g2, Rectangle2D plotArea, Rectangle2D dataArea);
    Draws the radial axis. This extends from the center of the plot out towards the right hand side
    of the chart.

→ protected void render(Graphics2D g2, Rectangle2D dataArea, PlotRenderingInfo info);
    Draws the data values on the chart using the current renderer.

→ protected void drawGridlines(Graphics2D g2, Rectangle2D dataArea, List angularTicks, List
    radialTicks);
    Draws the gridlines for the chart.
```

### 33.35.9 Zooming Methods

This plot supports zooming for the range axis only. Most of the methods documented below belong to the `Zoomable` interface, but because the plot has only one axis, some of the methods do nothing. The objective of these methods is to support the zooming mechanism provided by the `ChartPanel` class.

```
→ public PlotOrientation getOrientation();
    Returns PlotOrientation.HORIZONTAL always. This method is required by the Zoomable interface,
    but not used by this class.

→ public boolean isDomainZoomable();
    Returns false, because there is no domain axis to zoom.

→ public boolean isRangeZoomable();
    Returns true to indicate that the range axis is zoomable.

→ public void zoom(double percent);
    Zooms in or out by the specified amount. Values less than 1.0 reduce the axis range (“zoom
    in”) and values greater than 1.0 expand the axis range (“zoom out”).

→ public void zoomRangeAxes(double factor, PlotRenderingInfo state, Point2D source);
    Zooms the range axis.
```

```

→ public void zoomRangeAxes(double lowerPercent, double upperPercent,
    PlotRenderingInfo state, Point2D source);
Zooms the range axis.

→ public void zoomDomainAxes(double factor, PlotRenderingInfo state, Point2D source);
This method does nothing, since the plot has no domain axes.

→ public void zoomDomainAxes(double lowerPercent, double upperPercent,
    PlotRenderingInfo state, Point2D source);
This method does nothing, since the plot has no domain axes.

```

### 33.35.10 Other Methods

The remaining methods in this class are:

```

→ public String getPlotType();
Returns a short (localised) string describing the plot type.

→ public int getSeriesCount();
A convenience method that returns the number of series in the plot's dataset (or zero if the
dataset is null).

→ public Range getDataRange(ValueAxis axis);
Returns the range of y-values for the specified axis. For this plot (which has only one axis and
one dataset) this is the range of y-values in the plot's dataset.

→ public LegendItemCollection getLegendItems();
Returns the legend items for the plot. This method is called by the chart drawing code, you
won't normally need to call it yourself. You can override this method to alter the items that
are displayed in the legend.

```

The plot registers itself with its dataset and receives notification of any changes to the dataset via the following method:

```

→ public void datasetChanged(DataSetChangeEvent event);
This method is called whenever the plot's dataset is updated. You won't normally need to call
this method directly.

```

Likewise, the plot registers itself with its renderer and receives notification of any changes to the renderer via the following method:

```

→ public void rendererChanged(RendererChangeEvent event);
This method is called whenever the plot's renderer is updated. You won't normally need to
call this method directly.

```

### 33.35.11 Equals, Cloning and Serialization

To test a plot for equality with an arbitrary object:

```

→ public boolean equals(Object obj);
Returns true if this plot is equal to obj and false otherwise.

```

To create a clone of the plot:

```

→ public Object clone() throws CloneNotSupportedException;
Returns a clone of the plot (note that the dataset is not cloned).

```

This class implements `Serializable`.

### 33.35.12 Notes

Some points to note:

- this plot does not support multiple axes, datasets or renderers;
- a demo (`PolarChartDemo1.java`) is included in the JFreeChart demo collection.

## 33.36 RainbowPalette

### 33.36.1 Overview

A rainbow palette (extends [ColorPalette](#)) used by the [ContourPlot](#) class. *This class is deprecated as of version 1.0.4.*

## 33.37 RingPlot

### 33.37.1 Overview

A *ring plot* is an adaptation of a pie plot, where a hole is left in the middle of the “pie”—see figure [33.18](#) for an example. This class extends the [PiePlot](#) class.

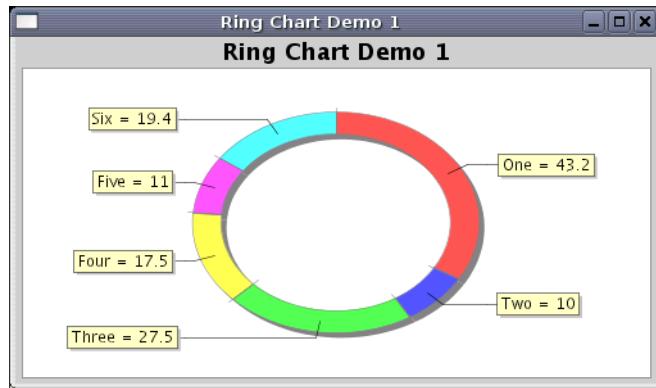


Figure 33.18: A ring chart

### 33.37.2 Constructors

The default constructor:

```
→ public RingPlot();  
Creates a new plot with null for the dataset.
```

To create a new plot with a given dataset:

```
→ public RingPlot(PieDataset dataset);  
Creates a new plot with the specified dataset (null is permitted).
```

### 33.37.3 Section Depth

The depth (or thickness) of the ring is configurable with the following methods:

```
→ public double getSectionDepth(); [1.0.3]  
Returns the section depth as a percentage of the plot's radius. The default value is 0.20 (twenty percent).  
  
→ public void setSectionDepth(double sectionDepth); [1.0.3]  
Sets the section depth and sends a PlotChangeEvent to all registered listeners.
```

### 33.37.4 Separators

The plot can draw lines to highlight the separation between sections. The separators have a number of attributes that can be customised:

- *visibility* – you can display or hide the separators;
- *stroke* – the line style for the separator lines;
- *paint* – the color for the separator lines;
- *inner extension* – the amount by which the separator lines extend into the interior of the plot;
- *outer extension* – the amount by which the separator lines extend outside the plot.

These attributes are controlled by the following methods:

- ▶ `public boolean getSeparatorsVisible();`  
Returns `true` if the separators between sections are visible, and `false` otherwise. The default value is `true`.
- ▶ `public void setSeparatorsVisible(boolean visible);`  
Sets the flag that controls whether or not the separators between sections are visible, and sends a `PlotChangeEvent` to all registered listeners.
- ▶ `public Stroke getSeparatorStroke();`  
Returns the stroke used to draw the separator lines, if they are visible. This method never returns `null`. The default value is `BasicStroke(0.5f)`.
- ▶ `public void setSeparatorStroke(Stroke stroke);`  
Sets the stroke used to draw the separator lines (`null` not permitted) and sends a `PlotChangeEvent` to all registered listeners.
- ▶ `public Paint getSeparatorPaint();`  
Returns the paint used to draw the separator lines, if they are visible. This method never returns `null`. The default value is `Color.gray`.
- ▶ `public void setSeparatorPaint(Paint paint);`  
Sets the paint used to draw the separator lines (`null` not permitted) and sends a `PlotChangeEvent` to all registered listeners.
- ▶ `public double getInnerSeparatorExtension();`  
Returns the length of the separator line drawn inside the ring for each section. The value is a percentage of the ring depth (the default is 0.20).
- ▶ `public void setInnerSeparatorExtension(double percent);`  
Sets the length of the inner separator line as a percentage of the ring depth and sends a `PlotChangeEvent` to all registered listeners.
- ▶ `public double getOuterSeparatorExtension();`  
Returns the length of the outer separator line for each section, as a percentage of the ring depth. The default value is 0.20 (twenty percent).
- ▶ `public void setOuterSeparatorExtension(double percent);`  
Sets the length of the outer separator line as a percentage of the ring depth, and sends a `PlotChangeEvent` to all registered listeners.

### 33.37.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- ▶ `public boolean equals(Object obj);`  
Tests this plot for equality with an arbitrary object. Note that the plot's dataset is NOT considered in the equality test.

This class is `Cloneable` and `Serializable`.

### 33.37.6 Notes

Some points to note:

- this plot only supports a single ring. Support for multiple concentric rings would be an interesting addition to JFreeChart;
- a demo (`RingChartDemo1.java`) is included in the JFreeChart demo collection.

## 33.38 SeriesRenderingOrder

### 33.38.1 Overview

Used to represent the order in which a plot passes the series in a dataset to the renderer. There are two values, as listed in table 33.9.

Class:	Description:
<code>SeriesRenderingOrder.FORWARD</code>	Forward.
<code>SeriesRenderingOrder.REVERSE</code>	Reverse.

Table 33.9: Series rendering order values

### 33.38.2 Notes

See the `setSeriesRenderingOrder()` method in the `XYPlot` class.

## 33.39 SpiderWebPlot

### 33.39.1 Overview

A plot that displays data from a `CategoryDataset` in a format that resembles a spider web—see figure 33.19 for an example.

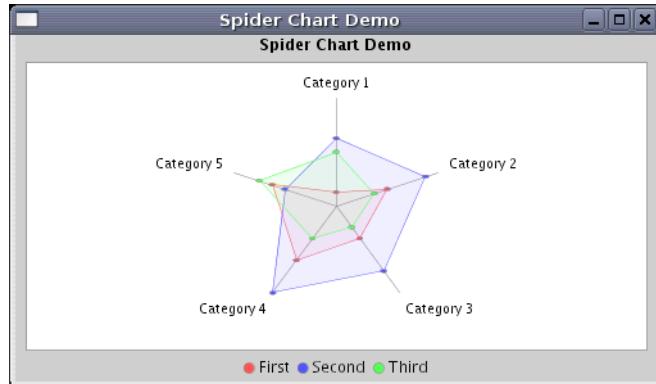


Figure 33.19: A spider web chart (see `SpiderWebChartDemo1.java`)

### 33.39.2 Constructors

There are three constructors for this class:

```
↳ public SpiderWebPlot();
```

Creates a new plot with a `null` dataset. All attributes are initialised with default values.

```
↳ public SpiderWebPlot(CategoryDataset dataset);
```

Creates a new plot with the given `dataset` (`null` is permitted), with each row in the dataset representing a series.

```
↳ public SpiderWebPlot(CategoryDataset dataset, TableOrder extract);
```

Creates a new plot with the given `dataset`. The `extract` argument controls whether rows or columns in the dataset are represented as “series” for the plot.

### 33.39.3 Methods

To get a brief description of the plot type:

```
↳ public String getPlotType();
```

Returns a short, localised, string describing the plot type (“Spider Web Plot” in English).

To access the plot’s dataset:

```
↳ public CategoryDataset getDataset();
```

Returns the plot’s dataset, which may be `null`.

```
↳ public void setDataset(CategoryDataset dataset);
```

Sets the dataset for the plot and sends a `PlotChangeEvent` to all registered listeners. The dataset can be set to `null`.

To control the order in which data items are read from the dataset:

```
↳ public TableOrder getDataExtractOrder();
```

Returns the “order” in which data items are extracted from the dataset. For `TableOrder.BY_ROW` (the default), each row is considered to be a series. For `TableOrder.BY_COLUMN`, each column in the dataset is considered to be a series.

```
↳ public void setDataExtractOrder(TableOrder order);
```

Sets the “order” in which data items are extracted from the dataset, and sends a `PlotChangeEvent` to all registered listeners. If `order` is `null`, this method throws an `IllegalArgumentException`.

To specify the starting position for the first category:

```
↳ public double getStartAngle();
```

Returns the angle of the first category, in degrees, relative to a radial line extending from the center of the plot horizontally to the right (that is, 3 o’clock on a clock face), in an anti-clockwise direction. The default value is `90.0` which draws the first category at the top of the plot (that is, 12 o’clock).

```
↳ public void setStartAngle(double angle);
```

Sets the angle for the first category, in degrees, and sends a `PlotChangeEvent` to all registered listeners.

To specify the direction in which categories are added to the plot:

```
↳ public Rotation getDirection();
```

Returns the direction in which the categories are added to the plot. The default is `Rotation.CLOCKWISE`.

```
↳ public void setDirection(Rotation direction);
```

Sets the direction in which the categories are added to the plot and sends a `PlotChangeEvent` to all registered listeners.

To control the size of the shapes drawn for each data item:

```
↳ public double getHeadPercent();
```

Returns the size of the shapes drawn at each data point. This is a percentage of width and height of the plot area. The default value is `0.01` (one percent).

```
↳ public void setHeadPercent(double percent);
```

Sets the size of the shapes drawn at each data point and sends a `PlotChangeEvent` to all registered listeners. The size is a percentage of the plot area height and width.

To control whether the “web” for each data series is filled or unfilled:

- ▶ `public boolean isWebFilled();`  
Returns the flag that controls whether the interior of the polygon defined by the data points for one series is filled. The default value is `true`.
- ▶ `public void setWebFilled(boolean flag);`  
Sets the flag that controls whether the interior of the polygon defined by the data points for each series is filled, and sends a `PlotChangeEvent` to all registered listeners.
- ▶ `public double getMaxValue();`  
Returns the maximum value for display on the axes.
- ▶ `public void setMaxValue(double value);`  
Sets the maximum value for display on the axes.
- ▶ `public double getInteriorGap();`  
Returns a percentage between 0.0 and 0.40 (forty percent) indicating the amount of whitespace to leave around the plot (some of which is used for the labels). The default value is 0.25.
- ▶ `public void setInteriorGap(double percent);`  
Sets the amount of whitespace around the plot as a percentage (in the range 0.0 to 0.40).

To appearance and position of the category labels on the chart can be controlled via the following methods:

- ▶ `public Font getLabelFont();`  
Returns the font used to display category labels (never `null`). The default is `Font("SansSerif", Font.PLAIN, 10)`.
- ▶ `public void setLabelFont(Font font);`  
Sets the font used to display category labels and sends a `PlotChangeEvent` to all registered listeners. An exception is thrown if `font` is `null`.
- ▶ `public Paint getLabelPaint();`  
Returns the paint used to display category labels (never `null`). The default is `Color.black`.
- ▶ `public void setLabelPaint(Paint paint);`  
Sets the paint used to display category labels and sends a `PlotChangeEvent` to all registered listeners. An exception is thrown if `paint` is `null`.
- ▶ `public double getAxisLabelGap();`  
Returns the gap between the end of each “radial” axis and the corresponding label, expressed as a percentage of the axis length. The default value is 0.10 (ten percent).
- ▶ `public void setAxisLabelGap(double gap);`  
Sets the gap between the end of each “radial” axis and the corresponding label, expressed as a percentage of the axis length, and sends a `PlotChangeEvent` to all registered listeners.

### 33.39.4 Series Attributes

#### Paint

The paint used to draw each series is specified on a “per series” basis:

- ▶ `public Paint getSeriesPaint(int series);`  
Returns the paint for the specified series. If `getSeriesPaint()` returns a non-null value, this is returned. Otherwise, the method checks to see if a specific value has been set for the series, in which case that is returned. If all else fails, the value returned by `getBaseSeriesPaint()` is used.
- ▶ `public void setSeriesPaint(int series, Paint paint);`  
Sets the paint for the specified series and sends a `PlotChangeEvent` to all registered listeners. It is permitted to set this to `null`.

The base paint is used as the fallback for any series that doesn’t have a paint explicitly defined:

- ▶ `public Paint getBaseSeriesPaint();`  
Returns the default series paint (never `null`).

► public void setBaseSeriesPaint(Paint paint);  
 Sets the default series paint and sends a [PlotChangeEvent](#) to all registered listeners.

As a convenience, you can override the paint for ALL series, although in typical usage you won't need to do this:<sup>6</sup>

► public Paint getSeriesPaint();  
 Returns the override paint for all series. The default value is `null`.  
 ► public void setSeriesPaint(Paint paint);  
 Sets the override paint for all series and sends a [PlotChangeEvent](#) to all registered listeners.

### OutlinePaint

The outline paint used to for each series (to draw the outline of the shape at each data point) is specified on a "per series" basis:

► public Paint getSeriesOutlinePaint(int series);  
 Returns the outline paint for the specified series. If `getSeriesOutlinePaint()` returns a non-`null` value, this is returned. Otherwise, the method checks to see if a specific value has been set for the series, in which case that is returned. If all else fails, the value returned by `getBaseSeriesOutlinePaint()` is used.  
 ► public void setSeriesOutlinePaint(int series, Paint paint);  
 Sets the outline paint for the specified series and sends a [PlotChangeEvent](#) to all registered listeners. It is permitted to set this to `null`.

The base paint is used as the fallback for any series that doesn't have a paint explicitly defined:

► public Paint getBaseSeriesOutlinePaint();  
 Returns the default series outline paint (never `null`).  
 ► public void setBaseSeriesOutlinePaint(Paint paint);  
 Sets the default series outline paint and sends a [PlotChangeEvent](#) to all registered listeners.

As a convenience, you can override the outline paint for ALL series, although in typical usage you won't need to do this:<sup>7</sup>

► public Paint getSeriesOutlinePaint();  
 Returns the override outline paint for all series. The default value is `null`.  
 ► public void setSeriesOutlinePaint(Paint paint);  
 Sets the outline paint for the specified series and sends a [PlotChangeEvent](#) to all registered listeners. It is permitted to set this to `null`.

### OutlineStroke

The outline stroke used to for each series (to draw the outline of the shape at each data point) is specified on a "per series" basis:

► public Stroke getSeriesOutlineStroke(int series);  
 Returns the outline stroke for the specified series. If `getSeriesOutlineStroke()` returns a non-`null` value, this is returned. Otherwise, the method checks to see if a specific value has been set for the series, in which case that is returned. If all else fails, the value returned by `getBaseSeriesOutlineStroke()` is used.  
 ► public void setSeriesOutlineStroke(int series, Stroke stroke);  
 Sets the outline stroke for the specified series and sends a [PlotChangeEvent](#) to all registered listeners. It is permitted to set this to `null`.

The base stroke is used as the fallback for any series that doesn't have a stroke explicitly defined:

► public Stroke getBaseSeriesOutlineStroke();  
 Returns the default series outline stroke (never `null`).

---

<sup>6</sup>These methods could be (much) better named.

<sup>7</sup>These methods could be (much) better named.

```
→ public void setBaseSeriesOutlineStroke(Stroke stroke);  
Sets the default series outline stroke and sends a PlotChangeEvent to all registered listeners.
```

As a convenience, you can override the outline stroke for ALL series, although in typical usage you won't need to do this:<sup>8</sup>

```
→ public Stroke getSeriesOutlineStroke();  
Returns the override outline stroke for all series. The default value is null.  
  
→ public void setSeriesOutlineStroke(Stroke stroke);  
Sets the outline stroke for the specified series and sends a PlotChangeEvent to all registered listeners. It is permitted to set this to null.
```

### 33.39.5 Legend Methods

A range of methods control the appearance of the legend for the plot (if the chart displays a legend):

```
→ public Shape getLegendItemShape();  
Returns the shape used for each legend item. The default value is Ellipse2D.Double(-4.0, -4.0, 8.0, 8.0).  
  
→ public void setLegendItemShape(Shape shape);  
Sets the shape to use for the legend items (null not permitted) and sends a PlotChangeEvent to all registered listeners. For correct alignment, the supplied shape should be centered on (0, 0).  
  
→ public CategoryItemLabelGenerator getLabelGenerator();  
Returns the generator that creates the labels for each category in the chart.  
  
→ public void setLabelGenerator(CategoryItemLabelGenerator generator);  
Sets the generator used to create labels for each category in the chart and sends a PlotChangeEvent to all registered listeners.  
  
→ public LegendItemCollection getLegendItems();  
Returns a collection of legend items for the plot. By default, this method returns one item for each category—you can override the method to change this behaviour.
```

### 33.39.6 Tool Tips and URLs

From version 1.0.2 onwards, this plot has support for generating tool tips (for display by the `ChartPanel` class, or in HTML image maps) and URLs (for HTML image maps):

```
→ public CategoryToolTipGenerator getToolTipGenerator(); [1.0.2]  
Returns the tool tip generator (possibly null). The default value is null.  
  
→ public void setToolTipGenerator(CategoryToolTipGenerator generator); [1.0.2]  
Sets the tool tip generator (null is permitted) and sends a PlotChangeEvent to all registered listeners.  
  
→ public CategoryURLGenerator getURLGenerator(); [1.0.2]  
Returns the URL generator (possibly null). The default value is null.  
  
→ public void setURLGenerator(CategoryURLGenerator generator); [1.0.2]  
Sets the URL generator (null is permitted) and sends a PlotChangeEvent to all registered listeners.
```

### 33.39.7 Other Methods

The `draw()` method is typically called by the `JFreeChart` class:

```
→ public void draw(Graphics2D g2, Rectangle2D area, Point2D anchor, PlotState parentState,  
PlotRenderingInfo info);  
Draws the plot within the specified area.
```

---

<sup>8</sup>These methods could be (much) better named.

### 33.39.8 Notes

Some points to note:

- this plot doesn't use a separate renderer mechanism, although that would be a useful enhancement;
- the plot does support negative values in the dataset, and the axis does not display any scale (both of these issues need to be addressed);
- a demo (`SpiderWebChartDemo1.java`) is included in the JFreeChart demo collection.

## 33.40 ThermometerPlot

### 33.40.1 Overview

A plot that displays a single value in a thermometer-style representation—see figure 33.20.

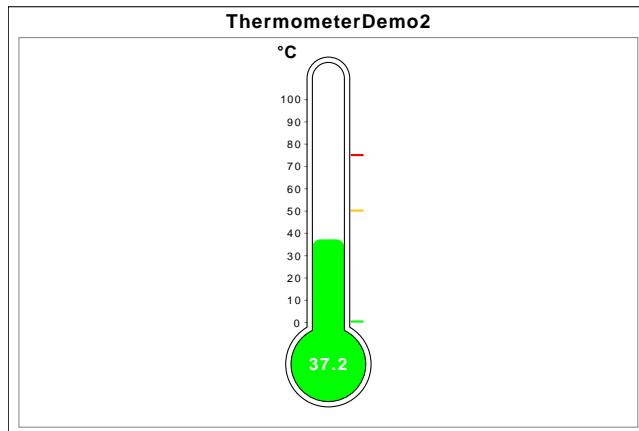


Figure 33.20: A simple thermometer chart (see `ThermometerDemo2.java`)

You can define three sub-ranges on the thermometer scale to provide some context for the displayed value: the *normal*, *warning* and *critical* sub-ranges. The colour of the “mercury” in the thermometer can be configured to change for each sub-range.

By default, the display range for the thermometer is fixed (using the overall range specified by the user). However, there is an option to automatically adjust the thermometer scale to display only the sub-range in which the current value falls. This allows the current data value to be displayed with more precision.

### 33.40.2 Constructors

To create a new `ThermometerPlot`:

► `public ThermometerPlot();`

Equivalent to `ThermometerPlot(new DefaultValueDataset())`—see the next method.

► `public ThermometerPlot(ValueDataset dataset);`

Creates a thermometer with default settings, using the supplied dataset (which may be `null`).

### 33.40.3 The Dataset

The ThermometerPlot displays a single value which it obtains from a `ValueDataset`. The following methods provide access to the dataset currently assigned to the plot:

- ▶ `public ValueDataset getDataset();`  
Returns the current dataset (possibly `null`).
- ▶ `public void setDataset(ValueDataset dataset);`  
Sets the dataset for the plot (replacing any existing dataset). It is permitted to set the dataset to `null`. Calling this method triggers a `PlotChangeEvent`.

When you assign a dataset to the plot, the plot registers itself as a `DatasetChangeListener`. Updating the dataset will result in the plot being notified, and the plot (in turn) notifies the chart. If you display your chart in a `ChartPanel`, this notification mechanism results in the chart being repainted whenever the dataset changes.

### 33.40.4 General Attributes

You can control the amount of white space around the thermometer:

- ▶ `public RectangleInsets getPadding();`  
Returns the padding around the outside of the thermometer (never `null`). The default value is `new RectangleInsets(UnitType.RELATIVE, 0.05, 0.05, 0.05, 0.05)`.
- ▶ `public void setPadding(RectangleInsets padding);`  
Sets the padding around the thermometer and sends a `PlotChangeEvent` to all registered listeners. If `padding` is `null`, this method throws an `IllegalArgumentException`.

To control the size of the thermometer bulb:

- ▶ `public int getBulbRadius(); [1.0.7]`  
Returns the radius (or half-width) of the thermometer, in Java2D units. The default value is 40.
- ▶ `public void setBulbRadius(int r); [1.0.7]`  
Sets the radius of the thermometer bulb, and sends a `PlotChangeEvent` to all registered listeners. No validation is performed on the argument—you should generally specify a value that is larger than the column radius.
- ▶ `public int getBulbDiameter(); [1.0.7]`  
Returns the bulb diameter, which is always twice the value returned by `getBulbRadius()`. There is no setter method for this attribute.

To control the width of the main column of the thermometer:

- ▶ `public int getColumnRadius(); [1.0.7]`  
Returns the radius (or half-width) of the main column of the thermometer, in Java2D units. The default value is 20.
- ▶ `public void setColumnRadius(int r); [1.0.7]`  
Sets the radius of the main column of the thermometer, in Java2D units, and sends a `PlotChangeEvent` to all registered listeners. No validation is performed on the new value—in general, you should specify a value that is less than the bulb radius.
- ▶ `public int getColumnDiameter(); [1.0.7]`  
Returns the column diameter, which is always twice the value returned by `getColumnRadius()`. There is no setter method for this attribute.

To control the gap between the two outlines drawn to represent the thermometer:

- ▶ `public int getGap(); [1.0.7]`  
Returns the gap, in Java2D units, between the thermometer outlines.
- ▶ `public void setGap(int gap); [1.0.7]`  
Sets the gap, in Java2D units, between the outlines used to represent the thermometer, and sends a `PlotChangeEvent` to all registered listeners.

To control the colour of the thermometer outline:

```
→ public Paint getThermometerPaint();
```

Returns the outline paint (never `null`). The default value is `Color.black`.

```
→ public void setThermometerPaint(Paint paint);
```

Sets the paint used to draw the outline of the thermometer and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, this method does nothing.

To control the pen used to draw the thermometer outline:

```
→ public Stroke getThermometerStroke();
```

Returns the outline stroke (never `null`). The default value is `new BasicStroke(1.0f)`.

```
→ public void setThermometerStroke(Stroke stroke);
```

Sets the stroke used to draw the outline of the thermometer and sends a `PlotChangeEvent` to all registered listeners. If `stroke` is `null`, this method does nothing.

### 33.40.5 The Value Label

The current data value can be displayed on the plot, with settings to control the location, font, colour, and number formatter.

First, to control the location:

```
→ public int getValueLocation();
```

Returns the current value location, which is one of:

- **NONE** (0) – no value label is displayed;
- **RIGHT** (1) – the value label is displayed to the right of the thermometer;
- **LEFT** (2) – the value label is displayed to the left of the thermometer;
- **BULB** (3) – the value label is displayed inside the thermometer bulb.

The default value is **BULB**.

```
→ public void setValueLocation(int location);
```

Sets the location at which the current value will be displayed, and sends a `PlotChangeEvent` to all registered listeners. If `location` is not one of the values specified in `getValueLocation()`, this method throws an `IllegalArgumentException`. Note that the default value label colour is white, so if you change the location you should also change the colour so that the label remains visible.

The font for the value label can be set as follows:

```
→ public Font getValueFont();
```

Returns the font used to display the value label, if it is visible. The default value is `new Font("SansSerif", Font.BOLD, 16)`.

```
→ public void setValueFont(Font font);
```

Sets the font used to display the value label, if it is visible, and sends a `PlotChangeEvent` to all registered listeners. If `font` is `null`, this method throws an `IllegalArgumentException`.

Similarly, the paint for the value label can be set as follows:

```
→ public Paint getValuePaint();
```

Returns the paint used to display the value label, if it is visible. The default value is `Color.white`.

```
→ public void setValuePaint(Paint paint);
```

Sets the paint used to display the current value and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

You can set a formatter for the value label:<sup>9</sup>

```
→ public void setValueFormat(NumberFormat formatter);
```

Sets the formatter for the value label and sends a `PlotChangeEvent` to all registered listeners.

---

<sup>9</sup>For whatever reason, there is currently no corresponding “getter” method.

Near the top of the thermometer, a label can be shown with a temperature unit:

► `public int getUnits();`

Returns a code indicating the unit type:

- `UNITS_NONE` – no unit indicator is displayed;
- `UNITS_FAHRENHEIT` – Fahrenheit units;
- `UNITS_CELCIUS` – Celcius units;
- `UNITS_KELVIN` – Kelvins.

The default value is `UNITS_CELCIUS`.

► `public void setUnits(int u);`

Sets the unit type and sends a `PlotChangeEvent` to all registered listeners. If `u` is not one of the constants specified in `getUnits()`, this method does nothing.

► `public void setUnits(String u); [DEPRECATED]`

Sets the unit type via a string. This method is deprecated—use `setUnits(int)` instead.

### 33.40.6 The Range Axis

The `ThermometerPlot` has a single range axis that controls the display of data values. You can access the range axis with the following methods:

► `public ValueAxis getRangeAxis();`

Returns the axis used by the plot. This is never `null`. The default value is a default `NumberAxis` with no label.

► `public void setRangeAxis(ValueAxis axis);`

Sets the axis for the plot and sends a `PlotChangeEvent` to all registered listeners. If `axis` is `null`, this method throws an `IllegalArgumentException`.

You can change any of the visual characteristics of the axis, but you should avoid changing the axis range directly—instead, use the plot bounds methods specified in the next sub-section.

The axis can be displayed at the left or right of the thermometer, or not displayed at all:

► `public int getAxisLocation();`

Returns the axis location, which is one of:

- `NONE`;
- `LEFT`;
- `RIGHT`.

The default value is `LEFT`.

► `public void setAxisLocation(int location);`

Sets the axis location and sends a `PlotChangeEvent` to all registered listeners. If `location` is not one of the values specified in `getAxisLocation`, this method throws an `IllegalArgumentException`.

The following methods don't appear to do anything useful, and have been deprecated:

► `public boolean getShowValueLines(); [DEPRECATED]`

Returns the `showValueLines` flag.

► `public void setShowValueLines(boolean flag); [DEPRECATED]`

Sets a flag that doesn't appear to do anything at all! This method has been deprecated.

### 33.40.7 The Plot Bounds

The plot stores lower and upper bounds for the values it can display, controlled by the methods below. Note that the axis may display some subset of the specified bounds (see the subranges in the next section):

- `public double getLowerBound();`  
Returns the lower bound for the plot. The default value is 0.0.
- `public void setLowerBound(double lower);`  
Sets the lower bound for the plot and updates the current axis range.
- `public double getUpperBound();`  
Returns the upper bound for the plot. The default value is 100.0.
- `public void setUpperBound(double upper);`  
Sets the upper bound for the plot and updates the current axis range.

To set the overall range of values to be displayed in the thermometer:

- `public void setRange(double lower, double upper);`  
Sets the lower and upper bounds for the value that can be displayed in the thermometer. If the data value is outside this range, the thermometer will be drawn as “empty” or “full”.

### 33.40.8 Subranges

Within the plot’s bounds, you can specify three sub-ranges:

- **NORMAL** (0) – the normal range;
- **WARNING** (1) – the warning range;
- **CRITICAL** (2) – the critical range.

Each subrange can have a different mercury paint.

To define subranges:

- `public void setSubrange(int subrange, double lower, double upper);`  
Sets the lower and upper bounds for a sub-range. Use one of the constants **NORMAL**, **WARNING** or **CRITICAL** to indicate the sub-range.

In addition to the actual bounds for the sub-ranges, you can specify *display bounds* for each subrange:

- `public void setDisplayBounds(int range, double lower, double upper);`  
Sets the lower and upper bounds of the display range for a sub-range. The display range is usually equal to or slightly bigger than the actual bounds of the sub-range.

The display bounds are only used if the thermometer axis range is automatically adjusted to display the current sub-range (see the `followDataInSubranges` flag in the next section).

- `public boolean getFollowDataInSubranges();`  
Returns the flag that controls whether or not the axis range automatically changes to show only the subrange within which the current data value falls. The default value is `false`.
- `public void setFollowDataInSubranges(boolean flag);`  
If `true`, the thermometer range is adjusted to display only the current sub-range (which displays the value with greater precision). If `false`, the overall range is displayed at all times.

By default, this flag is set to `false`.

A couple of additional setter methods allow you to specify the subrange value and display bounds in one call:

- `public void setSubrangeInfo(int range, double low, double hi);`  
Sets the value and display bounds for the specified subrange.
- `public void setSubrangeInfo(int range, double rangeLow, double rangeHigh, double displayLow, double displayHigh);`  
Sets the value and display bounds for the specified subrange.

### 33.40.9 The Mercury Colour

In a real thermometer, mercury is the substance inside the thermometer that expands and contracts, rising and falling within the glass tube and indicating the current temperature. In the `Thermometer` plot, a tall thin rectangle is drawn in the middle of the plot to highlight the current value—the colour of this indicator is the “mercury colour”.

The default mercury colour is defined with the following methods:

► `public Paint getMercuryPaint();`

Returns the default mercury paint (never `null`). This is used:

- for all values, if `useSubrangePaint` is `false`;
- for any value outside the three defined subranges, if `useSubrangePaint` is `true`.

The default value is `Color.lightGray`.

► `public void setMercuryPaint(Paint paint);`

Sets the default mercury paint and sends a `PlotChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

A different colour can be specified for each of three user-definable subranges:

► `public Paint getSubrangePaint(int range);`

Returns the paint (never `null`) defined for the specified subrange. The default values are `Color.green`, `Color.orange` and `Color.red`. If `range` is not in the range 0 to 2, this method returns the default mercury paint (as returned by `getMercuryPaint()`).

► `public void setSubrangePaint(int range, Paint paint);`

Sets the paint for the specified subrange and sends a `PlotChangeEvent` to all registered listeners. If `range` is not in the range 0 to 2, or `paint` is `null`, this method does nothing.<sup>10</sup>

The mercury colours for subranges are only used if the `useSubrangePaint` flag is set to `true` (which is the default):

► `public boolean getUseSubrangePaint();`

Returns the flag that controls whether or not the subrange colours are used. The default value is `true`.

► `public void setUseSubrangePaint(boolean flag);`

Sets the flag that controls whether or not the sub-range colours are used for the mercury in the thermometer, and sends a `PlotChangeEvent` to all registered listeners.

### 33.40.10 Other Methods

The following methods are called by the `JFreeChart` class during chart drawing—you shouldn’t need to call them directly:

► `public Range getDataRange(ValueAxis axis);`

Returns the default range for the specified axis. This is required by the `Zoomable` interface.

► `public LegendItemCollection getLegendItems();`

Always returns `null`. This plot displays only a single value, so it doesn’t need a default legend.

► `public PlotOrientation getOrientation();`

Always returns `PlotOrientation.VERTICAL`.

► `public void draw(Graphics2D g2, Rectangle2D plotArea,`

`Point2D anchor, PlotState parentState, PlotRenderingInfo state);`

Draws the plot within the specified area.

In typical situations, you won’t normally call this method directly.

---

<sup>10</sup>This is at odds with the general JFreeChart style where an exception would be thrown.

### 33.40.11 Zooming

This plot has in-built support zooming (a mechanism that is utilised, for example, by the `ChartPanel` class):

```
➔ public boolean isDomainZoomable();  
Returns false because this plot type has no domain axis.
```

The domain axis zooming methods are all no-ops:

```
➔ public void zoomDomainAxes(double factor, PlotRenderingInfo state, Point2D source);  
Does nothing—this plot has no domain axis.  
  
➔ public void zoomDomainAxes(double factor, PlotRenderingInfo state, Point2D source, boolean  
useAnchor); [1.0.7]  
As above.  
  
➔ public void zoomDomainAxes(double lowerPercent, double upperPercent, PlotRenderingInfo state,  
Point2D source);  
As above.
```

The range axis on the plot is zoomable:

```
➔ public boolean isRangeZoomable();  
Returns true to indicate that the range axis is zoomable for this plot type.
```

The following zooming methods are defined:

```
➔ public void zoomRangeAxes(double factor, PlotRenderingInfo state, Point2D source);  
Equivalent to zoomRangeAxis(factor, state, source, false)—see next method.  
  
➔ public void zoomRangeAxes(double factor, PlotRenderingInfo state, Point2D source,  
boolean useAnchor); [1.0.7]  
Scales the length of the range axis by the specified factor. If useAnchor is true, the axis bounds  
are scaled about the source point, otherwise the scaling occurs around the central point of the  
current axis range.  
  
➔ public void zoomRangeAxes(double lowerPercent, double upperPercent,  
PlotRenderingInfo state, Point2D source);  
Updates the lower and upper bounds of the axis range to the specified points (expressed as a  
point along the axis in percentage terms).
```

### 33.40.12 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);  
Tests this plot for equality with an arbitrary object. Note that the dataset is ignored for the  
equality test. If obj is null, this method returns false.
```

Instances of this class are `Cloneable` and `Serializable`.

### 33.40.13 Notes

Some points to note:

- the `ThermometerPlot` class was originally contributed by Bryan Scott from the Australian Antarctic Division.
- the `JThermometer` class provides a simple (but incomplete) Javabean wrapper for this class.
- a demo (`ThermometerDemo1.java`) is included in the JFreeChart Demo Collection.

#### See Also

[ValueDataset](#).

## 33.41 ValueAxisPlot

### 33.41.1 Overview

This interface should be implemented by plots that use the `ValueAxis` class (or its subclasses). This provides an entry point for the axis to query the plot to find out the range of data values that will be plotted against the axis (bear in mind that more than one dataset from the plot may be mapped to a particular axis).

### 33.41.2 Methods

This interface defines a single method:

```
➔ public Range getDataRange(ValueAxis axis);
```

Returns the range that is required to display all data values that are plotted against the specified axis.

### 33.41.3 Notes

This interface is implemented by `CategoryPlot`, `FastScatterPlot`, `PolarPlot`, `ThermometerPlot` and `XYPlot`.

## 33.42 ValueMarker

### 33.42.1 Overview

A *value marker* is used to indicate a constant value against the domain or range axis for a `CategoryPlot` or an `XYPlot`. This class extends the `Marker` class.

### 33.42.2 Usage

You can add a marker to an `XYPlot` using the `addDomainMarker()` or `addRangeMarker()` methods. Similarly, you can add a range marker to a `CategoryPlot` using the `addRangeMarker()` method.

There is a demo application (`MarkerDemo1.java`) included in the JFreeChart demo collection that illustrates the use of this class.

### 33.42.3 Constructors

The following constructors are defined:

```
➔ public ValueMarker(double value);
Creates a new instance with the specified value and default values for all other attributes.

➔ public ValueMarker(double value, Paint paint, Stroke stroke);
Creates a new instance with the specified value, paint and stroke.

➔ public ValueMarker(double value, Paint paint, Stroke stroke, Paint outlinePaint, Stroke
outlineStroke, float alpha);
Creates a new instance with the specified value and attributes.
```

### 33.42.4 Methods

In addition to the methods inherited from `Marker`, this class defines:

```
➔ public double getValue();
Returns the current value for the marker.

➔ public void setValue(double value); [1.0.3]
Sets the value for the marker and sends a MarkerChangeEvent to all registered listeners.
```

### 33.42.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
```

Tests this instance for equality with an arbitrary object (which may be `null`).

Instances of this class are cloneable and serializable.

### 33.42.6 Notes

Some points to note:

- the marker is most often drawn as a line, but in a chart with a 3D-effect the marker will be drawn as a polygon—for this reason, the marker has both `paint` and `outlinePaint` attributes, and `stroke` and `outlineStroke` attributes;

#### See Also

[IntervalMarker](#).

## 33.43 WaferMapPlot

### 33.43.1 Overview

A plot for generating wafer map charts.

### 33.43.2 Draw Method

The following method is called by the `JFreeChart` class during chart drawing:

```
➔ public void draw(Graphics2D g2, Rectangle2D plotArea,
    Point2D anchor, PlotState parentState, PlotRenderingInfo state);
    Draws the plot within the specified area.
```

In typical situations, you won't normally call this method directly.

#### See Also

[WaferMapRenderer](#).

## 33.44 XYPlot

### 33.44.1 Overview

Draws a visual representation of data from an `XYDataset`, where the domain axis measures the x-values and the range axis measures the y-values.

The type of plot is typically displayed using a vertical orientation, but it is possible to change to a horizontal orientation which can be useful for certain applications.

### 33.44.2 Layout

Axes are laid out at the left and bottom of the drawing area. The space allocated for the axes is determined automatically. The following diagram shows how this area is divided:

Determining the dimensions of these regions is an awkward problem. The plot area can be resized arbitrarily, but the vertical axis and horizontal axis sizes are more difficult. Note that the height of the vertical axis is related to the height of the horizontal axis, and, likewise, the width of the vertical axis is related to the width of the horizontal axis. This results in a “chicken and egg” problem, because changing the width of an axis can affect its height (especially if the tick units change with the resize) and changing its height can affect the width (for the same reason).



*Figure 33.21: The plot regions*

### 33.44.3 Constructors

To create a default instance:

```
➔ public XYPlot();
```

Creates a new plot with no dataset, no axes and no renderer. Take care to specify these attributes before using the plot, otherwise JFreeChart may throw a `NullPointerException`.

To create a plot with a specific renderer:

```
➔ public XYPlot(XYDataset dataset, ValueAxis domainAxis, ValueAxis rangeAxis,
XYItemRenderer renderer);
```

Creates a new `XYPlot` instance with the specified dataset, axes and renderer. Any of these arguments can be `null`, but in that case you should take care to specify the missing attributes before using the plot, otherwise JFreeChart may throw a `NullPointerException`.

### 33.44.4 Datasets and Renderers

An `XYPlot` can have zero, one or many datasets and each dataset is usually associated with a renderer (the object that is responsible for drawing the visual representation of each item in a dataset). A dataset is an instance of any class that implements the `XYDataset` interface and a renderer is an instance of any class that implements the `XYItemRenderer` interface.

To get/set a dataset:

```
➔ public XYDataset getDataset();
```

Equivalent to `getDataset(0)`—see below. This is a convenience method for developers that typically work with a single dataset.

```
➔ public XYDataset getDataset(int index);
```

Returns the dataset at the specified index (possibly `null`).

```
➔ public void setDataset(XYDataset dataset);
```

Equivalent to `setDataset(0, dataset)`—see below. This is a convenience method for developers that typically work with a single dataset.

```
➔ public void setDataset(int index, XYDataset dataset);
```

Assigns a dataset to the plot at the given index. The new dataset replaces any existing dataset at the specified index. It is permitted to set a dataset to `null` (in that case, no data will be displayed on the chart). For non-`null` datasets, this plot will be registered with the dataset to receive notification of changes to the dataset.

To get/set a renderer:

```
➔ public XYItemRenderer getRenderer(int index);
```

Returns the renderer at the specified index (possibly `null`).

```
➔ public void setRenderer(int index, XYItemRenderer renderer);
```

Sets the renderer at the specified index and sends a `PlotChangeEvent` to all registered listeners. It is permitted to set any renderer to `null`.

A number of renderer implementations are available (and you are free to develop your own, of course):

- `CandlestickRenderer`;
- `ClusteredXYBarRenderer`;
- `HighLowRenderer`;
- `StandardXYItemRenderer`;
- `XYAreaRenderer`;
- `XYBarRenderer`;
- `XYBubbleRenderer`;
- `XYDifferenceRenderer`;

The items in each dataset are, by default, plotted against the primary axes. You can change that by adding a mapping between a dataset and the axes it should be plotted against—see section [33.44.10](#).

### 33.44.5 Dataset Rendering Order

When a plot has multiple datasets and renderers, the order in which the datasets are rendered has an impact on the appearance of the chart. You can control the rendering order using the following methods:

- ➔ `public DatasetRenderingOrder getDatasetRenderingOrder()`;  
Returns the current dataset rendering order (never `null`).
- ➔ `public void setDatasetRenderingOrder(DatasetRenderingOrder order)`;  
Sets the dataset rendering order and sends a `PlotChangeEvent` to all registered listeners. It is not permitted to set the rendering order to `null`.

By default, datasets will be rendered in reverse order so that the “primary” dataset appears to be “on top” of the other datasets.

### 33.44.6 Series Rendering Order

The series within each dataset are, by default, rendered in reverse order (so that the first series in each dataset appears in front of all the other series in that dataset). You can control this using the following methods:

- ➔ `public SeriesRenderingOrder getSeriesRenderingOrder()`;  
Returns the current series rendering order (never `null`). The default value is `SeriesRenderingOrder.REVERSE`.
- ➔ `public void setSeriesRenderingOrder(SeriesRenderingOrder order)`;  
Sets the series rendering order and sends a `PlotChangeEvent` to all registered listeners. This method throws an `IllegalArgumentException` if `order` is `null`.

### 33.44.7 Axes

Most plots will have a single domain axis (or x-axis) and a single range axis (or y-axis). To get/set the domain axis:

- ➔ `public ValueAxis getDomainAxis()`;  
Returns the domain axis with index 0.
- ➔ `public void setDomainAxis(ValueAxis axis)`;  
Sets the domain axis with index 0 and sends a `PlotChangeEvent` to all registered listeners.

To get/set the range axis:

```
► public ValueAxis getRangeAxis();
Returns the range axis with index 0.
```

```
► public void setRangeAxis(ValueAxis axis);
Sets the range axis with index 0 and sends a PlotChangeEvent to all registered listeners.
```

Multiple domain and/or range axes are also supported—see Chapter 13 for details.

```
► public ValueAxis getDomainAxis(int index);
Returns the domain axis at the specified index. This method can return null.
```

```
► public void setDomainAxis(int index, ValueAxis axis);
Equivalent to setDomainAxis(index, axis, true)—see below.
```

```
► public void setDomainAxis(int index, ValueAxis axis, boolean notify);
Sets the domain axis at the specified index (replacing any axis already there) and, if requested, sends a PlotChangeEvent to all registered listeners.
```

```
► public void setDomainAxes(ValueAxis[] axes);
This is a convenience method that calls setDomainAxis(int, ValueAxis) for each axis in the array, firing a single PlotChangeEvent once all the axes have been added. An exception is thrown if axes is null.
```

```
► public ValueAxis getRangeAxis(int index);
Returns the range axis at the specified index. This method can return null.
```

```
► public void setRangeAxis(int index, ValueAxis axis);
Equivalent to setRangeAxis(index, axis, true)—see below.
```

```
► public void setRangeAxis(int index, ValueAxis axis, boolean notify);
Sets the range axis at the specified index (replacing any axis already there) and, if requested, sends a PlotChangeEvent to all registered listeners.
```

```
► public void setRangeAxes(ValueAxis[] axes);
This is a convenience method that calls setRangeAxis(int, ValueAxis) for each axis in the array, firing a single PlotChangeEvent once all the axes have been added. An exception is thrown if axes is null.
```

### 33.44.8 Location of Axes

The plot's axes can appear at the top, bottom, left or right of the plot area. The location for an axis is specified using the [AxisLocation](#) class, which combines two possible locations within each option—which one is actually used depends on the orientation (horizontal or vertical) of the plot.

For “vertical” plots (the usual default), the domain axis will appear at the top or bottom of the plot area, and the range axis will appear at the left or right of the plot area. For “horizontal” plots, the domain axis will appear at the left or right of the plot area, and the range axis will appear at the top or bottom of the plot area.

To set the location for the domain axis:

```
► public void setDomainAxisLocation(AxisLocation location);
Sets the location for the domain axis and sends a PlotChangeEvent to all registered listeners.
```

Similarly, to set the location for the range axis:

```
► public void setRangeAxisLocation(AxisLocation location);
Sets the range axis location and sends a PlotChangeEvent to all registered listeners.
```

For example, to display the range axis on the right side of a chart:

```
plot.setRangeAxisLocation(AxisLocation.BOTTOM\_OR\_RIGHT);
```

This assumes the plot orientation is vertical, if it changes to horizontal the axis will be displayed at the bottom of the chart.

### 33.44.9 Axis Offsets

By default, the axes are drawn “flush” against the edge of the plot’s data area. It is possible to specify an amount by which the plot’s axes are offset from the data area using the following methods:

► public `RectangleInsets` `getAxisOffset()`;

Returns the gap between the plot’s data area and the axes.

► public void `setAxisOffset(RectangleInsets offset)`;

Sets the gap between the plot’s data area and the axes. You cannot set this to `null`—for no gap, use `RectangleInsets.ZERO_INSETS`.

### 33.44.10 Mapping Datasets to Axes

For a plot with multiple datasets, renderers and axes, you need to specify which axes should be used for each dataset. By default, the items in a dataset will be plotted against the “primary” domain and range axes—that is, the axes at index 0.

If you want a dataset plotted against a different axis, you need to “map” the dataset to the axis. There are separate methods to map a dataset to a domain axis and a range axis:

► public void `mapDatasetToDomainAxis(int index, int axisIndex)`;

Maps a dataset to a domain axis. You need to take care that the dataset and axis both exist when you create a mapping entry.

► public void `mapDatasetToRangeAxis(int index, int axisIndex)`;

Maps a dataset to a range axis. You need to take care that the dataset and axis both exist when you create a mapping entry.

To find the domain and/or range axis that a dataset is currently mapped to:

► public `ValueAxis` `getDomainAxisForDataset(int index)`

Returns the domain axis that the specified dataset is currently mapped to.

► public `ValueAxis` `getRangeAxisForDataset(int index)`;

Returns the range axis that the specified dataset is currently mapped to.

### 33.44.11 Gridlines

The `XYPlot` class provides support for drawing gridlines against the primary domain axis and the primary range axis (gridlines for other axes are not currently supported).<sup>11</sup> For each axis, there is a flag that controls whether or not the gridlines are visible. For visible gridlines, you can customise the line style (`Stroke`) and color (`Paint`).

For example, to change the grid to display solid black lines:

```
XYPlot plot = (XYPlot) chart.getPlot();
plot.setDomainGridlineStroke(new BasicStroke(0.5f));
plot.setDomainGridlinePaint(Color.black);
plot.setRangeGridlineStroke(new BasicStroke(0.5f));
plot.setRangeGridlinePaint(Color.black);
```

If you prefer to have no gridlines at all, you can turn them off:

```
XYPlot plot = (XYPlot) chart.getPlot();
plot.setDomainGridlinesVisible(false);
plot.setRangeGridlinesVisible(false);
```

The settings for the domain axis gridlines and the range axis gridlines are independent of one another. The following methods control the domain axis gridlines:

► public boolean `isDomainGridlinesVisible()`;

Returns `true` if the plot should draw gridlines for the primary domain axis, and `false` otherwise.

---

<sup>11</sup>There is a good argument for moving this feature into the axis classes, but that hasn’t been done yet.

```
→ public void setDomainGridlinesVisible(boolean visible);
Sets the flag that controls whether or not the plot should draw gridlines for the primary domain axis, and sends a PlotChangeEvent to all registered listeners.
```

```
→ public Stroke getDomainGridlineStroke();
Returns the stroke used to draw the gridlines for the primary domain axis (the default is a thin dashed line). This method never returns null.
```

```
→ public void setDomainGridlineStroke(Stroke stroke);
Sets the stroke used to draw the gridlines for the primary domain axis, and sends a PlotChangeEvent to all registered listeners. This method throws an IllegalArgumentException if stroke is null.
```

```
→ public Paint getDomainGridlinePaint();
Returns the paint used to draw the gridlines for the primary domain axis (the default is Color.lightGray). This method never returns null.
```

```
→ public void setDomainGridlinePaint(Paint paint);
Sets the paint used to draw the gridlines for the primary domain axis, and sends a PlotChangeEvent to all registered listeners. This method throws an IllegalArgumentException if paint is null.
```

Similarly, the following methods control the range axis gridlines:

```
→ public boolean isRangeGridlinesVisible();
Returns true if the plot should draw gridlines for the primary range axis, and false otherwise.
```

```
→ public void setRangeGridlinesVisible(boolean visible);
Sets the flag that controls whether or not the plot should draw gridlines for the primary range axis, and sends a PlotChangeEvent to all registered listeners.
```

```
→ public Stroke getRangeGridlineStroke();
Returns the stroke used to draw the gridlines for the primary range axis (the default is a thin dashed line). This method never returns null.
```

```
→ public void setRangeGridlineStroke(Stroke stroke);
Sets the stroke used to draw the gridlines for the primary range axis, and sends a PlotChangeEvent to all registered listeners. This method throws an IllegalArgumentException if stroke is null.
```

```
→ public Paint getRangeGridlinePaint();
Returns the paint used to draw the gridlines for the primary range axis (the default is Color.lightGray). This method never returns null.
```

```
→ public void setRangeGridlinePaint(Paint paint);
Sets the paint used to draw the gridlines for the primary range axis, and sends a PlotChangeEvent to all registered listeners. This method throws an IllegalArgumentException if paint is null.
```

### 33.44.12 Grid Bands

The `XYPlot` class can color alternate bands between the tick marks on the plot's axes. To control this facility for the (primary) domain axis:

```
→ public Paint getDomainTickBandPaint();
Returns the paint used to fill alternate bands between the tick values on the primary domain axis. The default value is null (no bands are filled).
```

```
→ public void setDomainTickBandPaint(Paint paint);
Sets the paint used to fill alternate bands between the tick values on the domain axis, and sends a PlotChangeEvent to all registered listeners. If paint is null, no bands will be filled.
```

A similar feature is supported for the (primary) range axis:

```
→ public Paint getRangeTickBandPaint();
Returns the paint used to fill alternate bands between the tick values on the primary range axis. The default value is null (no bands are filled).
```

```
→ public void setRangeTickBandPaint(Paint paint);
Sets the paint used to fill alternate bands between the tick values on the range axis, and sends a PlotChangeEvent to all registered listeners. If paint is null, no bands will be filled.
```

A demo (`GridBandDemo1.java`) is included in the JFreeChart demo collection.

### 33.44.13 Zero Base Line

A facility is provided where the plot can draw a base line against the range axis at the zero value. This is not visible by default, but you can switch it on and control the stroke and paint using the following methods:

- `public boolean isRangeZeroBaselineVisible();`  
Returns the flag that controls whether or not the plot draws a base line against the zero value of the range axis. The default value is `false`.
- `public void setRangeZeroBaselineVisible(boolean visible);`  
Sets the flag that controls whether or not the plot draws a base line against the zero value of the range axis. A `PlotChangeEvent` is sent to all registered listeners.
- `public Stroke getRangeZeroBaselineStroke();`  
Returns the stroke (never `null`) used to draw the zero baseline, if it is visible. The default is `BasicStroke(0.5f)`.
- `public void setRangeZeroBaselineStroke(Stroke stroke);`  
Sets the stroke used to draw the zero baseline, and sends a `PlotChangeEvent` to all registered listeners. This method throws an `IllegalArgumentException` if `stroke` is `null`.
- `public Paint getRangeZeroBaselinePaint();`  
Returns the paint (never `null`) used to draw the zero baseline, if it is visible. The default is `Color.black`.
- `public void setRangeZeroBaselinePaint(Paint paint);`  
Sets the paint used to draw the zero baseline and sends a `PlotChangeEvent` to all registered listeners.

### 33.44.14 Crosshairs

The `XYPLOT` class supports the display of crosshairs for the primary domain and range axes.<sup>12</sup> Use the following methods:

- `public boolean isDomainCrosshairVisible();`  
Returns `true` if the crosshair for the primary domain axis is visible, and `false` otherwise.
- `public void setDomainCrosshairVisible(boolean flag);`  
Sets the visibility flag for the crosshair linked to the primary domain axis, and sends a `PlotChangeEvent` to all registered listeners.
- `public boolean isDomainCrosshairLockedOnData();`  
Returns `true` if the crosshair automatically “snaps” to the nearest data value, and `false` otherwise.
- `public void setDomainCrosshairLockedOnData(boolean flag);`  
Sets the flag that determines whether or not the crosshair automatically snaps to the nearest data value. If the flag value changes, this method sends a `PlotChangeEvent` to all registered listeners.
- `public double getDomainCrosshairValue();`  
Returns the value of the crosshair for the domain axis. Note that this value is recalculated as the chart is repainted, so you should only rely on this value if you know that the chart has finished painting. In particular, this method will return the old value if you call it from within a mouse click event handler. You can use a `ChartProgressListener` to determine when chart painting has completed—see `CrosshairDemo1.java` for an example.
- `public void setDomainCrosshairValue(double value);`  
Sets the crosshair value for the primary domain axis and sends a `PlotChangeEvent` to all registered listeners.
- `public void setDomainCrosshairValue(double value, boolean notify);`  
Sets the crosshair value for the primary domain axis and if requested sends a `PlotChangeEvent` to all registered listeners (but only if the crosshairs are visible).

---

<sup>12</sup>Unfortunately, it is not possible to display crosshairs for other axes at this time.

```
► public Stroke getDomainCrosshairStroke();
Returns the domain crosshair stroke (never null). The default value is a thin dashed line.

► public void setDomainCrosshairStroke(Stroke stroke);
Sets the stroke used to display the crosshair for the primary domain axis (null is not permitted), and sends a PlotChangeEvent to all registered listeners.

► public Paint getDomainCrosshairPaint();
Returns the paint (never null) used to draw the crosshair for the primary domain axis. The default value is Color.blue.

► public void setDomainCrosshairPaint(Paint paint);
Sets the paint used to draw the crosshair for the primary domain axis and sends a PlotChangeEvent to all registered listeners.
```

Similarly for the crosshair related to the primary range axis:

```
► public boolean isRangeCrosshairVisible();
Returns true if the crosshair for the primary range axis is visible, and false otherwise.

► public void setRangeCrosshairVisible(boolean flag);
Sets the visibility flag for the crosshair linked to the primary range axis, and sends a PlotChangeEvent to all registered listeners.

► public boolean isRangeCrosshairLockedOnData();
Returns true if the crosshair automatically “snaps” to the nearest data value, and false otherwise.

► public void setRangeCrosshairLockedOnData(boolean flag);
Sets the flag that determines whether or not the crosshair automatically snaps to the nearest data value. If the flag value changes, this method sends a PlotChangeEvent to all registered listeners.

► public double getRangeCrosshairValue();
Returns the value of the crosshair for the range axis. Note that this value is recalculated as the chart is repainted, so you should only rely on this value if you know that the chart has finished painting. In particular, this method will return the old value if you call it from within a mouse click event handler. You can use a ChartProgressListener to determine when chart painting has completed—see CrosshairDemo1.java for an example.

► public void setRangeCrosshairValue(double value);
Sets the crosshair value for the primary range axis and sends a PlotChangeEvent to all registered listeners.

► public void setRangeCrosshairValue(double value, boolean notify);
Sets the crosshair value for the primary range axis and if requested sends a PlotChangeEvent to all registered listeners (but only if the crosshairs are visible).

► public Stroke getRangeCrosshairStroke();
Returns the range crosshair stroke (never null). The default value is a thin dashed line.

► public void setRangeCrosshairStroke(Stroke stroke);
Sets the stroke used to display the crosshair for the primary range axis (null is not permitted), and sends a PlotChangeEvent to all registered listeners.

► public Paint getRangeCrosshairPaint();
Returns the paint (never null) used to draw the crosshair for the primary range axis. The default value is Color.blue.

► public void setRangeCrosshairPaint(Paint paint);
Sets the paint used to draw the crosshair for the primary range axis and sends a PlotChangeEvent to all registered listeners.
```

### 33.44.15 Markers

Markers are used to highlight particular values along the domain axis or the range axis for a plot. Typically, a marker will be represented by a solid line perpendicular to the axis against which it is measured, although custom renderers can alter this default behaviour.

To add a marker along the domain axis:

```
➔ public void addDomainMarker(Marker marker);
```

Adds a marker for the domain axis. This is usually represented as a vertical line on the plot (assuming a vertical orientation for the plot).

To add a marker along the range axis:

```
➔ public void addRangeMarker(Marker marker);
```

Adds a marker for the range axis. This is usually represented as a horizontal line on the plot (assuming a vertical orientation for the plot).

To clear all domain markers:

```
➔ public void clearDomainMarkers();
```

Clears all the domain markers.

Likewise, to clear all range markers:

```
➔ public void clearRangeMarkers();
```

Clears all the range markers.

### 33.44.16 Annotations

You can add annotations to a plot to highlight particular data items. For example, to add the text “Hello World!” to a plot:

```
XYPlot plot = (XYPlot) chart.getPlot();
XYAnnotation annotation = new XYTextAnnotation("Hello World!", 10.0, 25.0);
plot.addAnnotation(annotation);
```

Annotations added directly to the plot are drawn relative to the plot’s primary axes. If you need an annotation drawn against different axes, you should assign the annotation to the appropriate **XYItemRenderer** (see section 37.23.17).

The following methods are used to control the annotations for a plot:

```
➔ public void addAnnotation(XYAnnotation annotation);
```

Adds an annotation to the plot and sends a **PlotChangeEvent** to all registered listeners. If **annotation** is **null**, this method throws an **IllegalArgumentException**. During rendering the annotations are drawn last, so they always overwrite any other items drawn by the plot.

```
➔ public boolean removeAnnotation(XYAnnotation annotation);
```

Removes a specific annotation from the plot and, if the annotation is successfully removed, sends a **PlotChangeEvent** to all registered listeners. If **annotation** is **null**, this method throws an **IllegalArgumentException**. This method returns **true** if **annotation** was removed, and **false** otherwise (this usually indicates that **annotation** was never assigned to the plot in the first place).

```
➔ public List getAnnotations(); [1.0.1]
```

Returns a list of the annotations that have been assigned to the plot. The returned list may be empty, but never **null**. Modifying the returned list has no effect on the plot.

```
➔ public void clearAnnotations();
```

Clears all annotations from the plot and sends a **PlotChangeEvent** to all registered listeners.

### 33.44.17 Zooming

To support the zooming operations implemented in the `ChartPanel` class, this plot implements the `Zoomable` interface:

```
→ public boolean isDomainZoomable();
Returns true to indicate that the domain axis (or axes) are, in fact, zoomable.

→ public boolean isRangeZoomable();
Returns true to indicate that the range axis (or axes) are, in fact, zoomable.

→ public PlotOrientation getOrientation();
Returns the orientation of the plot. The ChartPanel class uses this to determine whether a “horizontal” zoom maps to the domain or range axis, and likewise for a “vertical” zoom.

→ public void zoomDomainAxes(double factor, PlotRenderingInfo state, Point2D source);
Calls resizeRange(double) for each domain axis in the plot. Note that the state and source arguments are currently ignored.

→ public void zoomDomainAxes(double lowerPercent, double upperPercent,
PlotRenderingInfo state, Point2D source);
Calls zoom(lowerPercent, upperPercent) for each domain axis in the plot. Note that the state and source arguments are currently ignored.

→ public void zoomRangeAxes(double factor, PlotRenderingInfo state, Point2D source);
Calls resizeRange(double) for each range axis in the plot. Note that the state and source arguments are currently ignored.

→ public void zoomRangeAxes(double lowerPercent, double upperPercent,
PlotRenderingInfo state, Point2D source);
Calls zoom(lowerPercent, upperPercent) for each range axis in the plot. Note that the state and source arguments are currently ignored.
```

### 33.44.18 Quadrants

The `XYPlot` class provides an optional facility to specify a background color for each quadrant in the plot.

```
→ public Point2D getQuadrantOrigin();
Returns the quadrant origin, in data space. By default, the origin is (0, 0).

→ public void setQuadrantOrigin(Point2D origin);
Sets the quadrant origin, in data space, and sends a PlotChangeEvent to all registered listeners.
This method throws an IllegalArgumentException if origin is null.

→ public Paint getQuadrantPaint(int index);
Returns the paint for the specified quadrant (possibly null). This method throws an Illegal-
ArgumentException if index is not in the range 0 to 3.

→ public void setQuadrantPaint(int index, Paint paint);
Sets the paint for the specified quadrant and sends a PlotChangeEvent to all registered listeners.
```

There is a demo (`PlotOrientationDemo1.java`) in the JFreeChart demo collection that shows this facility in use.

### 33.44.19 Other Methods

Other methods defined by `XYPlot` include:

```
→ public int getWeight();
Returns the weight for a plot which is used to determine how much space to allocate to the
plot when it is being used as a subplot within a combined plot.

→ public void setWeight(int weight);
Sets the weight for the plot and sends a PlotChangeEvent to all registered listeners.
```

The following methods are used by the combined plot classes to align the axes of subplots:

- `public void setFixedDomainAxisSpace(AxisSpace space);`  
Equivalent to `setFixedDomainAxisSpace(space, true)`—see the next method.
- `public void setFixedDomainAxisSpace(AxisSpace space, boolean notify); [1.0.9]`  
Sets the fixed amount of space to reserve for the domain axes on this plot and, if requested, sends a `PlotChangeEvent` to all registered listeners.
- `public void setFixedRangeAxisSpace(AxisSpace space);`  
Equivalent to `setFixedRangeAxisSpace(space, true)`—see the next method.
- `public void setFixedRangeAxisSpace(AxisSpace space, boolean notify); [1.0.9]`  
Sets the fixed amount of space to reserve for the range axes on this plot and, if requested, sends a `PlotChangeEvent` to all registered listeners.

### 33.44.20 Notes

It is possible to display time series data with `XYPlot` by employing a `DateAxis` in place of the usual `NumberAxis`. In this case, the x-values are interpreted as “milliseconds since 1-Jan-1970” as used in `java.util.Date`.

#### See Also

[Plot](#), [XYItemRenderer](#), [CombinedDomainXYPlot](#), [CombinedRangeXYPlot](#).

## 33.45 Zoomable

### 33.45.1 Overview

This interface is designed to allow a caller (in particular, the `ChartPanel` class) to determine the zooming functions supported by a plot, and to invoke those zooming operations as required. This interface is implemented by the following plots:

- [CategoryPlot](#);
- [CombinedDomainCategoryPlot](#);
- [CombinedDomainXYPlot](#);
- [CombinedRangeCategoryPlot](#);
- [CombinedRangeXYPlot](#);
- [FastScatterPlot](#);
- [PolarPlot](#);
- [ThermometerPlot](#);
- [XYPlot](#).

If a plot doesn't implement this interface, the `ChartPanel` will assume that no zooming is possible.

### 33.45.2 Interface Methods

To determine whether or not the domain and range axes are zoomable:

► public boolean isDomainZoomable();  
Returns `true` if the plot's domain (x-axis) is zoomable, and `false` otherwise.

► public boolean isRangeZoomable();  
Returns `true` if the plot's range (y-axis) is zoomable, and `false` otherwise.

To determine the orientation of the plot:

► public PlotOrientation getOrientation();  
Returns the plot's orientation.

To invoke zooming operations:

► public void zoomDomainAxes(double factor, PlotRenderingInfo state, Point2D source);  
Performs a zoom operation on the plot's domain axes.

► public void zoomDomainAxes(double lowerPercent, double upperPercent,  
PlotRenderingInfo state, Point2D source);  
Performs a zoom operation on the plot's domain axes.

► public void zoomRangeAxes(double factor, PlotRenderingInfo state, Point2D source);  
Performs a zoom operation on the plot's range axes.

► public void zoomRangeAxes(double lowerPercent, double upperPercent,  
PlotRenderingInfo state, Point2D source);  
Performs a zoom operation on the plot's range axes.

Two new methods have been added to the interface in JFreeChart version 1.0.7—these allow the caller to specify whether the zooming anchor point is the supplied `source` point, or just the centre of the plot:

► public void zoomDomainAxes(double factor, PlotRenderingInfo state, Point2D source, boolean  
useAnchor); [1.0.7]  
Performs a zoom operation on the plot's domain axes.

► public void zoomRangeAxes(double factor, PlotRenderingInfo state, Point2D source, boolean  
useAnchor); [1.0.7]  
Performs a zoom operation on the plot's range axes.

#### See Also

[ChartPanel](#).

# Chapter 34

## Package: org.jfree.chart.plot.dial

### 34.1 Overview

The `org.jfree.chart.plot.dial` package (new in version 1.0.7) contains a collection of classes dedicated to creating dial plots—see figure 34.1 for an example. The main class in this package is the `DialPlot` class.

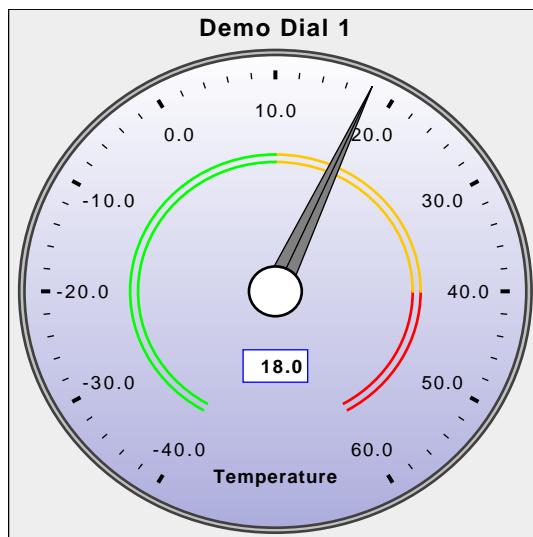


Figure 34.1: Dial Plot

### 34.2 AbstractDialLayer

#### 34.2.1 Overview

A base class for creating the layers that make up a dial. This class provides an event notification mechanism so that changes to a layer can be communicated to the plot that the layer belongs to. Subclasses include:

- `ArcDialFrame`;
- `DialBackground`;
- `DialCap`;

- `DialPointer`;
- `DialTextAnnotation`;
- `DialValueIndicator`;
- `SimpleDialFrame`;
- `StandardDialRange`.
- `StandardDialScale`.

This class implements the `DialLayer` interface.

### 34.2.2 Constructors

To create a new instance:

```
→ protected AbstractDialLayer();
```

Creates a new instance. Note that this constructor is `protected`—it can only be called by subclasses.

### 34.2.3 Attributes

Every layer has a flag that controls the visibility of the layer:

```
→ public boolean isVisible();
```

Returns `true` if the layer is currently visible, and `false` otherwise. The default value is `true`.

```
→ public void setVisible(boolean visible);
```

Sets the flag that controls whether or not the layer is visible, and sends a `DialLayerChangeEvent` to all registered listeners.

### 34.2.4 Event Notification

A typical dial plot is constructed from a number of layers. Whenever a layer is modified, the plot needs to detect this so it can pass on a `PlotChangeEvent` to its listeners.

```
→ public void addChangeListener(DialLayerChangeListener listener);
```

Registers a listener so that it receives change events from this layer.

```
→ public void removeChangeListener(DialLayerChangeListener listener);
```

Deregisters a listener so that it no longer receives change events from this layer.

To determine if a listener is currently registered with a layer:

```
→ public boolean hasListener(EventListener listener);
```

Returns `true` if the specified listener is registered with the layer, and `false` otherwise.

```
→ protected void notifyListeners(DialLayerChangeEvent event);
```

Sends the specified `event` to all registered listeners.

### 34.2.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj);
```

Tests this layer for equality with an arbitrary object. This method returns `true` if and only if:

- `obj` is an instance of `AbstractDialLayer`;
- the `visible` flag is the same for both instances.

Instances of this class are `Cloneable` and `Serializable`.

## 34.3 ArcDialFrame

### 34.3.1 Overview

This frame is an alternative to the [SimpleDialFrame](#) class. Rather than a circular frame, it draws an arc-like frame.

### 34.3.2 Constructors

To create a new instance:

- `public ArcDialFrame();`  
Equivalent to `ArcDialFrame(0, 180.0)`—see the next constructor.
- `public ArcDialFrame(double startAngle, double extent);`  
Creates a new frame spanning the specified arc. The `startAngle` and `extent` are specified in degrees, using the same encoding as Java's `Arc2D` class.

### 34.3.3 General Attributes

The start angle defines where the arc for the frame begins:

- `public double getStartAngle();`  
Returns the start angle, in degrees. The initial value is specified in the constructor.
- `public void setStartAngle(double angle);`  
Sets the start angle and sends a [DialLayerChangeEvent](#) to all registered listeners.

The extent is the angle through which the arc extends:

- `public double getExtent();`  
Returns the extent, in degrees. The initial value is specified in the constructor.
- `public void setExtent(double extent);`  
Sets the extent, in degrees, and sends a [DialLayerChangeEvent](#) to all registered listeners.

The background paint determines the colour used to fill the area between the frame outlines:

- `public Paint getBackgroundPaint();`  
Returns the background paint (never `null`). The default value is `Color.gray`.
- `public void setBackgroundPaint(Paint paint);`  
Sets the background paint for the frame and sends a [DialLayerChangeEvent](#) to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

The foreground paint determines the colour used to draw the frame outline:

- `public Paint getForegroundPaint();`  
Returns the foreground paint (never `null`). The default value is `Color(100, 100, 150)`.
- `public void setForegroundPaint(Paint paint);`  
Sets the foreground paint for the frame and sends a [DialLayerChangeEvent](#) to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the stroke used to draw the frame outline:

- `public Stroke getStroke();`  
Returns the stroke (never `null`) used to draw the frame outline. The default value is `BasicStroke(2.0f)`.
- `public void setStroke(Stroke stroke);`  
Sets the stroke used to draw the frame outline and sends a [DialLayerChangeEvent](#) to all registered listeners. If `stroke` is `null`, this method throws an `IllegalArgumentException`.

To control the radius of the inner edge of the frame:

- `public double getInnerRadius();`  
Returns the inner radius, expressed as a percentage within the dial's framing rectangle. The default value is 0.25.

```
► public void setInnerRadius(double radius);
Sets the inner radius and sends a DialLayerChangeEvent to all registered listeners.
```

To control the radius of the outer edge of the frame:

```
► public double getOuterRadius();
Returns the outer radius, expressed as a percentage within the dial's framing rectangle. The default value is 0.75.

► public void setOuterRadius(double radius);
Sets the outer radius and sends a DialLayerChangeEvent to all registered listeners.
```

### 34.3.4 Other Methods

The following methods are typically called by the [DialPlot](#) class—you shouldn't need to call them directly:

```
► public Shape getWindow(Rectangle2D frame);
Returns the shape for the frame's window. Other layers in the DialPlot may be clipped to this window shape.

► public boolean isClippedToWindow();
Returns false to indicate that this layer should not be clipped to the frame's window.

► public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);
Draws the frame.
```

### 34.3.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this frame for equality with an arbitrary object.
```

Instances of this class are [Cloneable](#) and [Serializable](#).

#### See Also

[SimpleDialFrame](#).

## 34.4 DialBackground

### 34.4.1 Overview

A dial plot layer that fills the background with a single colour or a gradient.

### 34.4.2 Constructors

To create a new instance:

```
► public DialBackground();
Equivalent to DialBackground(Color.white)—see the next constructor.

► public DialBackground(Paint paint);
Creates a new dial background layer with the specified colour. If paint is null, this constructor throws an IllegalArgumentException.
```

### 34.4.3 General Attributes

The main attribute for this class is the background colour:

► public Paint getPaint();

Returns the paint used to fill the background. The default value is specified in the constructor.

► public void setPaint(Paint paint);

Sets the paint used to fill the background, and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

If the background paint is an instance of `GradientPaint`, a transformer is used to dynamically modify the gradient paint coordinates to match the size and shape of the dial plot:

► public GradientPaintTransformer getGradientPaintTransformer();

Returns the current transformer (never `null`). The default value is an instance of `StandardGradientPaintTransformer`.

► public void setGradientPaintTransformer(GradientPaintTransformer t);

Sets the transformer used for `GradientPaint` instances, and sends a `DialLayerChangeEvent` to all registered listeners. If `t` is `null`, this method throws an `IllegalArgumentException`.

### 34.4.4 Other Methods

The plot will call the following method to determine whether the layer is clipped to the dial frame window:

► public boolean isClippedToWindow();

Returns `true`, to indicate that this layer is clipped to the dial frame window.

The plot will call the following method to draw the layer:

► public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);

Draws the background within the specified `frame` and `view` rectangles.

### 34.4.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);

Tests this instance for equality with an arbitrary object.

Instances of this class are cloneable (`PublicCloneable`) and serializable.

### 34.4.6 Notes

Some points to note:

- this class extends `AbstractDialLayer`;

- use the `setBackground()` method in the `DialPlot` class to add an instance of this class to a plot.

## 34.5 DialCap

### 34.5.1 Overview

A small circular graphic that represents the “cap” over the rotation point of a dial’s needle.

### 34.5.2 Constructors

To create a new instance:

► public DialCap();

Creates a new cap with default attributes.

### 34.5.3 General Attributes

The size of the cap is controlled by the radius:

```
► public double getRadius();
```

Returns the radius for the cap, as a percentage of the size of the plot's framing rectangle. The default value is 0.05 (five percent).

```
► public void setRadius(double radius);
```

Sets the radius for the cap and sends a `DialLayerChangeEvent` to all registered listeners.

To control the colour of the cap:

```
► public Paint getFillPaint();
```

Returns the colour used to fill the cap. The default value is `Color.white`.

```
► public void setFillPaint(Paint paint);
```

Sets the colour used to fill the cap and sends a `DialLayerChangeEvent` to all registered listeners.

If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the colour of the cap's outline:

```
► public Paint getOutlinePaint();
```

Returns the colour used to draw the cap outline. The default value is `Color.black`.

```
► public void setOutlinePaint(Paint paint);
```

Sets the colour used to draw the cap outline and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the pen stroke used to draw the cap's outline:

```
► public Stroke getOutlineStroke();
```

Returns the stroke used to draw the cap's outline (never `null`). The default value is `BasicStroke(2.0f)`.

```
► public void setOutlineStroke(Stroke stroke);
```

Sets the stroke used to draw the cap's outline and sends a `DialLayerChangeEvent` to all registered listeners. If `stroke` is `null`, this method throws an `IllegalArgumentException`.

### 34.5.4 Other Methods

The plot will call the following method to determine whether the layer is clipped to the dial frame window:

```
► public boolean isClippedToWindow();
```

Returns `true` to indicate that this layer should be clipped to the dial plot's window.

The plot will call the following method to draw the layer:

```
► public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);
```

Draws the cap relative to the specified `frame` and `view` rectangles.

### 34.5.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
```

Tests this instance for equality with an arbitrary object.

Instances of this class are cloneable (`PublicCloneable`) and serializable.

### 34.5.6 Notes

Some points to note:

- this class extends `AbstractDialLayer`;

- use the `setCap()` method in the `DialPlot` class to add an instance of this class to a plot.

## 34.6 DialFrame

### 34.6.1 Overview

A dial frame is the layer on the `DialPlot` that is drawn last, over top of everything else on the plot. Two implementations of this interface are provided:

- `SimpleDialFrame` – a simple circular dial;
- `StandardDialFrame` – allows more complex dial shapes;

This interface extends `DialLayer`.

### 34.6.2 Interface Methods

This interface adds one method to those inherited from the `DialLayer` interface:

```
➔ public Shape getWindow(Rectangle2D frame);
```

Returns the shape representing the viewing window for the frame. Other layers in the `DialPlot` may be clipped to this window.

### 34.6.3 Notes

Classes that implement this interface should also implement `Serializable`, otherwise serialization for the dial plots will not work.

## 34.7 DialLayer

### 34.7.1 Overview

A `DialPlot` is composed of a number of layers. This interface defines the methods common to all layers. Classes that implement this interface include:

- `DialBackground`;
- `DialCap`;
- `DialPointer`;
- `DialTextAnnotation`;
- `DialValueIndicator`;
- `SimpleDialFrame`;
- `StandardDialFrame`;
- `StandardDialRange`.
- `StandardDialScale`.

See also the `AbstractDialLayer` class.

### 34.7.2 General Methods

All dial layers have a flag that controls visibility:

```
➔ public boolean isVisible();
```

Returns `true` if the layer is currently visible, and `false` otherwise.

### 34.7.3 Change Notification

A simple change notification mechanism is provided by all layers so that the `DialPlot` can track changes to any of its layers:

```
► public void addChangeListener(DialLayerChangeListener listener);
Registers a listener so that it receives notification of changes to this layer.

► public void removeChangeListener(DialLayerChangeListener listener);
Deregisters a listener so that it no longer receives notification of changes to this layer.

► public boolean hasListener(EventListener listener);
Returns true if the specified listener is registered with this layer. This method is provided
mainly for unit testing purposes.
```

You won't typically need to use this mechanism directly.

### 34.7.4 Drawing

The drawing methods are called by the `DialPlot` class, which first determines whether or not the drawing for a layer should be clipped to the plot's window (the window itself is defined by the `DialFrame`):

```
► public boolean isClippedToWindow();
Returns true if drawing should be clipped, and false otherwise.
```

To draw the layer:

```
► public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);
Draws the layer within the specified frame.
```

### 34.7.5 Notes

The `DialFrame` and `DialScale` interfaces extend this interface.

## 34.8 DialLayerChangeEvent

### 34.8.1 Overview

An event that originates from a `DialLayer` and received by a `DialLayerChangeListener`. This is part of the change notification mechanism that allows the `DialPlot` class to track changes to the `DialLayer` instances that it manages. By default, the dial plot will respond to layer changes by forwarding a `PlotChangeEvent` to all its own listeners.

### 34.8.2 Constructors

To create a new instance:

```
► public DialLayerChangeEvent(DialLayer layer);
Creates a new event with the specified layer as the source.
```

### 34.8.3 Methods

To find the layer from which this event originated:

```
► public DialLayer getDialLayer();
Returns the layer from which this event originated.
```

### See Also

[DialLayerChangeListener](#).

## 34.9 DialLayerChangeListener

### 34.9.1 Overview

This interface is implemented by classes that which to receive change event notifications from a [DialLayer](#).

### 34.9.2 Methods

This interface declares a single method:

```
► public void dialLayerChanged(DialLayerChangeEvent event);
```

Called to indicate that a change has been made to a [DialLayer](#) (the layer itself can be retrieved from the `event` instance).

#### See Also

[DialLayerChangeEvent](#).

## 34.10 DialPlot

### 34.10.1 Overview

A plot that displays a value (from a [ValueDataset](#)) in a dial format. The plot is composed of multiple layers, providing a lot of scope to customise the appearance of the plot.

### 34.10.2 Constructors

This class defines two constructors:

```
► public DialPlot();
```

Equivalent to `DialPlot(null)`—see the next constructor.

```
► public DialPlot(ValueDataset dataset);
```

Creates a new plot based on the specified `dataset` (which may be `null`).

### 34.10.3 General Attributes

To control the background for the dial:

```
► public DialLayer getBackground();
```

Returns the background for the plot. The default value is `null`, which means no background is drawn.

```
► public void setBackground(DialLayer background);
```

Sets the background for the dial (`null` is permitted) and sends a [PlotChangeEvent](#) to all registered listeners. If you set the background to `null`, no background is drawn. Any [DialLayer](#) can be used for the background, but a common choice is the [DialBackground](#) class.

The dial frame is a special layer that is drawn last on the dial, possibly covering parts of the layers drawn earlier:

```
► public DialFrame getDialFrame();
```

Returns the dial frame (never `null`). The default value is a new instance of [StandardDialFrame](#).

```
► public void setDialFrame(DialFrame frame);
```

Sets the dial frame and sends a [PlotChangeEvent](#) to all registered listeners.

The cap is a layer that is drawn immediately after the dial's needle is drawn:

```
► public DialLayer getCap();
```

Returns the dial's cap, which may be `null` (which means no cap is drawn). The cap is a layer that is drawn immediately after the dial pointer, and is very often a small shape covering the point where the needle is attached to the dial. The default value is `null`.

► `public void setCap(DialLayer cap);`

Sets the cap and sends a `PlotChangeEvent` to all registered listeners. If you set the cap to `null`, no cap will be drawn.

#### 34.10.4 General Layers

The following methods allow general layers to be added to and removed from a plot:

► `public void addLayer(DialLayer layer);`

Adds a new layer to the plot and sends a `PlotChangeEvent` to all registered listeners. If `layer` is `null`, this method throws an `IllegalArgumentException`.

► `public int getLayerIndex(DialLayer layer);`

Returns the index for the specified layer.

► `public void removeLayer(int index);`

Removes the layer at the specified index.

► `public void removeLayer(DialLayer layer);`

Removes the specified layer from the plot.

#### 34.10.5 Datasets and Scales

The plot displays data from a `ValueDataset` (or possibly multiple datasets):

► `public ValueDataset getDataset();`

Returns the primary dataset for the plot. This may be `null`.

► `public ValueDataset getDataset(int index);`

Returns the dataset at the specified index, or `null` if there is no dataset.

► `public void setDataset(ValueDataset dataset);`

Sets the main dataset for the plot (`null` is permitted) and sends a `PlotChangeEvent` to all registered listeners.

► `public void setDataset(int index, ValueDataset dataset);`

Sets the dataset at the specified index (`dataset` may be `null`) and sends a `PlotChangeEvent` to all registered listeners.

► `public double getValue(int datasetIndex);`

A convenience method that returns the current value for the specified dataset. If there is no dataset at the specified index, this method will return `Double.NaN`.

► `public int getDatasetCount();`

Returns the number of datasets in the plot.

Data values are plotted against a scale. Scales are added to the plot as special layers:

► `public DialScale getScale(int index);`

Returns the scale with the specified index, or `null`.

► `public void addScale(int index, DialScale scale);`

Adds a scale to the plot at the specified index, and sends a `PlotChangeEvent` to all registered listeners.

► `public void mapDatasetToScale(int index, int scaleIndex);`

Maps a dataset to a particular scale (necessary when the plot has multiple scales).

► `public DialScale getScaleForDataset(int datasetIndex);`

Returns the scale that applies to the specified dataset.

### 34.10.6 Pointers

The plot displays a pointer to indicate the current value of the corresponding dataset:

```
➔ public void addPointer(DialPointer pointer);
    Adds a pointer to the plot.

➔ public int getPointerIndex(DialPointer pointer);
    Returns the index of the specified pointer.

➔ public void removePointer(int index);
    Removes the pointer at the specified index.

➔ public void removePointer(DialPointer pointer);
    Removes the specified pointer.
```

### 34.10.7 Frame and View Rectangles

The dimensions of the dial plot are calculated relative to a framing rectangle, but in some cases you only want a subset of the dial to be presented in the plot area—the viewing rectangle defines this visible subset:

```
➔ public double getViewX();
    The relative position of the viewing rectangle's x-coordinate with reference to the framing
    rectangle. This will be a value in the range 0.0 to 1.0, with the default value being 0.0.

➔ public double getViewY();
    Returns the relative position of the viewing rectangle's y-coordinate with reference to the
    framing rectangle. This will be a value in the range 0.0 to 1.0, with the default value being
    0.0.

➔ public double getViewWidth();
    Returns the relative width of the viewing rectangle with reference to the framing rectangle.
    This will be a value in the range 0.0 to 1.0, with the default value being 1.0.

➔ public double getViewHeight();
    Returns the relative height of the viewing rectangle with reference to the framing rectangle.
    This will be a value in the range 0.0 to 1.0, with the default value being 1.0.

➔ public void setView(double x, double y, double w, double h);
    Sets the viewing rectangle relative to the plot's (not known until rendering time) framing
    rectangle, and sends a PlotChangeEvent to all registered listeners.
```

### 34.10.8 Other Methods

A range of other methods in this class are typically called by JFreeChart rather than user code.

```
➔ public String getPlotType();
    Returns a human readable string describing the plot type.
```

A listener method is called whenever a layer belonging to the plot is changed:

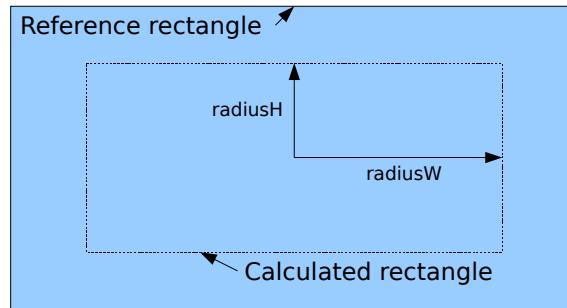
```
➔ public void dialLayerChanged(DialLayerChangeEvent event);
    Receives notification of a change to one of the layers managed by the plot (you shouldn't need
    to call this method yourself). This method responds by forwarding a PlotChangeEvent to all
    registered listeners.
```

A utility method calculates rectangles using radius values that are specified as a percentage of the height and width of a reference rectangle—see figure 34.2:

```
➔ public static Rectangle2D rectangleByRadius(Rectangle2D rect, double radiusW, double radiusH);
    Returns a new rectangle centered on the same point as rect, which the width and height cal-
    culated using the specified radius values (which are expressed as a percentage of the width and
    height of the reference rectangle). Note that 0.75 is interpreted as seventy five percent.
```

JFreeChart will draw the plot by calling the following method:

```
➔ public void draw(Graphics2D g2, Rectangle2D area, Point2D anchor, PlotState parentState,
    PlotRenderingInfo info);
    Draws the plot within the specified area.
```



*Figure 34.2: Calculating a rectangle “by radius”*

### 34.10.9 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
    Tests this plot for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 34.10.10 Notes

Some points to note:

- some demos (`DialPlotDemo1-5.java`) are included in the JFreeChart demo collection.

## 34.11 DialPointer

### 34.11.1 Overview

The base class for a pointer indicating the current value on the dial. This class implements the `DialLayer` interface. Subclasses include:

- `DialPointer.Pin`;
- `DialPointer.Pointer`.

### 34.11.2 Constructors

The two constructors defined by this class are `protected`—they can only be called by subclasses:

```
► protected DialPointer();
    Equivalent to DialPointer(0)—see the next constructor.

► protected DialPointer(int datasetIndex);
    Creates a new pointer associated with the specified dataset.
```

### 34.11.3 General Attributes

The pointer is associated with a specific dataset:

```
► public int getDatasetIndex();
    Returns the dataset index for this pointer. The default value is specified in the constructor.
```

```
► public void setDatasetIndex(int index);
```

Sets the dataset that this pointer is associated with and sends a `DialLayerChangeEvent` to all registered listeners.

The length of the pointer is determined by the radius, which is specified relative to the dial's framing rectangle:

```
► public double getRadius();
```

Returns the radius of the pointer, as a percentage of the dial's framing rectangle. The default value is 0.95 (ninety five percent).

```
► public void setRadius(double radius);
```

Sets the radius of the pointer, and sends a `DialLayerChangeEvent` to all registered listeners.

#### 34.11.4 Other Methods

The `DialPlot` class will call the following method to determine whether or not this layer should be clipped:

```
► public boolean isClippedToWindow();
```

Returns `true` to indicate that this layer is clipped to the dial frame window.

#### 34.11.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
```

Tests this pointer for equality with an arbitrary object.

### 34.12 DialPointer.Pin

#### 34.12.1 Overview

A dial pointer that is drawn as a thin straight line. This class extends `DialPointer` and implements the `DialLayer` interface.

#### 34.12.2 Constructors

To create a new instance:

```
► public Pin();
```

Equivalent to `Pin(0)`—see the next constructor.

```
► public Pin(int datasetIndex);
```

Creates a new dial pointer associated with the specified dataset. By default, the pointer is drawn as a thin line (`BasicStroke(3.0f)`) in red.

#### 34.12.3 General Attributes

To control the colour of the pointer:

```
► public Paint getPaint();
```

Returns the paint (never `null`) used to draw the pointer. The default value is `Color.red`.

```
► public void setPaint(Paint paint);
```

Sets the paint used to draw the pointer, and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the stroke used to draw the pointer:

```
► public Stroke getStroke();
```

Returns the stroke used to draw the pointer. The default value is `BasicStroke(3.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_BEVEL)`.

► `public void setStroke(Stroke stroke);`

Sets the stroke used to draw the pointer, and sends a `DialLayerChangeEvent` to all registered listeners. If `stroke` is `null`, this method throws an `IllegalArgumentException`.

#### 34.12.4 Other Methods

The `DialPlot` will call the following method to draw the pointer:

► `public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);`  
Draws the pointer.

#### 34.12.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`  
Tests this pointer for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### 34.12.6 Notes

The length of the pointer is controlled by the inherited radius attribute.

### See Also

[DialPointer.Pointer](#).

### 34.13 DialPointer.Pointer

#### 34.13.1 Overview

A dial pointer that is drawn in a long triangular shape.

#### 34.13.2 Constructors

To create a new instance:

► `public Pointer();`  
Equivalent to `Pointer(0)`—see the next constructor.

► `public Pointer(int datasetIndex);`  
Creates a new pointer that is associated with the specified dataset.

#### 34.13.3 General Attributes

The width of the base of the pointer is specified as a radius, in percentage terms relative to the

► `public double getWidthRadius();`

Returns the radius used to calculate the width of the base of the pointer. The default value is 0.05 (five percent).

► `public void setWidthRadius(double radius);`

Sets the radius used to calculate the width of the base of the pointer, and sends a `DialLayerChangeEvent` to all registered listeners.

To control the paint used to fill the pointer:

► `public Paint getFillPaint(); [1.0.8]`

Returns the paint (never `null`) used to fill the pointer. The default value is `Color.gray`.

► public void setFillPaint(Paint paint); [1.0.8]  
 Sets the paint used to fill the pointer and sends a [DialLayerChangeEvent](#) to all registered listeners.  
 If `paint` is `null`, this method throws an [IllegalArgumentException](#).

To control the paint used to draw the pointer outline:

► public Paint getOutlinePaint(); [1.0.8]  
 Returns the paint (never `null`) used to draw the outline of the paint. The default value is `Color.black`.  
 ► public void setOutlinePaint(Paint paint); [1.0.8]  
 Sets the paint used to draw the pointer outline and sends a [DialLayerChangeEvent](#) to all registered listeners. If `paint` is `null`, this method throws an [IllegalArgumentException](#).

### 34.13.4 Other Methods

The [DialPlot](#) will call the following method during chart rendering—you won't normally need to call this method directly:

► public void draw(Graphics2D g2, [DialPlot](#) plot, Rectangle2D frame, Rectangle2D view)  
 Draws the pointer.

### 34.13.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);  
 Tests this pointer for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### See Also

[DialPointer.Pin](#).

## 34.14 DialScale

### 34.14.1 Overview

A `DialScale` is a specialised [DialLayer](#) that has the ability to convert data values into a corresponding angle, and vice versa. The [DialPlot](#) uses this scale to position a dial pointer that indicates the current value of the dataset that is mapped to this scale. A typical dial scale implementation will include drawing code to display the scale within the dial (see [StandardDialScale](#), for example).

### 34.14.2 Interface Methods

To convert a data value to an angle:

► public double valueToAngle(double value);  
 Returns the angle (in degrees) that corresponds to the specified value.

To convert an angle to a data value:

► public double angleToValue(double angle);  
 Returns the value that corresponds to the given angle (which is specified in degrees).

### 34.14.3 Notes

This interface extends the [DialLayer](#) interface.

#### See Also

[StandardDialScale](#).

## 34.15 DialTextAnnotation

### 34.15.1 Overview

A dial layer that draws a text string at an arbitrary location on a dial. For example, in figure 34.1, the “Temperature” label is drawn by an instance of this class.

### 34.15.2 Constructors

To create a new annotation:

```
➔ public DialTextAnnotation(String label);
```

Creates a new text annotation with default attributes. If `label` is `null`, this constructor throws an `IllegalArgumentException`.

### 34.15.3 General Attributes

To control the text displayed by the annotation:

```
➔ public String getLabel();
```

Returns the text (never `null`) displayed by the annotation. The initial value is specified in the constructor.

```
➔ public void setLabel(String label);
```

Sets the text to be displayed by the annotation and sends a `DialLayerChangeEvent` to all registered listeners. If `label` is `null`, this method throws an `IllegalArgumentException`.

To control the font used to display the label:

```
➔ public Font getFont();
```

Returns the font (never `null`) used to display the label. The default value is `Font("Dialog", Font.BOLD, 14)`.

```
➔ public void setFont(Font font);
```

Sets the font used to display the label and sends a `DialLayerChangeEvent` to all registered listeners. If `font` is `null`, this method throws an `IllegalArgumentException`.

To control the foreground colour of the label:

```
➔ public Paint getPaint();
```

Returns the paint (never `null`) used to display the label. The default value is `Color.black`.

```
➔ public void setPaint(Paint paint);
```

Sets the paint used to display the label and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

The anchor point for the text label is defined by an angle and a radius:

```
➔ public double getAngle();
```

Returns the angle (in degrees) that determines the text anchor point. The encoding is the same as for Java’s `Arc2D` class. The default value is `-90.0`.

```
➔ public void setAngle(double angle);
```

Sets the angle (in degrees) that determines the text anchor point, and sends a `DialLayerChangeEvent` to all registered listeners.

```
➔ public double getRadius();
```

Returns the radius as a percentage relative to the dial’s framing rectangle. The default value is `0.30` (thirty percent).

```
➔ public void setRadius(double radius);
```

Sets the radius as a percentage of the dial’s framing rectangle and sends a `DialLayerChangeEvent` to all registered listeners.

Having determined the anchor point, the text is aligned according to its anchor:

```
➔ public TextAnchor getAnchor();
Returns the text anchor (never null) for the annotation. The default value is TextAnchor.TOP_CENTER.

➔ public void setAnchor(TextAnchor anchor);
Sets the text anchor and sends a DialLayerChangeEvent to all registered listeners. If anchor is
null, this method throws an IllegalArgumentException.
```

### 34.15.4 Other Methods

The following methods are typically called by the `DialPlot` class during chart rendering, you won't normally call these methods directly:

```
➔ public boolean isClippedToWindow();
Returns true to indicate that this layer should be clipped to the dial frame window.

➔ public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);
Draws the text annotation.
```

### 34.15.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
Tests this text annotation for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

## 34.16 DialValueIndicator

### 34.16.1 Overview

A dial layer that displays the value from a dataset.

### 34.16.2 Constructors

To create a new instance:

```
➔ public DialValueIndicator();
Equivalent to DialValueIndicator(0)—see the next constructor.

➔ public DialValueIndicator(int datasetIndex);
Creates a new instance that will display the value from the specified dataset.
```

### 34.16.3 General Attributes

To control the dataset from which the indicator obtains the current value:

```
➔ public int getDatasetIndex();
Returns the index of the dataset from which the indicator obtains the current value. The initial
value is specified in the constructor.

➔ public void setDatasetIndex(int index);
Sets the index of the dataset from which the indicator obtains the current value and sends a
DialLayerChangeEvent to all registered listeners.
```

To control the formatter used to convert the dataset value into a string for display purposes:

```
➔ public NumberFormat getNumberFormat();
Returns the number formatter (never null) used to convert the dataset value into a String for
the display. The default value is DecimalFormat("0.0").
```

► public void setNumberFormat(NumberFormat formatter);

Sets the number formatter used to convert the dataset value into a `String` for the display, and sends a `DialLayerChangeEvent` to all registered listeners. If `formatter` is `null`, this method throws an `IllegalArgumentException`.

To control the font used to display the dataset value:

► public Font getFont();

Returns the font (never `null`) used to display the dial value. The default value is `Font("Dialog", Font.BOLD, 14)`.

► public void setFont(Font font);

Sets the font used to display the dial value and sends a `DialLayerChangeEvent` to all registered listeners. If `font` is `null`, this method throws an `IllegalArgumentException`.

To control the colour used to display the dataset value:

► public Paint getPaint();

Returns the paint (never `null`) used to display the dial value. The default value is `Color.black`.

► public void setPaint(Paint paint);

Sets the paint used to display the dial value, and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the background colour for the indicator:

► public Paint getBackgroundPaint();

Returns the paint (never `null`) used to fill the background of the indicator. The default value is `Color.white`.

► public void setBackgroundPaint(Paint paint);

Sets the paint used to fill the background of the indicator and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the stroke used to draw the outline for the indicator:

► public Stroke getOutlineStroke();

Returns the stroke (never `null`) used to draw the outline around the indicator. The default value is `BasicStroke(1.0f)`.

► public void setOutlineStroke(Stroke stroke);

Sets the stroke used to draw the outline around the indicator and sends a `DialLayerChangeEvent` to all registered listeners. If `stroke` is `null`, this method throws an `IllegalArgumentException`.

To control the colour used to draw the outline for the indicator:

► public Paint getOutlinePaint();

Returns the paint (never `null`) used to draw the outline around the indicator. The default value `Color.blue`.

► public void setOutlinePaint(Paint paint);

Sets the paint used to draw the outline around the indicator and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

#### 34.16.4 Indicator Position

The anchor point for positioning the indicator is specified in terms of an angle and a radius. To control the angle:

► public double getAngle();

Returns the angle, in degrees, for calculating the anchor point. The default value is `-90.0`, which corresponds to six o'clock on a clock face.

► public void setAngle(double angle);

Sets the angle (in degrees, using the same encoding as Java's `Arc2D` class) for calculating the anchor point used to position the indicator, and sends a `DialLayerChangeEvent` to all registered listeners.

To control the radius:

► public double getRadius();

Returns the radius, relative to the dial's framing rectangle. The default value is 0.3.

► public void setRadius(double radius);

Sets the radius used to calculate the anchor point for the indicator, and sends a `DialLayerChangeEvent` to all registered listeners. The radius is specified as a percentage relative to the dial's framing rectangle.

The value indicator is displayed as a text item in a frame. The frame anchor specifies the point on this frame that will be aligned to the anchor point (which is calculated from the angle and radius described previously):

► public `RectangleAnchor` setFrameAnchor();

Returns the frame anchor (never `null`). The default value is `RectangleAnchor.CENTER`.

► public void setFrameAnchor(`RectangleAnchor` anchor);

Sets the frame anchor and sends a `DialLayerChangeEvent` to all registered listeners. If `anchor` is `null`, this method throws an `IllegalArgumentException`.

### 34.16.5 Indicator Dimensions

To ensure that the value indicator has a fixed width regardless of the current value in the dataset, a template value is used to calculate the dimensions of the indicator. This template value will be formatted using the current number formatter, then the dimensions of the string will be used to determine the dimensions of the value indicator:

► public Number getTemplateValue();

Returns the template value (never `null`). The default value is `Double(100.0)`.

► public void setTemplateValue(Number value);

Sets the template value and sends a `DialLayerChangeEvent` to all registered listeners. If `value` is `null`, this method throws an `IllegalArgumentException`.

You should set the template value to the largest number that is likely to be displayed in the indicator (if negative values are possible, make the template negative to allow for the sign in the formatted string).

To control the white space around the indicator value (but within the outline):

► public `RectangleInsets` getInsets();

Returns the insets (never `null`) which determine the white-space around the indicator value. The default value is `RectangleInsets(4, 4, 4, 4)`.

► public void setInsets(`RectangleInsets` insets);

Sets the insets which determine the white space around the indicator value and sends a `DialLayerChangeEvent` to all registered listeners.

### 34.16.6 Aligning the Indicator Value

The value anchor determines the point to which the value text will be aligned—you'll typically set this to `RectangleAnchor.LEFT` or `RectangleAnchor.RIGHT`:

► public `RectangleAnchor` getValueAnchor();

Returns the token (never `null`) which determines the anchor point for the value text. The default value is `RectangleAnchor.RIGHT`.

► public void setValueAnchor(`RectangleAnchor` anchor);

Sets the token that determines the anchor point for the value text, and sends a `DialLayerChangeEvent` to all registered listeners. If `anchor` is `null`, this method throws an `IllegalArgumentException`.

The text anchor determines the point on the text that will be aligned to the value anchor:

```
► public TextAnchor getTextAnchor();
Returns the token (never null) that determines the point on the text that is aligned to the value anchor. The default value is TextAnchor.CENTER_RIGHT.
```

```
► public void setTextAnchor(TextAnchor anchor);
Sets the token that determines the point on the text that is aligned to the value anchor, and sends a DialLayerChangeEvent to all registered listeners. If anchor is null, this method throws an IllegalArgumentException.
```

### 34.16.7 Other Methods

The following methods are typically called by the `DialPlot` class during chart rendering, you won't normally call these methods directly:

```
► public boolean isClippedToWindow();
Returns true to indicate that this layer should be clipped to the dial frame window.
```

```
► public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);
Draws the dial value indicator.
```

### 34.16.8 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this indicator for equality with an arbitrary object.
```

### 34.16.9 Notes

To add an indicator to a `DialPlot`, use the `addLayer()` method.

## 34.17 SimpleDialFrame

### 34.17.1 Overview

A plain circular dial frame. This class implements the `DialFrame` interface.

### 34.17.2 Constructors

To create a new instance:

```
► public SimpleDialFrame();
Creates a new dial frame with default attributes.
```

### 34.17.3 General Attributes

To control the radius of the dial frame:

```
► public double getRadius();
Returns the radius, expressed as a percentage relative to the framing rectangle. The default value is 0.95 (ninety-five percent).
```

```
► public void setRadius(double radius);
Sets the radius of the circular dial, as a percentage relative to the framing rectangle, and sends a DialLayerChangeEvent to all registered listeners.
```

To control the background paint:

```
► public Paint getBackgroundPaint();
Returns the paint (never null) used to fill the space between the double lines around the outer edge of the circle. The default value is Color.gray.
```

► public void setBackgroundPaint(Paint paint);  
 Sets the paint used to fill the space between the double lines around the edge of the dial, and sends a [DialLayerChangeEvent](#) to all registered listeners. If `paint` is `null`, this method throws an [IllegalArgumentException](#).

To control the foreground paint:

► public Paint getForegroundPaint();  
 Returns the paint (never `null`) used to draw the double lines around the outside of the dial. The default value is `Color.black`.  
 ► public void setForegroundPaint(Paint paint);  
 Sets the paint used to draw the double lines around the outside of the circular dial, and sends a [DialLayerChangeEvent](#) to all registered listeners. If `paint` is `null`, this method throws an [IllegalArgumentException](#).

To control the stroke used to draw the dial outline:

► public Stroke getStroke();  
 Returns the stroke (never `null`) used to draw the dial outline. The default value is `BasicStroke(2.0f)`.  
 ► public void setStroke(Stroke stroke);  
 Sets the stroke used to draw the dial outline and sends a [DialLayerChangeEvent](#) to all registered listeners. If `stroke` is `null`, this method throws an [IllegalArgumentException](#).

#### 34.17.4 Other Methods

The plot will fetch the window for this dial frame using the following method:

► public Shape getWindow(Rectangle2D frame);  
 Returns the window for this frame, in this case an `Ellipse2D` that fits within the specified `frame`.  
 ► public boolean isClippedToWindow();  
 Returns `false`—this layer does not need to be clipped.  
 ► public void draw(Graphics2D g2, [DialPlot](#) plot, Rectangle2D frame, Rectangle2D view);  
 Draws the dial frame within the specified `frame` and `view` rectangles.

#### 34.17.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);  
 Tests this frame for equality with an arbitrary object.

Instances of this class are cloneable ([PublicCloneable](#)) and serializable.

#### See Also

[StandardDialFrame](#).

### 34.18 StandardDialRange

#### 34.18.1 Overview

A layer for a [DialPlot](#) that highlights (statically) a range of values. You could use this to indicate ranges on a dial such as, for example, “normal”, “high” and “extreme”.

### 34.18.2 Constructors

This class defines two constructors:

- ▶ `public StandardDialRange();`  
Equivalent to `StandardDialRange(0.0, 100.0, Color.white)`—see the next constructor.
- ▶ `public StandardDialRange(double lower, double upper, Paint paint);`  
Creates a new instance with the specified bounds. If `paint` is `null`, this constructor throws an `IllegalArgumentException`.

### 34.18.3 General Attributes

The range is measured relative to a specific scale on the [DialPlot](#):

- ▶ `public int getScaleIndex();`  
Returns the scale index. The default value is 0.
- ▶ `public void setScaleIndex(int index);`  
Sets the scale index and sends a [DialLayerChangeEvent](#) to all registered listeners.

The range is primarily defined by its lower and upper bounds, which are specified in data units:

- ▶ `public double getLowerBound();`  
Returns the lower bound for the range, in data units.
- ▶ `public void setLowerBound(double bound);`  
Sets the lower bound for the range and sends a [DialLayerChangeEvent](#) to all registered listeners. If `bound` is not less than `getUpperBound()`, this method throws an `IllegalArgumentException`.
- ▶ `public double getUpperBound();`  
Returns the upper bound for the range, in data units.
- ▶ `public void setUpperBound(double bound);`  
Sets the upper bound for the range and sends a [DialLayerChangeEvent](#) to all registered listeners. If `bound` is not greater than `getLowerBound()`, this method throws an `IllegalArgumentException`.
- ▶ `public void setBounds(double lower, double upper);`  
Sets the bounds for the range and sends a [DialLayerChangeEvent](#) to all registered listeners. If `lower` is not less than `upper`, this method throws an `IllegalArgumentException`.

To control the colour used to highlight the range:

- ▶ `public Paint getPaint();`  
Returns the paint (never `null`) used to highlight the range. The initial value is specified in the constructor.
- ▶ `public void setPaint(Paint paint);`  
Sets the paint used to highlight the range and sends a [DialLayerChangeEvent](#) to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

The range highlights are drawn at a specified radius within the dial:

- ▶ `public double getInnerRadius();`  
Returns the radius for the inner highlight on the range. The default value is 0.48.
- ▶ `public void setInnerRadius(double radius);`  
Sets the radius for the inner highlight and sends a [DialLayerChangeEvent](#) to all registered listeners.
- ▶ `public double getOuterRadius();`  
Returns the radius for the outer highlight on the range. The default value is 0.52.
- ▶ `public void setOuterRadius(double radius);`  
Sets the radius for the outer highlight and sends a [DialLayerChangeEvent](#) to all registered listeners.

### 34.18.4 Other Methods

The following methods are typically called by the `DialPlot` class during chart rendering, you won't normally call these methods directly:

► `public boolean isClippedToWindow();`

Returns `true` to indicate that this layer should be clipped to the dial frame window.

► `public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);`

Draws the range indicator.

### 34.18.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`

Tests this dial range for equality with an arbitrary object.

Instances of this class are `Cloneable` ([PublicCloneable](#)) and `Serializable`.

## 34.19 StandardDialScale

### 34.19.1 Overview

A standard scale that can be used on a `DialPlot`. The scale can display major and minor tick marks at user-specified intervals, and numerical labels for the major tick marks.

### 34.19.2 Constructors

To create a new instance:

► `public StandardDialScale();`

Equivalent to `StandardDialScale(0.0, 100.0, 175, -170, 10.0, 4)`—see the next constructor.

► `public StandardDialScale(double lowerBound, double upperBound, double startAngle,`

`double extent, double majorTickIncrement, int minorTickCount);`

Creates a new dial scale covering the values `lowerBound` to `upperBound`.

### 34.19.3 General Attributes

To control the arc through which the scale is drawn on the dial, you can specify the starting angle and the extent (in degrees, using the same encoding as Java's `Arc2D` class):

► `public double getStartAngle();`

Returns the angle (in degrees) for the starting point of the scale. The initial value is specified in the constructor.

► `public void setStartAngle(double angle);`

Sets the angle (in degrees, same encoding as Java's `Arc2D` class) for the starting point of the scale, and sends a `DialLayerChangeEvent` to all registered listeners.

To control the extent:

► `public double getExtent();`

Returns the extent (in degrees) for the scale. The initial value is specified in the constructor.

► `public void setExtent(double extent);`

Sets the extent (in degrees, same encoding as Java's `Arc2D` class) for the scale, and sends a `DialLayerChangeEvent` to all registered listeners.

To control the lower bound:

► `public double getLowerBound(); [1.0.8]`

Returns the lower bound for the scale.

► public void setLowerBound(double lower); [1.0.8]

Sets the lower bound for the scale and sends a `DialLayerChangeEvent` to all registered listeners.

To control the upper bound:

► public double getUpperBound(); [1.0.8]

Returns the upper bound for the scale.

► public void setUpperBound(double upper); [1.0.8]

Sets the upper bound for the scale and sends a `DialLayerChangeEvent` to all registered listeners.

### 34.19.4 Tick Marks and Labels

The tick radius determines how far out on the dial the tick marks are drawn (typically they will appear near the outer edge of the dial, but for a dial that shows more than one scale you might want to show a scale closer to the centre):

► public double getTickRadius();

Returns the radius as a percentage of the dial's framing rectangle. The default value is 0.70 (seventy percent).

► public void setTickRadius(double radius);

Sets the radius used to position the tick marks, and sends a `DialLayerChangeEvent` to all registered listeners.

The tick radius defines the reference point for the major and minor tick marks, as well as the labels for the major tick marks. These are all described in the following sections.

### 34.19.5 Major Tick Marks

The scale will draw tick marks at regular (user-definable) intervals along the range. To control the interval between tick marks:

► public double getMajorTickIncrement();

Returns the interval between major tick marks. The initial value is specified in the constructor.

► public void setMajorTickIncrement(double increment);

Sets the interval between major tick marks and sends a `DialLayerChangeEvent` to all registered listeners.

To control the length of the major tick marks:

► public double getMajorTickLength();

Returns the length of the major tick marks, expressed as an amount to be subtracted from the tick radius (see `getTickRadius()`). The default value is 0.04.

► public void setMajorTickLength(double length);

Sets the length of the major tick marks and sends a `DialLayerChangeEvent` to all registered listeners. The `length` must be greater than or equal() to zero.

To control the paint used to draw the major tick marks:

► public Paint getMajorTickPaint();

Returns the paint (never null) used to draw the major tick marks. The default value is `Color.black`.

► public void setMajorTickPaint(Paint paint);

Sets the paint used to draw the major tick marks and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is null, this method throws an `IllegalArgumentException`.

To control the stroke used to draw the major tick marks:

► public Stroke getMajorTickStroke();

Returns the stroke (never null) used to draw the major tick marks. The default value is `BasicStroke(3.0f)`.

► public void setMajorTickStroke(Stroke stroke);

Sets the stroke used to draw the major tick marks, and sends a `DialLayerChangeEvent` to all registered listeners. If `stroke` is null, this method throws an `IllegalArgumentException`.

### 34.19.6 Tick Labels

The scale will display numerical labels for the major tick marks. There is a flag to enable/disable this feature:

► public boolean getTickLabelsVisible();

Returns the flag that controls whether or not the tick labels are visible. The default value is `true`.

► public void setTickLabelsVisible(boolean visible);

Sets the flag that controls whether or not the tick labels are visible and sends a `DialLayerChangeEvent` to all registered listeners.

The anchor point for each label is determined using the tick radius (see `getTickRadius()`) adjusted by an offset value. Once the anchor point is determined, the value label is centred on the anchor point:

► public double getTickLabelOffset();

Returns the tick label offset. The default value is `0.10`.

► public void setTickLabelOffset(double offset);

Sets the tick label offset and sends a `DialLayerChangeEvent` to all registered listeners.

To control the font used to display the tick labels:

► public Font getTickLabelFont();

Returns the font (never `null`) used to display the tick labels. The default value is `Font("Dialog", Font.BOLD, 16)`.

► public void setTickLabelFont(Font font);

Sets the font used to display the tick labels, and sends a `DialLayerChangeEvent` to all registered listeners. If `font` is `null`, this method throws an `IllegalArgumentException`.

To control the foreground colour of the tick labels:

► public Paint getTickLabelPaint();

Returns the paint (never `null`) used to display the tick labels. The default value is `Color.black`.

► public void setTickLabelPaint(Paint paint);

Sets the paint used to display the tick labels, and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

There is a flag that controls whether or not the first tick label is displayed—this is intended for dials that are fully circular (in which case the first and last labels occupy the same position):

► public boolean isFirstTickLabelVisible();

Returns the flag that controls whether or not the first tick label is visible. The default value is `true`.

► public void setFirstTickLabelVisible(boolean visible);

Sets the flag that controls whether or not the first tick label is visible, and sends a `DialLayerChangeEvent` to all registered listeners.

### 34.19.7 Minor Tick Marks

The scale can display minor tick marks (with no labels).

► public int getMinorTickCount();

Returns the number of minor tick marks to display between each pair of major tick marks. The initial value is specified in the constructor.

► public void setMinorTickCount(int count);

Sets the minor tick count and sends a `DialLayerChangeEvent` to all registered listeners. If `count` is less than zero, this method throws an `IllegalArgumentException`.

To control the length of the minor tick marks:

► `public double getMinorTickLength();`

Returns the length of minor tick marks, as a radius specified as a percentage of the dial's framing rectangle. The default value is 0.02 (two percent).

► `public void setMinorTickLength(double length);`

Sets the minor tick length and sends a `DialLayerChangeEvent` to all registered listeners. If you set the length to 0.0, no minor tick marks will be drawn.

To control the paint used for the minor tick marks:

► `public Paint getMinorTickPaint();`

Returns the paint (never `null`) used for the minor tick marks. The default value is `Color.black`.

► `public void setMinorTickPaint(Paint paint);`

Sets the paint used to display the minor tick marks, and sends a `DialLayerChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

To control the stroke used for the minor tick marks:

► `public Stroke getMinorTickStroke(); [1.0.8]`

Returns the stroke (`null`) used for the minor tick marks. The default value is `BasicStroke(1.0f)`.

► `public void setMinorTickStroke(Stroke stroke); [1.0.8]`

Sets the stroke used to draw the minor tick marks and sends a `DialLayerChangeEvent` to all registered listeners. If `stroke` is `null`, this method throws an `IllegalArgumentException`.

### 34.19.8 Conversion Methods

The following methods are used to convert from data values to angles and back again—these are used by the plot, and you won't typically need to call these methods yourself:

► `public double valueToAngle(double value);`

Returns the angle (in degrees) that corresponds to the specified data value. This is a function of the lower and upper bounds for the scale, and the starting angle and extent.

► `public double angleToValue(double angle);`

Returns the value that corresponds to the given angle (which is specified in degrees). This is a function of the starting angle and extent, and the lower and upper bounds for the scale.

### 34.19.9 Other Methods

The following methods are typically called by the `DialPlot` class during chart rendering, you won't normally call these methods directly:

► `public boolean isClippedToWindow();`

Returns `true` to indicate that the scale should be clipped to the dial frame's window.

► `public void draw(Graphics2D g2, DialPlot plot, Rectangle2D frame, Rectangle2D view);`

Draws the scale within the specified `frame` and `view` rectangles.

### 34.19.10 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`

Tests this scale for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

# Chapter 35

## Package: org.jfree.chart.renderer

### 35.1 Overview

This package contains interfaces and classes that are used to implement renderers, plug-in objects that are responsible for drawing individual data items on behalf of a plot.

Renderers offer a lot of scope for changing the appearance of your charts, either by changing the attributes of an existing renderer, or by implementing a completely new renderer.

### 35.2 AbstractRenderer

#### 35.2.1 Overview

An abstract class that provides support for the features common to all renderer implementations:

- colors, line styles and shapes for each series (section 35.2.3);
  - paint (section 35.2.5);
  - fill paint (section 35.2.6);
  - outline paint (section 35.2.7);
  - stroke (section 35.2.8);
  - outline stroke (section 35.2.9);
  - shape (section 35.2.11);
- series visibility (section 35.2.12);
- series in legend visibility (section 35.2.13);
- item labels (section 35.2.14);
  - item labels visible (section 35.2.15);
  - item label font (section 35.2.16);
  - item label paint (section 35.2.17);
  - positive item label positions (section 35.2.18);
  - negative item label positions (section 35.2.19);
- chart entity generation (section 35.2.21);

This base class is extended by:

- `AbstractCategoryItemRenderer`;
- `AbstractXYItemRenderer`.

### 35.2.2 Per Series Attribute Mechanism

Many renderer attributes need to have a value defined for each series in the dataset that is assigned to the renderer. In addition to the per-series attributes, JFreeChart will typically define a “base” (or default) attribute value, that will be used in the event that no attribute value is defined for a particular series.

In version 1.0.5 and earlier, many attributes also define an override setting which takes precedence over the per-series and base settings. This, however, has caused confusion and is mostly unnecessary, so the overrides settings have been deprecated from version 1.0.6 onwards.<sup>1</sup>

### 35.2.3 Common Attributes

All renderers use a common set of attributes as listed in Table 35.1.

Attribute:	Description:
<code>paint</code>	The paint override ( <code>null</code> permitted, deprecated from 1.0.6 onwards).
<code>paintList</code>	A list of paints that apply to individual series (only referenced if <code>paint</code> is <code>null</code> ).
<code>autoPopulateSeriesPaint</code>	A flag that controls whether or not the per-series settings are auto-populated (the default is <code>true</code> ).
<code>basePaint</code>	The paint that is used if there is no other setting.
<code>fillPaint</code>	The fill paint override ( <code>null</code> permitted, deprecated from 1.0.6 onwards).
<code>fillPaintList</code>	A list of fill paints that apply to individual series (only referenced if <code>paint</code> is <code>null</code> ).
<code>autoPopulateSeriesXXX</code>	A flag that controls whether or not the per-series settings are auto-populated.
<code>baseFillPaint</code>	The fill paint that is used if there is no other setting.
<code>outlinePaint</code>	The outline paint override ( <code>null</code> permitted, deprecated from 1.0.6 onwards).
<code>outlinePaintList</code>	A list of outline paints that apply to individual series (only referenced if <code>outlinePaint</code> is <code>null</code> ).
<code>autoPopulateSeriesXXX</code>	A flag that controls whether or not the per-series settings are auto-populated.
<code>baseOutlinePaint</code>	The outline paint that is used if there is no other setting.
<code>stroke</code>	The stroke override ( <code>null</code> permitted, deprecated from 1.0.6 onwards).
<code>strokeList</code>	A list of stroke objects that apply to individual series (only referenced if <code>stroke</code> is <code>null</code> ).
<code>autoPopulateSeriesXXX</code>	A flag that controls whether or not the per-series settings are auto-populated.
<code>baseStroke</code>	The stroke that is used if there is no other setting.
<code>outlineStroke</code>	The outline stroke override ( <code>null</code> permitted, deprecated from 1.0.6 onwards).
<code>outlineStrokeList</code>	A list of outline strokes that apply to individual series (only referenced if <code>outlineStroke</code> is <code>null</code> ).
<code>autoPopulateSeriesXXX</code>	A flag that controls whether or not the per-series settings are auto-populated.
<code>baseOutlineStroke</code>	The outline stroke override.
<code>shape</code>	The shape override ( <code>null</code> permitted, deprecated from 1.0.6 onwards).
<code>shapeList</code>	A list of shapes that apply to individual series (only referenced if <code>shape</code> is <code>null</code> ).
<code>autoPopulateSeriesXXX</code>	A flag that controls whether or not the per-series settings are auto-populated.
<code>baseShape</code>	The shape that is used if there is no other setting.

Table 35.1: *Attributes for the AbstractRenderer class*

<sup>1</sup>You can still use them, but they may be removed in a future release of JFreeChart.

### 35.2.4 Setting Series Colours

Renderers are responsible for drawing the data items within a plot, so this class provides attributes for controlling the colours that will be used. Colours are typically defined on a “per series” basis, and stored in a lookup table.

There is a default mechanism to automatically populate the lookup table with default colours (using the `DrawingSupplier` interface). However, you can manually update the paint list at any time. First, you need to obtain a reference to the renderer(s) (note that many charts do not use more than one renderer). Here is the code for a `CategoryPlot`:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
AbstractRenderer r1 = (AbstractRenderer) plot.getRenderer(0);
AbstractRenderer r2 = (AbstractRenderer) plot.getRenderer(1);
```

The code is similar for charts that use `XYPlot`:

```
XYPlot plot = (XYPlot) chart.getPlot();
AbstractRenderer r1 = (AbstractRenderer) plot.getRenderer(0);
AbstractRenderer r2 = (AbstractRenderer) plot.getRenderer(1);
```

To update the series paint used by a renderer:

```
// change the paint for series 0, 1 and 2...
r1.setSeriesPaint(0, Color.red);
r1.setSeriesPaint(1, Color.green);
r1.setSeriesPaint(2, Color.blue);
```

### 35.2.5 Paint

The paint attribute defines the main colour used to identify a series in a chart:

Attribute:	Description:
<code>paint</code>	The paint override ( <code>null</code> permitted). <i>Deprecated since 1.0.6</i> .
<code>paintList</code>	A list of paints that apply to individual series (only referenced if <code>paint</code> is <code>null</code> ).
<code>autoPopulateSeriesPaint</code>	A flag that controls whether or not the per-series settings are auto-populated (the default is <code>true</code> ).
<code>basePaint</code>	The paint that is used if there is no other setting.

Table 35.2: *Paint attributes for the `AbstractRenderer` class*

► `public Paint getItemPaint(int row, int column);`

This method is called by JFreeChart to obtain a paint instance for each item that the renderer draws. By default, it calls `lookupSeriesPaint(row)` which checks the current paint setting for the series. If this is `null` and `autoPopulateSeriesPaint` is `true`, the drawing supplier is queried for a new paint instance, otherwise the default paint (`getBasePaint()`) is returned.

You can override this method to return an arbitrary colour for any data item.

► `public Paint getSeriesPaint(int series);`

Returns the paint to use for all items in the specified series.

Some renderers will also use the fill and/or outline paint attributes for the renderer.

#### Override Paint

The override paint attribute can be used to override the per-series and base settings, but is seldom used in practice (it defaults to `null`):

► `public void setPaint(Paint paint); [Deprecated, 1.0.6]`

Equivalent to `setPaint(paint, true)`, see the next method for details.

► **public void setPaint(Paint paint, boolean notify); [Deprecated, 1.0.6]**  
 Sets the override paint attribute and, if requested, sends a `RendererChangeEvent` to all registered listeners.

A call to `setPaint(Paint)` forces the renderer to use the same colour for ALL series in a dataset. To achieve the same result with non-deprecated methods, you need to set the base paint and also switch off the auto-population of the per-series paint:

```
renderer.setBasePaint(Color.BLUE);
renderer.setAutoPopulateSeriesPaint(false);
```

### Per Series Paint

The paint attribute is typically defined on a per-series basis with the following methods:

► **public void setSeriesPaint(int series, Paint paint);**  
 Equivalent to `setSeriesPaint(series, paint, true)`, see the next method for details.

► **public void setSeriesPaint(int series, Paint paint, boolean notify);**  
 Sets the paint for a series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

### Base Paint

The base paint is used as the default when the override and series paint settings are `null`:

► **public Paint getBasePaint();**  
 Returns the base paint (never `null`). The default value is defined by `AbstractRenderer.DEFAULT_PAINT` (`Color.blue`).

► **public void setBasePaint(Paint paint);**  
 Equivalent to `setBasePaint(paint, true)`, see the next method for details.

► **public void setBasePaint(Paint paint, boolean notify);**  
 Sets the base paint and, if requested, sends a `RendererChangeEvent` to all registered listeners.  
 An `IllegalArgumentException` is thrown if `paint` is `null`.

### 35.2.6 Fill Paint

The fill paint attribute defines the color used to fill shapes that are drawn by the renderer:

Attribute:	Description:
<code>fillPaint</code>	The fill paint override ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>fillPaintList</code>	A list of fill paints that apply to individual series (only referenced if <code>fillPaint</code> is <code>null</code> ).
<code>baseFillPaint</code>	The fill paint that is used if there is no other setting.

Table 35.3: Fill paint attributes for the `AbstractRenderer` class

► **public Paint getItemFillPaint(int row, int column);**  
 This method is called to obtain the fill paint for each item that the renderer draws. By default, it simply returns the series fill paint obtained by calling `getSeriesFillPaint(row)`. However, you can override this method to return an arbitrary color for any data item.

► **public Paint getSeriesFillPaint(int series);**  
 Returns the fill paint to use for all items in the specified series.

Not all renderers will use the fill paint attribute.

## Override Fill Paint

The override fill paint attribute can be used to override the per-series and base settings (the default is `null`):

- `public void setFillPaint(Paint paint); [Deprecated, 1.0.6.]`  
Equivalent to `setFillPaint(paint, true)`, see the next method for details.
- `public void setFillPaint(Paint paint, boolean notify); [Deprecated, 1.0.6.]`  
Sets the override fill paint attribute and, if requested, sends a `RendererChangeEvent` to all registered listeners.

## Per Series Fill Paint

The fill paint attribute is typically defined on a per-series basis with the following methods:

- `public void setSeriesFillPaint(int series, Paint paint);`  
Equivalent to `setSeriesFillPaint(series, paint, true)`, see the next method for details.
- `public void setSeriesFillPaint(int series, Paint paint, boolean notify);`  
Sets the fill paint for a series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

## Base Fill Paint

The base fill paint is used as the default when the override and series paint settings are `null`:

- `public Paint getBaseFillPaint();`  
Returns the base fill paint (never `null`). The default value is `Color.white`.
- `public void setBaseFillPaint(Paint paint);`  
Equivalent to `setBaseFillPaint(paint, true)`, see the next method for details.
- `public void setBaseFillPaint(Paint paint, boolean notify);`  
Sets the base fill paint and, if requested, sends a `RendererChangeEvent` to all registered listeners.  
An `IllegalArgumentException` is thrown if `paint` is `null`.

### 35.2.7 OutlinePaint

The outline paint attribute defines the color used to outline shapes that are drawn by the renderer:

Attribute:	Description:
<code>outlinePaint</code>	The outline paint override ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>outlinePaintList</code>	A list of outline paints that apply to individual series (only referenced if <code>outlinePaint</code> is <code>null</code> ).
<code>baseOutlinePaint</code>	The outline paint that is used if there is no other setting.

Table 35.4: *Outline paint attributes for the AbstractRenderer class*

► `public Paint getItemOutlinePaint(int row, int column);`  
This method is called to obtain the outline paint for each item that the renderer draws. By default, it simply returns the series outline paint obtained by calling `getSeriesOutlinePaint(row)`. However, you can override this method to return an arbitrary color for any data item.

► `public Paint getSeriesOutlinePaint(int series);`  
Returns the outline paint to use for all items in the specified series.

## Override Outline Paint

The override outline paint attribute can be used to override the per-series and base settings (the default is `null`):

- ➔ `public void setOutlinePaint(Paint paint); [Deprecated, 1.0.6]`  
Equivalent to `setOutlinePaint(paint, true)`, see the next method for details.
- ➔ `public void setOutlinePaint(Paint paint, boolean notify); [Deprecated, 1.0.6]`  
Sets the override outline paint attribute and, if requested, sends a `RendererChangeEvent` to all registered listeners.

## Per Series Outline Paint

The outline paint attribute is typically defined on a per-series basis with the following methods:

- ➔ `public void setSeriesOutlinePaint(int series, Paint paint);`  
Equivalent to `setSeriesOutlinePaint(series, paint, true)`, see the next method for details.
- ➔ `public void setSeriesOutlinePaint(int series, Paint paint, boolean notify);`  
Sets the outline paint for a series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

## Base Outline Paint

The base outline paint is used as the default when the override and series paint settings are `null`:

- ➔ `public Paint getBaseOutlinePaint();`  
Returns the base paint (never `null`). The default value is defined by `AbstractRenderer.DEFAULT_OUTLINE_PAINT` (`Color.gray`).
- ➔ `public void setBaseOutlinePaint(Paint paint);`  
Equivalent to `setBaseOutlinePaint(paint, true)`, see the next method for details.
- ➔ `public void setBaseOutlinePaint(Paint paint, boolean notify);`  
Sets the base outline paint and, if requested, sends a `RendererChangeEvent` to all registered listeners. An `IllegalArgumentException` is thrown if `paint` is `null`.

### 35.2.8 Stroke

The stroke attributes control the pen style (width and dash pattern among other things) used by the renderer for drawing lines:

- ➔ `public Stroke getItemStroke(int row, int column);`  
Returns the stroke used for the specified item. This method simply returns the result from `getSeriesStroke(row)`, but you can override it to implement a different behaviour.
- ➔ `public Stroke getSeriesStroke(int series);`  
Returns the stroke to use for the specified series.

Some renderers won't use this at all, while other renderers may also make use of the outline stroke attribute—refer to the documentation for the specific renderer for more details.

## Override Stroke

The override stroke attribute can be used to override the per-series and base settings, but is seldom used in practice (it defaults to `null`):

- ➔ `public void setStroke(Stroke stroke); [Deprecated, 1.0.6]`  
Equivalent to `setStroke(stroke, true)`, see the next method for details.
- ➔ `public void setStroke(Stroke stroke, boolean notify); [Deprecated, 1.0.6]`  
Sets the override stroke and, if requested, sends a `RendererChangeEvent` to all registered listeners.

## Per Series Stroke

The stroke attribute is typically defined on a per series basis using the following method:

- ▶ `public void setSeriesStroke(int series, Stroke stroke);`  
Equivalent to `setSeriesStroke(series, stroke, true)`, see the next method for details.
- ▶ `public void setSeriesStroke(int series, Stroke stroke, boolean notify);`  
Sets the stroke for the specified series and, if requested, sends a `RendererChangeEvent` to all registered listeners. You can set the stroke for a series to `null`, in which case the base stroke will be used for that series.

## Base Stroke

The base stroke setting is used for any series where both the override and per series settings are not specified:

- ▶ `public Stroke getBaseStroke();`  
Returns the base stroke (never `null`). The default value is defined by `AbstractRenderer.DEFAULT_STROKE` (`BasicStroke(1.0f)`).
- ▶ `public void setBaseStroke(Stroke stroke);`  
Equivalent to `setBaseStroke(stroke, true)`, see the next method for details.
- ▶ `public void setBaseStroke(Stroke stroke, boolean notify);`  
Sets the base stroke and, if requested, sends a `RendererChangeEvent` to all registered listeners.

### 35.2.9 Outline Stroke

The outline stroke attributes control the pen style (width and dash pattern among other things) used by the renderer for drawing shape outlines:

- ▶ `public Stroke getItemOutlineStroke(int row, int column); [Deprecated, 1.0.6]`  
Returns the outline stroke used for the specified item. This method simply returns the result from `getSeriesOutlineStroke(row)`, but you can override it to implement a different behaviour.
- ▶ `public Stroke getSeriesOutlineStroke(int series); [Deprecated, 1.0.6]`  
Returns the outline stroke to use for the specified series.

Some renderers won't use this at all—refer to the documentation for the specific renderer for more details.

## Override Outline Stroke

The override outline stroke attribute can be used to override the per-series and base settings, but is seldom used in practice (it defaults to `null`):

- ▶ `public void setOutlineStroke(Stroke stroke);`  
Equivalent to `setOutlineStroke(stroke, true)`, see the next method for details.
- ▶ `public void setOutlineStroke(Stroke stroke, boolean notify);`  
Sets the override outline stroke and, if requested, sends a `RendererChangeEvent` to all registered listeners.

## Per Series Outline Stroke

The outline stroke attribute is typically defined on a per series basis using the following method:

- ▶ `public void setSeriesOutlineStroke(int series, Stroke stroke);`  
Equivalent to `setSeriesOutlineStroke(series, stroke, true)`, see the next method for details.
- ▶ `public void setSeriesOutlineStroke(int series, Stroke stroke, boolean notify);`  
Sets the outline stroke for the specified series and, if requested, sends a `RendererChangeEvent` to all registered listeners. You can set the outline stroke for a series to `null`, in which case the base outline stroke will be used for that series.

## Base Outline Stroke

The base outline stroke setting is used for any series where both the override and per series settings are not specified:

```
► public Stroke getBaseOutlineStroke();
Returns the base outline stroke (possibly null). The default value is is BasicStroke(1.0f)
(defined by the constant AbstractRenderer.DEFAULT_OUTLINE_STROKE).

► public void setBaseOutlineStroke(Stroke stroke);
Equivalent to setBaseOutlineStroke(stroke, true), see the next method for details.

► public void setBaseOutlineStroke(Stroke stroke, boolean notify);
Sets the base outline stroke and, if requested, sends a RendererChangeEvent to all registered
listeners. If you set this to null, the renderer will not draw an outline.
```

### 35.2.10 Setting Series Shapes

Renderers are initialised so that a range of default shapes are available if required. These are stored in a lookup table that is initially empty. The lookup table has two rows (one for the primary dataset, and one for the secondary dataset), and can have any number of columns (one per series). When the renderer requires a **Shape**, it uses the dataset index (primary or secondary) and the series index to read a shape from the lookup table. If the value is **null**, then the renderer turns to the **DrawingSupplier** for a new shape—the next shape is returned by the **getNextShape()** method.

If you require more control over the shapes that are used for your plots, you can populate the lookup table yourself using the **setSeriesShape(...)** method. The shape you supply can be any instance of **Shape**, but should be centered on  $(0, 0)$  in Java2D space (so that JFreeChart can position the shape at any data point).

Here is some sample code that sets four custom shapes for the primary dataset in an **XYPlot**:

```
XYPlot plot = chart.getXYPlot();
XYItemRenderer r = plot.getRenderer();
if (r instanceof StandardXYItemRenderer) {
    StandardXYItemRenderer renderer = (StandardXYItemRenderer) r;
    renderer.setPlotShapes(true);
    renderer.setDefaultValueFilled(true);
    renderer.setSeriesShape(0, new Ellipse2D.Double(-3.0, -3.0, 6.0, 6.0));
    renderer.setSeriesShape(1, new Rectangle2D.Double(-3.0, -3.0, 6.0, 6.0));
    GeneralPath s2 = new GeneralPath();
    s2.moveTo(0.0f, -3.0f);
    s2.lineTo(3.0f, 3.0f);
    s2.lineTo(-3.0f, 3.0f);
    s2.closePath();
    renderer.setSeriesShape(2, s2);
    GeneralPath s3 = new GeneralPath();
    s3.moveTo(-1.0f, -3.0f);
    s3.lineTo(1.0f, -3.0f);
    s3.lineTo(1.0f, -1.0f);
    s3.lineTo(3.0f, -1.0f);
    s3.lineTo(3.0f, 1.0f);
    s3.lineTo(1.0f, 1.0f);
    s3.lineTo(1.0f, 3.0f);
    s3.lineTo(-1.0f, 3.0f);
    s3.lineTo(-1.0f, 1.0f);
    s3.lineTo(-3.0f, 1.0f);
    s3.lineTo(-3.0f, -1.0f);
    s3.lineTo(-1.0f, -1.0f);
    s3.closePath();
    renderer.setSeriesShape(3, s3);
}
```

### 35.2.11 Shape

Many renderers (though not all) will display a shape at each data point, so this class includes a mechanism for storing the shapes that will be used. Be aware that any shapes you supply should be centred around the origin  $(0, 0)$ , as this is assumed in the code that translates shapes into position at chart rendering time. For example, a rectangle centred at the origin would be **Rectangle(-3.0, -4.0, 6.0, 8.0)**:

```
► public Shape getItemShape(int row, int column);
Returns the shape to use for the specified item. This method simply returns the value from
getSeriesShape(row), but you can override this if you want to return a custom shape for a
specific item.

► public Shape getSeriesShape(int series);
Returns the shape to use for items in the specified series.
```

## Override Shape

The override shape, if non-null, applies to all series:

```
► public void setShape(Shape shape); [Deprecated, 1.0.6]
Equivalent to setShape(shape, true), see the next method for details.

► public void setShape(Shape shape, boolean notify); [Deprecated, 1.0.6]
Sets the override shape and, if requested, sends a RendererChangeEvent to all registered listeners.
Set this to null for no override.
```

## Series Shapes

The renderer can store shape attributes on a per-series basis:

```
► public void setSeriesShape(int series, Shape shape);
Equivalent to setSeriesShape(series, shape, true), see the next method for details.

► public void setSeriesShape(int series, Shape shape, boolean notify);
Sets the shape for the specified series and, if requested, sends a RendererChangeEvent to all
registered listeners. You can set the shape to null, in which case the base shape (see the next
section) will be used for the specified series.
```

## Base Shape

The base shape is used as the default if no override or series shape is specified:

```
► public Shape getBaseShape();
Returns the current base shape (never null). The default base shape is a Rectangle(-3.0, -3.0,
6.0, 6.0).

► public void setBaseShape(Shape shape);
Equivalent to setBaseShape(shape, true), see the next method for details.

► public void setBaseShape(Shape shape, boolean notify);
Sets the base shape and, if requested, sends a RendererChangeEvent to all registered listeners.
An IllegalArgumentException is thrown if shape is null.
```

### 35.2.12 Series Visibility

By default, a renderer will display all the series in a dataset, but it is possible to change this behaviour by changing the series visibility flags—see table 35.5.<sup>2</sup>

Attribute:	Description:
<code>seriesVisible</code>	The override flag (null permitted). This field is redundant and deprecated from version 1.0.6 onwards.
<code>seriesVisibleList</code>	A list of flags that apply to individual series (these are only referenced if <code>seriesVisible</code> is null).
<code>baseSeriesVisible</code>	The default visibility for all series.

Table 35.5: *Series visibility attributes for the `AbstractRenderer` class*

To determine the current visibility of an item or series:

<sup>2</sup>Note that not all renderers respect these flags yet, but eventually they all will.

► `public boolean getItemVisible(int series, int item);`  
 Returns `true` if the specified item should be displayed by the renderer, and `false` otherwise. The default implementation of this method just returns the value from `isSeriesVisible(series)`. For different behaviour, subclass the renderer you are using and override this method.

► `public boolean isSeriesVisible(int series);`  
 Returns `true` if the specified series is visible, and `false` otherwise. This method checks the override, per-series and base settings as appropriate.

### Override Flag

An override flag is provided for the series visibility. This is mainly for completeness, typically you won't need to set this (it defaults to `null`). This flag is deprecated from version 1.0.6 onwards.

► `public Boolean getSeriesVisible(); [Deprecated 1.0.6]`  
 Returns the override flag for series visibility. If this is non-null, the `isSeriesVisible(int)` method returns the boolean value of this flag.

► `public void setSeriesVisible(Boolean visible); [Deprecated 1.0.6]`  
 Equivalent to `setSeriesVisible(visible, true)`, see the method below.

► `public void setSeriesVisible(Boolean visible, boolean notify); [Deprecated 1.0.6]`  
 Sets the override flag for series visibility and, if requested, sends a `RendererChangeEvent` to all registered listeners. If you don't need an override, set this flag to `null`.

### Per Series Flags

The per-series visibility flags allow complete control over which series are displayed in a chart (provided that the renderer subclass supports this feature):

► `public Boolean getSeriesVisible(int series);`  
 Returns the flag that controls the visibility of the specified series. This can be `null`, in which case the base level setting will apply.

► `public void setSeriesVisible(int series, Boolean visible);`  
 Calls `setSeriesVisible(series, visible, true)`—see below.

► `public void setSeriesVisible(int series, Boolean visible, boolean notify);`  
 Sets the visibility flag for the specified series and, if requested, sends a `RendererChangeEvent` to all registered listeners. This flag can be set to `null`, in which case the base level flag applies.

### Base Flag

The base flag, which defaults to `true`, applies when there is no override flag set and no series level flag set:

► `public boolean getBaseSeriesVisible();`  
 Returns the default series visibility.

► `public void setBaseSeriesVisible(boolean visible);`  
 Calls `setBaseSeriesVisible(visible, true)`. See the next method description for details.

► `public void setBaseSeriesVisible(boolean visible, boolean notify);`  
 Sets the default series visibility and, if requested, sends a `RendererChangeEvent` to all registered listeners.

### 35.2.13 Series Visibility in Legend

By default, a renderer will create legend items for all series when asked (see the `createLegendItems()` method implemented by `AbstractCategoryItemRenderer` and `AbstractXYItemRenderer`), but it is possible to change this behaviour by changing the series visibility flags—see table 35.6.

To determine the current visibility of a series in the legend:

Attribute:	Description:
<code>seriesVisibleInLegend</code>	The override flag ( <code>null</code> permitted). <i>Deprecated as of version 1.0.6.</i>
<code>seriesVisibleInLegendList</code>	A list of flags that apply to individual series (these are only referenced if <code>seriesVisibleInLegend</code> is <code>null</code> ).
<code>baseSeriesVisibleInLegend</code>	The default legend visibility for all series.

Table 35.6: *Legend visibility attributes for the AbstractRenderer class*

► `public boolean isSeriesVisibleInLegend(int series);`  
 Returns `true` if the specified series is visible, and `false` otherwise. This method checks the override, per-series and base level flags as appropriate (take care not to confuse this method with `getSeriesVisibleInLegend(int)`, which only returns the series level flag).

### Override Flag

An override flag is provided for the series visibility in the legend. This is mainly for completeness, typically you won't need to set this (it defaults to `null`):

► `public Boolean getSeriesVisibleInLegend(); [Deprecated, 1.0.6]`  
 Returns the override flag for series visibility in the legend. This may be `null`.  
 ► `public void setSeriesVisibleInLegend(Boolean visible); [Deprecated, 1.0.6]`  
 Equivalent to `setSeriesVisibleInLegend(visible, true)`, see the next method for details.  
 ► `public void setSeriesVisibleInLegend(Boolean visible, boolean notify); [Deprecated, 1.0.6]`  
 Sets the override flag and, if requested, sends a `RendererChangeEvent` to all registered listeners.  
 You can set the override flag to `null` (the default) if you want the per-series flags to take effect.

### Per Series Settings

The per-series visibility flags allow complete control over which series are displayed in the legend:

► `public Boolean getSeriesVisibleInLegend(int series);`  
 Returns the per-series flag controlling visibility in the legend, for the specified series. Don't confuse this method with the similarly named `isSeriesVisibleInLegend(int)` method.  
 ► `public void setSeriesVisibleInLegend(int series, Boolean visible);`  
 Equivalent to `setSeriesVisibleInLegend(series, visible, true)`, see the next method for details.  
 ► `public void setSeriesVisibleInLegend(int series, Boolean visible, boolean notify);`  
 Sets the flag controlling visibility of the specified series in the legend and, if requested, sends a `RendererChangeEvent` to all registered listeners. If the flag for a series is set to `null`, the value defined by the base flag (see the next section) will apply for that series when calling the `isSeriesVisibleInLegend(int)` method.

### Base Flag

The base flag provides the default visibility for a series in the legend. This value (the default is `true`) is used only if the override and per-series flags are both `null`:

► `public boolean getBaseSeriesVisibleInLegend();`  
 Returns the base flag controlling the visibility of series in the legend. This value is returned by the `isSeriesVisibleInLegend(int)` method when both the override flag and the series flag are `null`.  
 ► `public void setBaseSeriesVisibleInLegend(boolean visible);`  
 Equivalent to `setBaseSeriesVisibleInLegend(visible, true)`, see the next method for details.  
 ► `public void setBaseSeriesVisibleInLegend(boolean visible, boolean notify);`  
 Sets the base flag controlling the visibility of series in the legend and sends a `RendererChangeEvent` to all registered listeners.

### 35.2.14 Item Label Attributes

All renderers use a common set of item label attributes (some renderers may ignore these settings):

Attribute:	Description:
<code>itemLabelsVisible</code>	The <code>itemLabelsVisible</code> override flag ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelsVisibleList</code>	A list of flags that apply to individual series (only referenced if <code>itemLabelsVisible</code> is <code>null</code> ).
<code>baseItemLabelsVisible</code>	The flag that is used if there is no other setting.
<code>itemLabelFont</code>	The <code>itemLabelFont</code> override ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelFontList</code>	A list of fonts that apply to individual series (only referenced if <code>itemLabelFont</code> is <code>null</code> ).
<code>baseItemLabelFont</code>	The font that is used if there is no other setting.
<code>itemLabelPaint</code>	The <code>itemLabelPaint</code> override ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelPaintList</code>	A list of paints that apply to individual series (only referenced if <code>itemLabelPaint</code> is <code>null</code> ).
<code>baseItemLabelPaint</code>	The font that is used if there is no other setting.
<code>itemLabelAnchor</code>	The <code>itemLabelAnchor</code> override ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelAnchorList</code>	A list of anchors that apply to individual series (only referenced if <code>itemLabelAnchor</code> is <code>null</code> ).
<code>baseItemLabelAnchor</code>	The anchor that is used if there is no other setting.
<code>itemLabelTextAnchor</code>	The <code>itemLabelTextAnchor</code> override ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelTextAnchorList</code>	A list of text anchors that apply to individual series (only referenced if <code>itemLabelTextAnchor</code> is <code>null</code> ).
<code>baseItemLabelTextAnchor</code>	The text anchor that is used if there is no other setting.
<code>itemLabelRotationAnchor</code>	The <code>itemLabelRotationAnchor</code> override ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelRotationAnchorList</code>	A list of rotation anchors that apply to individual series (only referenced if <code>itemLabelRotationAnchor</code> is <code>null</code> ).
<code>baseItemLabelRotationAnchor</code>	The anchor that is used if there is no other setting.
<code>itemLabelAngle</code>	The <code>itemLabelAngle</code> override ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelAngleList</code>	A list of angles that apply to individual series (only referenced if <code>itemLabelAngle</code> is <code>null</code> ).
<code>baseItemLabelAngle</code>	The angle that is used if there is no other setting.

Table 35.7: Attributes for the `AbstractRenderer` class

### 35.2.15 Item Label Visibility

The `item label visibility` attributes control the visibility of individual item labels:

Attribute:	Description:
<code>itemLabelsVisible</code>	The override visibility flag ( <code>null</code> permitted).
<code>itemLabelsVisibleList</code>	A list of flags that apply to individual series (these are only referenced if <code>itemLabelsVisible</code> is <code>null</code> ).
<code>baseItemLabelsVisible</code>	The default item label visibility for all series.

Table 35.8: *Item label visibility attributes for the `AbstractRenderer` class*

```
➔ public boolean isItemLabelVisible(int row, int column);
```

Returns `true` if the item label is visible for the specified item, and `false` otherwise. Note that

the `row` index corresponds to the series and the `column` index corresponds to the item within the series (or the category). The default implementation of this method simply returns the value from `isSeriesItemLabelsVisible(row)`, but you can override the method to control label visibility on a per-item basis.

► `public boolean isSeriesItemLabelsVisible(int series);`  
 Returns `true` if item labels are visible for the specified series, and `false` otherwise.

### Override Item Label Visibility

The override item labels visible attribute can be used to override the per-series and base settings, but is seldom used in practice (it defaults to `null`):

► `public void setItemLabelsVisible(boolean visible); [Deprecated, 1.0.6]`  
 Equivalent to `setItemLabelsVisible(Boolean.valueOf(visible))`, see the next method for details.  
 ► `public void setItemLabelsVisible(Boolean visible); [Deprecated, 1.0.6]`  
 Equivalent to `setItemLabelsVisible(visible, true)`, see the next method for details.  
 ► `public void setItemLabelsVisible(Boolean visible, boolean notify); [Deprecated, 1.0.6]`  
 Sets the override item labels visible flag and, if requested, sends a `RendererChangeEvent` to all registered listeners.

### Per Series Item Label Visibility

The item labels attribute can be defined on a per series basis using the following methods:

► `public void setSeriesItemLabelsVisible(int series, boolean visible);`  
 Equivalent to `setSeriesStroke(series, Boolean.valueOf(visible))`, see the next method for details.  
 ► `public void setSeriesItemLabelsVisible(int series, Boolean visible);`  
 Equivalent to `setSeriesStroke(series, Boolean.valueOf(visible), true)`, see the next method for details.  
 ► `public void setSeriesItemLabelsVisible(int series, Boolean visible, boolean notify);`  
 Sets the item labels visible flag for the specified series and, if requested, sends a `RendererChangeEvent` to all registered listeners. You can set the flag for a series to `null`, in which case the base item labels visible flag will be used for that series.

### Base Item Label Visibility

The base item labels visible setting is used for any series where both the override and per series settings are not specified:

► `public Boolean getBaseItemLabelsVisible();`  
 Returns the base item labels flag. A `null` value should be interpreted as `Boolean.FALSE`.<sup>3</sup>  
 ► `public void setBaseItemLabelsVisible(boolean visible);`  
 Equivalent to `setBaseItemLabelsVisible(Boolean.valueOf(visible))`, see the next method for details.  
 ► `public void setBaseItemLabelsVisible(Boolean visible);`  
 Equivalent to `setBaseItemLabelsVisible(visible, true)`, see the next method for details.  
 ► `public void setBaseItemLabelsVisible(Boolean visible, boolean notify);`  
 Sets the base item labels visible flag and, if requested, sends a `RendererChangeEvent` to all registered listeners. You can set this flag to `null`, but that will be interpreted as equivalent to `Boolean.FALSE`.

Attribute:	Description:
<code>itemLabelFont</code>	The override font ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelFontList</code>	A list of fonts that apply to individual series (these are only referenced if <code>itemLabelFont</code> is <code>null</code> ).
<code>baseItemLabelFont</code>	The default item label font for all series.

Table 35.9: *Item label font attributes for the AbstractRenderer class*

### 35.2.16 Item Label Font

The item label font attributes control the font that is used by the renderer to draw item labels. For most applications, a single font will be used for all labels, so you can just set the base item label font, and leave the override and per series settings at their default `null` values.

To determine the item label font:

```
► public Font getItemLabelFont(int row, int column);
```

Returns the item label font for the specified item.

#### Override Item Label Font

The override item label font overrides the per series and base level settings, unless it is `null` (which is the default).

```
► public Font getItemLabelFont(); [Deprecated, 1.0.6]
```

Returns the override item label font. The default value is `null`.

```
► public void setItemLabelFont(Font font); [Deprecated, 1.0.6]
```

Equivalent to `setItemLabelFont(font, true)`—see the next method.

```
► public void setItemLabelFont(Font font, boolean notify); [Deprecated, 1.0.6]
```

Sets the override item label font and, if requested, sends a `RendererChangeEvent` to all registered listeners.

#### Per Series Item Label Font

To control the item label font on a per series basis:

```
► public Font getSeriesItemLabelFont(int series);
```

Returns the item label font for the specified series, or `null` if no font has been explicitly set for the series.

```
► public void setSeriesItemLabelFont(int series, Font font);
```

Equivalent to `setSeriesItemLabelFont(series, font, true)`—see the next method.

```
► public void setSeriesItemLabelFont(int series, Font font, boolean notify);
```

Sets the item label font for a series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

#### Base Item Label Font

The base item label font is used when no per series or override settings are in place (which is the default):

```
► public Font getBaseItemLabelFont();
```

Returns the base item label font (never `null`). The default is `Font("SansSerif", Font.PLAIN, 10)`.

```
► public void setBaseItemLabelFont(Font font);
```

Equivalent to `setBaseItemLabelFont(font, true)`—see the next method.

```
► public void setBaseItemLabelFont(Font font, boolean notify);
```

Sets the base item label font and, if requested, sends a `RendererChangeEvent` to all registered listeners.

---

<sup>3</sup>This field should have been defined as a `boolean`, but is now part of the published API so we have to support it.

### 35.2.17 Item Label Paint

The item label paint attributes control the paint that is used by the renderer to draw item labels. For most applications, a single paint will be used for all labels, so you can just set the base item label paint, and leave the override and per series settings at their default `null` values.

Attribute:	Description:
<code>itemLabelPaint</code>	The override paint ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>itemLabelPaintList</code>	A list of paints that apply to individual series (these are only referenced if <code>itemLabelPaint</code> is <code>null</code> ).
<code>baseItemLabelPaint</code>	The default item label paint for all series.

Table 35.10: *Item label paint attributes for the AbstractRenderer class*

To determine the item label paint for an item:

► `public Paint getItemLabelPaint(int row, int column);`  
 Returns the item label paint (never `null`) for an item.

#### Override Item Label Paint

The override item label font overrides the per series and base settings, unless it is `null` (which is the default):

► `public Paint getItemLabelPaint(); [Deprecated, 1.0.6]`  
 Returns the override item label font. The default value is `null`.  
 ► `public void setItemLabelPaint(Paint paint); [Deprecated, 1.0.6]`  
 Equivalent to `setItemLabelPaint(paint, true)`—see the next method.  
 ► `public void setItemLabelPaint(Paint paint, boolean notify); [Deprecated, 1.0.6]`  
 Sets the override item label paint and, if requested, sends a `RendererChangeEvent` to all registered listeners.

#### Per Series Item Label Paint

To control the item label paint on a per series basis:

► `public Paint getSeriesItemLabelPaint(int series);`  
 Returns the item label paint (possibly `null`) for a series.  
 ► `public void setSeriesItemLabelPaint(int series, Paint paint);`  
 Equivalent to `setSeriesItemLabelPaint(series, paint, true)`—see the next method.  
 ► `public void setSeriesItemLabelPaint(int series, Paint paint, boolean notify);`  
 Sets the item label paint for a series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

#### Base Item Label Paint

The base item label paint is the default paint used to draw the item labels (if they are visible). It is used when there is no per-series or override setting:

► `public Paint getBaseItemLabelPaint();`  
 Returns the default item label paint. The default value is `Color.black`.  
 ► `public void setBaseItemLabelPaint(Paint paint);`  
 Equivelant to `setBaseItemLabelPaint(paint, true)`—see the next method.  
 ► `public void setBaseItemLabelPaint(Paint paint, boolean notify);`  
 Sets the base item label paint and, if `notify` is `true`, sends a `RendererChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

### 35.2.18 Positive Item Label Position

The positive item label position attributes (see table 35.11) control the position of item labels for those items that have a data value that is positive.

Attribute:	Description:
<code>positiveItemLabelPosition</code>	The <code>positiveItemLabelPosition</code> override flag ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>seriesPositiveItemLabelPositionList</code>	A list of positions that apply to individual series (only referenced if <code>positiveItemLabelPosition</code> is <code>null</code> ).
<code>basePositiveItemLabelPosition</code>	The base position for all series, the default value is <code>ItemLabelPosition(ItemLabelAnchor.OUTSIDE12, TextAnchor.BOTTOM_CENTER)</code>

Table 35.11: Positive item label position attributes

To determine the label position for an item with a positive value, JFreeChart calls the following method:

```
➔ public ItemLabelPosition getPositiveItemLabelPosition(int row, int column);  
Returns the position for the specified item. The return value is derived from the override, per  
series and base level settings.
```

#### Override Positive Item Label Position

The override positive item label position attribute provides a mechanism to override all other settings (this is rarely required):

```
➔ public ItemLabelPosition getPositiveItemLabelPosition(); [Deprecated, 1.0.6]  
Returns the positive item label position override. The default value is null.  
  
➔ public void setPositiveItemLabelPosition(ItemLabelPosition position); [Deprecated, 1.0.6]  
Equivalent to setPositiveItemLabelPosition(position, true)—see the next method.  
  
➔ public void setPositiveItemLabelPosition(ItemLabelPosition position, boolean notify); [Deprecated,  
1.0.6]  
Sets the positive item label position for the specified series and, if requested, sends a RendererChangeEvent  
to all registered listeners.
```

#### Per Series Positive Item Label Position

The per series positive item label positions apply when no override is set.

```
➔ public ItemLabelPosition getSeriesPositiveItemLabelPosition(int series);  
Returns the positive item label position for the specified series (this may be null).  
  
➔ public void setSeriesPositiveItemLabelPosition(int series, ItemLabelPosition position);  
Equivalent to setSeriesPositiveItemLabelPosition(series, position, true)—see the next method.  
  
➔ public void setSeriesPositiveItemLabelPosition(int series, ItemLabelPosition position,  
boolean notify);  
Sets the positive item label position for the specified series and, if requested, sends a RendererChangeEvent  
to all registered listeners.
```

#### Base Positive Item Label Position

The base setting for the positive item label position attribute is used when no per series or override setting is specified:

```
➔ public ItemLabelPosition getBasePositiveItemLabelPosition();  
Returns the base positive item label position. The default value is ItemLabelPosition(ItemLabelAnchor.OUTSIDE12,  
TextAnchor.BOTTOM_CENTER).
```

```
→ public void setBasePositiveItemLabelPosition(ItemLabelPosition position);
Equivalent to setBasePositiveItemLabelPosition(position, true)—see the next method.

→ public void setBasePositiveItemLabelPosition(ItemLabelPosition position, boolean notify);
Sets the base positive item label position and, if requested, sends a RendererChangeEvent to all registered listeners.
```

### 35.2.19 Negative Item Label Position

The negative item label position attributes (see table 35.12) control the position of item labels for those items that have a data value that is negative.

Attribute:	Description:
<i>negativeItemLabelPosition</i>	The <i>negativeItemLabelPosition</i> override flag ( <code>null</code> permitted). <i>Deprecated as of 1.0.6</i> .
<i>seriesNegativeItemLabelPositionList</i>	A list of positions that apply to individual series (only referenced if <i>negativeItemLabelPosition</i> is <code>null</code> ).
<i>baseNegativeItemLabelPosition</i>	The base position for all series, the default value is <code>ItemLabelPosition(ItemLabelAnchor.OUTSIDE6, TextAnchor.TOP_CENTER)</code>

Table 35.12: Negative item label position attributes

To determine the label position for an item with a negative value, JFreeChart calls the following method:

```
→ public ItemLabelPosition getNegativeItemLabelPosition(int row, int column);
Returns the position for the specified item. The return value is derived from the override, per series and base level settings.
```

#### Override Negative Item Label Position

The override negative item label position attribute provides a mechanism to override all other settings (this is rarely required):

```
→ public ItemLabelPosition getNegativeItemLabelPosition(); [Deprecated, 1.0.6]
Returns the negative item label position override. The default value is null.

→ public void setNegativeItemLabelPosition(ItemLabelPosition position); [Deprecated, 1.0.6]
Equivalent to setNegativeItemLabelPosition(position, true)—see the next method.

→ public void setNegativeItemLabelPosition(ItemLabelPosition position, boolean notify); [Deprecated, 1.0.6]
Sets the negative item label position for the specified series and, if requested, sends a RendererChangeEvent to all registered listeners.
```

#### Per Series Negative Item Label Position

The per series negative item label positions apply when no override is set:

```
→ public ItemLabelPosition getSeriesNegativeItemLabelPosition(int series);
Returns the negative item label position for the specified series (this may be null).

→ public void setSeriesNegativeItemLabelPosition(int series, ItemLabelPosition position);
Equivalent to setSeriesNegativeItemLabelPosition(series, position, true)—see the next method.

→ public void setSeriesNegativeItemLabelPosition(int series, ItemLabelPosition position, boolean notify);
Sets the negative item label position for the specified series and, if requested, sends a RendererChangeEvent to all registered listeners.
```

### Base Negative Item Label Position

The base setting for the negative item label position attribute is used when no per series or override setting is specified:

```
→ public ItemLabelPosition getBaseNegativeItemLabelPosition();
Returns the base negative item label position. The default value is
ItemLabelPosition(ItemLabelAnchor.OUTSIDE6, TextAnchor.TOP_CENTER).

→ public void setBaseNegativeItemLabelPosition(ItemLabelPosition position);
Equivalent to setBaseNegativeItemLabelPosition(position, true)—see the next method.

→ public void setBaseNegativeItemLabelPosition(ItemLabelPosition position, boolean notify);
Sets the base negative item label position and, if requested, sends a RendererChangeEvent to all
registered listeners.
```

### 35.2.20 Item Label Anchor Offset

The item label anchor offset allows some control over the position of item labels, by controlling how far the anchor point is shifted from its natural position:

```
→ public double getItemLabelAnchorOffset();
Returns the offset (in Java2D units). The default value is 2.0.

→ public void setItemLabelAnchorOffset(double offset);
Sets the item label anchor offset and sends a RendererChangeEvent to all registered listeners.
```

The following utility method calculates the item label anchor point relative to the given (x, y) location:

```
→ protected Point2D calculateLabelAnchorPoint(ItemLabelAnchor anchor,
double x, double y, PlotOrientation orientation);
Calculates the item label anchor point relative to the specified data point. Some renderers
override this method if there is a more natural way to calculate the anchor point (for instance,
the BarRenderer can calculate the anchor points relative to the bar rectangle).
```

### 35.2.21 Entity Generation

Support for tooltips, mouse events, and URLs in HTML image maps relies on the generation of a `ChartEntity` for each item in a series. In some situations, it can be useful to generate entities for a subset of the series in a dataset only. All renderers inherit a set of flags that make this possible.

Attribute:	Description:
<code>createEntities</code>	The <code>createEntities</code> override flag ( <code>null</code> permitted). <i>Deprecated as of 1.0.6.</i>
<code>seriesCreateEntitiesList</code>	A list of flags that apply to individual series (only referenced if <code>createEntities</code> is <code>null</code> ).
<code>baseCreateEntities</code>	The default flag for all series.

Table 35.13: Attributes for the `AbstractRenderer` class

To determine whether or not an entity should be created for an item in a chart, JFreeChart calls the following method:

```
→ public boolean getItemCreateEntity(int series, int item);
Returns true if an entity should be created for the specified item, and false otherwise. The
result is determined by looking at the override, per series and base level settings for this
attribute.
```

### Override Create Entities Flag

The override create entities flag can be used to override the per series and base level flags. It is rarely needed, and is typically left at its default value of `null`:

- ▶ `public Boolean getCreateEntities(); [Deprecated, 1.0.6]`  
Returns the override flag. This is `null` by default.
- ▶ `public void setCreateEntities(Boolean create); [Deprecated, 1.0.6]`  
Equivalent to `setCreateEntities(create, true)`—see the next method.
- ▶ `public void setCreateEntities(Boolean create, boolean notify); [Deprecated, 1.0.6]`  
Sets the override flag (`null` is permitted) and, if requested, sends a `RendererChangeEvent` to all registered listeners.

### Per Series Create Entities Flag

The per series flags apply when no override is defined. If the per series flag is `null`, then the base flag will be used.

- ▶ `public Boolean getSeriesCreateEntities(int series);`  
Returns the create entities flag for the specified series (possible `null`).
- ▶ `public void setSeriesCreateEntities(int series, Boolean create);`  
Equivalent to `setSeriesCreateEntities(series, create, true)`—see the next method.
- ▶ `public void setSeriesCreateEntities(int series, Boolean create, boolean notify);`  
Sets the create series flag for the specified series (`null` is permitted) and, if requested, sends a `RendererChangeEvent` to all registered listeners.

### Base Create Entities Flag

The base create entities flag is used when no other setting is defined.

- ▶ `public boolean getBaseCreateEntities();`  
Returns the base create entities flag. The default value is `true`.
- ▶ `public void setBaseCreateEntities(boolean create);`  
Equivalent to `setBaseCreateEntities(true)`—see the next method.
- ▶ `public void setBaseCreateEntities(boolean create, boolean notify);`  
Sets the base create entities flag and, if requested, sends a `RendererChangeEvent` to all registered listeners.

## 35.2.22 Equals, Cloning and Serialization

The `equals()` method is overridden:

- ▶ `public boolean equals(Object obj);`  
Returns `true` if this renderer is equal to `obj`, and `false` otherwise. An object is considered “equal” to this renderer if:
  - it is not `null`;
  - it is an instance of `AbstractRenderer`;
  - it has the same attribute settings as this renderer;

Registered listeners are not included in the equality test.

By design, all renderers should be `Cloneable` and `Serializable`. Some Java classes (particularly those that implement the Java2D `Shape` and `Paint` interfaces) do not provide built-in support for cloning and serialization. Where possible, special code has been written to handle these cases.

### 35.2.23 Notes

Some points to note:

- subclasses include `AbstractCategoryItemRenderer` and `AbstractXYItemRenderer`.

## 35.3 AreaRendererEndType

### 35.3.1 Overview

This class defines the tokens that can be used to specify the representation of the ends of an area chart. There are three tokens defined, as listed in table 35.14.

Token:	Description:
<code>AreaRendererEndType.TAPER</code>	Taper down to zero.
<code>AreaRendererEndType.TRUNCATE</code>	Truncates at the first and last values.
<code>AreaRendererEndType.LEVEL</code>	Fill to the edges of the chart level with the first and last data values.

Table 35.14: `AreaRendererEndType` tokens

### 35.3.2 Usage

The `AreaRenderer` class has a method named `setEndType()` that accepts the tokens defined by this class.

## 35.4 DefaultPolarItemRenderer

### 35.4.1 Overview

A default renderer for use by the `PolarPlot` class. This class extends `AbstractRenderer` and implements the `PolarItemRenderer` interface.

### 35.4.2 Constructor

To create a new renderer:

```
➔ public DefaultPolarItemRenderer();
Creates a new renderer instance.
```

### 35.4.3 General Attributes

Most attributes are inherited from the `AbstractRenderer` class.

To control whether or not each series is drawn as a “filled” polygon:

```
➔ public boolean isSeriesFilled(int series);
Returns true if the area “inside” the series should be filled, and false otherwise.

➔ public void setSeriesFilled(int series, boolean filled);
Sets the flag that controls whether or not the specified series is “filled”. By default, the setting
is false.
```

### 35.4.4 Other Methods

The remaining methods are called by JFreeChart—you won’t normally call these methods directly. To find the plot that the renderer is assigned to:

- ▶ `public PolarPlot getPlot();`  
Returns the plot that the renderer is assigned to.
- ▶ `public void setPlot(PolarPlot plot);`  
Sets the plot that the renderer is assigned to. Typically the plot will set this when the renderer is added to the plot.
- ▶ `public DrawingSupplier getDrawingSupplier();`  
A convenience method that returns the drawing supplier from the plot that the renderer is assigned to.
- ▶ `public LegendItem getLegendItem(int series);`  
Returns a legend item for the specified series.
- ▶ `public void drawAngularGridLines(Graphics2D g2, PolarPlot plot, List ticks, Rectangle2D dataArea);`  
Draws the gridlines representing the angles around the plot.
- ▶ `public void drawRadialGridLines(Graphics2D g2, PolarPlot plot, ValueAxis radialAxis, List ticks, Rectangle2D dataArea);`  
Draws the circular gridlines showing the units along the axis.
- ▶ `public void drawSeries(Graphics2D g2, Rectangle2D dataArea, PlotRenderingInfo info, PolarPlot plot, XYDataset dataset, int seriesIndex);`  
Draws a series within the specified dataArea.

### 35.4.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- ▶ `public boolean equals(Object obj);`  
Tests this renderer for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### See Also

[PolarPlot](#).

## 35.5 GrayPaintScale

### 35.5.1 Overview

A `PaintScale` implementation that returns shades of gray to represent the values in a range.

*This class was first introduced in JFreeChart version 1.0.4.*

### 35.5.2 Constructors

To create a new instance:

- ▶ `public GrayPaintScale(); [1.0.4]`  
Equivalent to `GrayPaintScale(0.0, 1.0)`—see the next constructor.
- ▶ `public GrayPaintScale(double lowerBound, double upperBound); [1.0.4]`  
Creates a new scale covering the specified value range.

### 35.5.3 Methods

The following methods are defined:

- `public double getLowerBound(); [1.0.4]`  
Returns the lower bound for the value range.
- `public double getUpperBound(); [1.0.4]`  
Returns the upper bound for the value range.
- `public Paint getPaint(double value); [1.0.4]`  
Returns a shade of gray for the specified value. A value at the lower bound will return black, a value at the upper bound will return white.

### 35.5.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj); [1.0.4]`  
Tests this paint scale for equality with an arbitrary object.

### 35.5.5 Notes

This class can be used with the `XYBlockRenderer` class.

#### See Also

[PaintScale](#), [PaintScaleLegend](#).

## 35.6 LookupPaintScale

### 35.6.1 Overview

A `PaintScale` implementation that uses a lookup table to convert values to corresponding `Paint` instances. This class is intended for use by the `XYBlockRenderer` class. **This class was first introduced in JFreeChart version 1.0.4.**

### 35.6.2 Constructors

To create a new instance:

- `public LookupPaintScale(); [1.0.4]`  
Equivalent to `LookupPaintScale(0.0, 1.0, Color.lightGray)`—see the next constructor.
- `public LookupPaintScale(double lowerBound, double upperBound, Paint defaultPaint); [1.0.4]`  
Creates a new scale covering the specified range and with the given default paint. If `defaultPaint` is `null`, this method throws an `IllegalArgumentException`.

### 35.6.3 Methods

The following methods are defined:

- `public Paint getDefaultPaint(); [1.0.4]`  
Returns the default paint, as specified in the constructor. This is never `null`.
- `public double getLowerBound(); [1.0.4]`  
Returns the lower bound for the value range.
- `public double getUpperBound(); [1.0.4]`  
Returns the upper bound for the value range.

To fetch a `Paint` instance from the lookup table:

► `public Paint getPaint(double value); [1.0.4]`

Returns the color for the specified value. For any `value` outside the bounds specified in the constructor, this method will return `getDefaultValue()`.

To add a new entry to the lookup table:

► `public void add(Number n, Paint p); [1.0.4]`

Adds a new entry to the lookup table (or replaces an existing entry if `n` matches an existing value). The entries are stored in ascending order by value—a look up for a given value will return `p` if the value is greater than or equal to `n`, but less than the value for the next entry in the lookup table).

### 35.6.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj); [1.0.4]`

Tests this paint scale for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 35.6.5 Notes

Some points to note:

- this class is designed for use with the `XYBlockRenderer` class;
- a couple of demos (`XYBlockRendererDemo1-2.java`) are included in the JFreeChart demo collection.

#### See Also

[PaintScale](#), [PaintScaleLegend](#).

## 35.7 NotOutlierException

### 35.7.1 Overview

An exception that is, in fact, never used in JFreeChart.

## 35.8 Outlier

### 35.8.1 Overview

Represents an outlier in a box-and-whisker plot. Instances of this class are created on a temporary basis during chart drawing.

### 35.8.2 Constructor

To create a new instance:

► `public Outlier(double xCoord, double yCoord, double radius);`

Creates a new outlier at the specified coordinates (in Java2D space).

### 35.8.3 Methods

The following methods are defined:

- `public Point2D getPoint();`  
Returns the location of the outlier, in Java2D space.
- `public void setPoint(Point2D point);`  
Sets the location of the outlier, in Java2D space.
- `public double getX();`  
Returns the x-coordinate for the outlier, in Java2D space.
- `public double getY();`  
Returns the y-coordinate for the outlier, in Java2D space.
- `public double getRadius();`  
Returns the radius of the outlier, in Java2D units.
- `public void setRadius(double radius);`  
Sets the radius of the outlier, in Java2D units.
- `public int compareTo(Object o);`  
Compares this `Outlier` to an arbitrary object.
- `public boolean overlaps(Outlier other);`  
Tests if this outlier overlaps with the specified outlier.
- `public String toString();`  
Returns a string representation of this outlier, useful for debugging.

#### See Also

[BoxAndWhiskerRenderer](#), [XYBoxAndWhiskerRenderer](#).

## 35.9 OutlierList

### 35.9.1 Overview

Represents a collection of outliers for a single item in a box-and-whisker plot.

#### See Also

[Outlier](#), [OutlierListCollection](#).

## 35.10 OutlierListCollection

### 35.10.1 Overview

Represents a collection of outlier lists for a box-and-whisker plot.

#### See Also

[OutlierList](#).

## 35.11 PaintScale

### 35.11.1 Overview

An interface used by the `XYBlockRenderer` class to convert values (within a certain range) to `Paint` instances. Current implementations include:

- `GrayPaintScale`;
- `LookupPaintScale`;

This interface was first introduced in JFreeChart version 1.0.4.

### 35.11.2 Interface Methods

This interface defines the following methods:

- `public double getLowerBound(); [1.0.4]`  
Returns the lower bound of the value range.
- `public double getUpperBound(); [1.0.4]`  
Returns the upper bound of the value range.
- `public Paint getPaint(double value); [1.0.4]`  
Returns the color associated with the specified value. If `value` is outside the range `getLowerBound()` to `getUpperBound()`, the behaviour is implementation dependent—check the documentation for the implementing class for details.

#### See Also

[PaintScaleLegend](#).

## 35.12 PolarItemRenderer

### 35.12.1 Overview

A renderer that is used by the `PolarPlot` class. The `DefaultPolarItemRenderer` class provides an implementation of this interface.

### 35.12.2 Change Listeners

You can register any number of `RendererChangeListener` objects with the renderer and they will receive notification of any changes to the renderer:

- `public void addChangeListener(RendererChangeListener listener);`  
Registers a listener with the renderer.
- `public void removeChangeListener(RendererChangeListener listener);`  
Deregisters a listener so that it no longer receives change notifications from the renderer.

It is not common that you need to do this yourself, but the mechanism is used by the `PolarPlot` class to monitor changes to the renderer (in order to trigger automatic chart updates).

### 35.12.3 Methods

To create a legend item for a series (this method is called by the plot):

- `public LegendItem getLegendItem(int series);`  
Creates a legend item for the specified series.

To draw the representation of a series (this method is called by the plot):

- `public void drawSeries(Graphics2D g2, Rectangle2D dataArea, PlotRenderingInfo info, PolarPlot plot, XYDataset dataset, int seriesIndex);`  
Draws a series within the specified `dataArea`.

To draw the angle grid lines (this method is called by the plot):

- `public void drawAngularGridLines(Graphics2D g2, PolarPlot plot, List ticks, Rectangle2D dataArea);`  
Draws the angle gridlines for the plot.
- `public void drawRadialGridLines(Graphics2D g2, PolarPlot plot, ValueAxis radialAxis, List ticks, Rectangle2D dataArea);`  
Draws the radius (circular) gridlines for the plot.
- `public PolarPlot getPlot();`  
Returns the plot that the renderer is assigned to (or `null`).

```
↳ public void setPlot(PolarPlot plot);
```

Sets the plot that the renderer is assigned to. Typically the plot will set this itself when the renderer is added to the plot.

## 35.13 RendererState

### 35.13.1 Overview

A base class for maintaining state information that is initialised at the start of the chart drawing process, and passed to each invocation of the renderer's `drawItem()` method. Subclasses include:

- [CategoryItemRendererState](#);
- [XYItemRendererState](#).

### 35.13.2 Constructors

To create a new instance:

```
↳ public RendererState(PlotRenderingInfo info);
```

Creates a new instance that maintains a reference to the plot rendering info (which may be `null`).

### 35.13.3 Methods

This class defines two methods:

```
↳ public PlotRenderingInfo getInfo();
```

Returns the object that collects plot rendering information for the current chart drawing. If this is `null`, no information is recorded.

```
↳ public EntityCollection getEntityCollection();
```

Returns the object that collects chart entity information for the current rendering. This may be `null`, in which case no entity information is recorded. Chart entities are used to support tooltips, drill-down charts and HTML image maps.

### 35.13.4 Equals, Cloning and Serialization

As this class is intended to represent temporary state information, it is neither cloneable nor serializable, and it does not override the `equals()` method.

#### See Also

[CategoryItemRendererState](#), [XYItemRendererState](#).

## 35.14 WaferMapRenderer

### 35.14.1 Overview

A renderer used by the `WaferMapPlot` class.

# Chapter 36

## Package: `org.jfree.chart.renderer.category`

### 36.1 Overview

This package contains interfaces and classes that are used to implement renderers for the `CategoryPlot` class. Renderers offer a lot of scope for changing the appearance of your charts, either by changing the attributes of an existing renderer, or by implementing a completely new renderer.

### 36.2 AbstractCategoryItemRenderer

#### 36.2.1 Overview

A base class (extending `AbstractRenderer`) that can be used to implement a new `CategoryItemRenderer`. Subclasses include:

- `AreaRenderer`;
- `BarRenderer`;
- `LevelRenderer`;
- `LineAndShapeRenderer`.

To create a `CategoryItemRenderer` implementation requires a lot of methods to be written—by extending this base class, you can save yourself a lot of effort.

#### 36.2.2 Constructors

The default constructor creates a renderer with no tooltip generator and no URL generator. The constructor is `protected`.

#### 36.2.3 Attributes

The attributes maintained by this class are listed in Table 36.1.

#### 36.2.4 The Pass Count

Most renderers draw data items in a single pass through the dataset. However, some renderers might require two or more passes through the dataset, in order to draw items in a layered fashion. The plot will call the following method to determine how many passes the renderer requires:

<b>Attribute:</b>	<b>Description:</b>
<i>plot</i>	The <code>CategoryPlot</code> that the renderer is assigned to.
<i>toolTipGenerator</i>	The <code>CategoryToolTipGenerator</code> that generates tool tips for ALL series (can be null).
<i>toolTipGeneratorList</i>	A list of <code>CategoryToolTipGenerator</code> objects used to create tool tips for individual series.
<i>baseToolTipGenerator</i>	The base <code>CategoryToolTipGenerator</code> used to create tool tips when there is no other generator available.
<i>itemLabelGenerator</i>	The <code>CategoryItemLabelGenerator</code> that generates item labels for ALL series (can be null).
<i>itemLabelGeneratorList</i>	A list of <code>CategoryItemLabelGenerator</code> objects used to create item labels for individual series. If null, the <code>baseLabelGenerator</code> is used instead.
<i>baseItemLabelGenerator</i>	The base <code>CategoryItemLabelGenerator</code> used to create item labels when no other generator is available.
<i>itemURLGenerator</i>	The <code>CategoryURLGenerator</code> that applies to ALL series.
<i>itemURLGeneratorList</i>	A list of <code>CategoryURLGenerator</code> objects that apply to individual series. If null, the <code>baseItemURLGenerator</code> is used instead.
<i>baseItemURLGenerator</i>	The base <code>CategoryURLGenerator</code> , used when no other generator is available.
<i>legendItemLabelGenerator</i>	The generator for the series name in the legend.
<i>legendItemToolTipGenerator</i>	The generator for the tool tips for the legend items.
<i>legendItemURLGenerator</i>	The generator for the URL for hyperlinks for the legend items(in HTML image maps)

Table 36.1: Attributes for the `AbstractCategoryItemRenderer` class

```
➔ public int getPassCount();
```

Returns 1, which is the default number of passes required by a renderer. If a subclass requires more than one pass through the dataset, it must override this method.

Standard renderers that require multiple passes through the dataset include:

- `LineAndShapeRenderer`;
- `StackedAreaRenderer`;
- `StackedBarRenderer`;
- `StackedBarRenderer3D`.

### 36.2.5 Methods

The following method is called once every time the chart is drawn:

```
➔ public CategoryItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea,
CategoryPlot plot, Integer index, PlotRenderingInfo info);
```

Performs any initialisation required by the renderer. The default implementation simply stores a local reference to the `info` object (which may be null).

The number of rows and columns in the dataset (a `CategoryDataset`) is cached by the renderer in the `initialise()` method.

To draw the plot background:

```
➔ public void drawBackground(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
```

Draws the plot background. Some renderers will choose to override this method, but for most the default behaviour is OK.

To draw the plot outline:

```
➔ public void drawOutline(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
```

Draws the plot outline. Some renderers will choose to override this method, but for most the default behaviour is OK.

To draw a domain gridline:

```
→ public void drawDomainGridline(Graphics2D g2, CategoryPlot plot,
    Rectangle2D dataArea, double value);
    Draws a domain gridline at the specified value. This method is called by the CategoryPlot class.
```

To draw a range gridline:

```
→ public void drawRangeGridline(Graphics2D g2, CategoryPlot plot,
    ValueAxis axis, Rectangle2D dataArea, double value);
    Draws a range gridline at the specified value. This method is called by the CategoryPlot class.
```

To draw a range marker:

```
→ public void drawRangeMarker(Graphics2D g2, CategoryPlot plot,
    ValueAxis axis, Marker marker, Rectangle2D dataArea);
    Draws a range marker. This method is called by the CategoryPlot class.
```

To get a legend item:

```
→ public LegendItem getLegendItem(int datasetIndex, int series);
    Returns a legend item for the specified series. The datasetIndex is zero for the primary dataset, and 1..N for the secondary datasets.
```

To get the `CategoryItemLabelGenerator` for a data item:

```
→ public CategoryItemLabelGenerator getItemLabelGenerator(int row, int column);
    Returns the item label generator for a specific data item. By default, this method just calls the getSeriesLabelGenerator() method.
```

To get the `CategoryItemLabelGenerator` for a series:

```
→ public CategoryItemLabelGenerator getSeriesItemLabelGenerator(int series);
    Returns the item label generator for a series. This method returns the labelGenerator if it is set, otherwise it looks up the labelGeneratorList to get a generator specific to the series. If the series-specific generator is null, the baseLabelGenerator is returned.
```

To get the `CategoryURLGenerator` for a data item:

```
→ public CategoryURLGenerator getItemURLGenerator(int row, int column);
    Returns the item URL generator for a specific data item. By default, this method just calls the getSeriesItemURLGenerator() method.
```

To get the `CategoryURLGenerator` for a series:

```
→ public CategoryURLGenerator getSeriesItemURLGenerator(int series);
    Returns the item URL generator for a series. This method returns the itemURLGenerator if it is set, otherwise it looks up the itemURLGeneratorList to get a generator specific to the series. If the series-specific generator is null, the baseItemURLGenerator is returned.
```

To get the row count:

```
→ public int getRowCount();
    Returns the row count.
```

To get the column count:

```
→ public int getColumnCount();
    Returns the column count.
```

### 36.2.6 Legend Items

All renderers should implement the `LegendItemSource` interface, to allow a legend to fetch the relevant items for display. This requires the provision of the following method:

```
→ public LegendItemCollection getLegendItems();
```

Returns the legend items for this renderer. This implementation returns one item for each series that the renderer is responsible for, taking into account the series visibility flags.

The individual legend items are created with a call to the following method:

```
→ public LegendItem getLegendItem(int datasetIndex, int series);
```

Creates a legend item for the specified series.<sup>1</sup> This method can return `null`, in which case no legend item will be added to the series (in this default implementation, `null` is returned if either `isSeriesVisible(series)` or `isSeriesVisibleInLegend(series)` returns `false`). Subclasses may override this method to provide customised legend items.

A plug-in generator is used to create each legend item label, so that you can customise the label if necessary:

```
→ public CategorySeriesLabelGenerator getLegendItemLabelGenerator();
```

Returns the legend item label generator (never `null`). The default generator simply returns the series key converted to a string using the key's `toString()` method.

```
→ public void setLegendItemLabelGenerator(CategorySeriesLabelGenerator generator);
```

Sets the legend item label generator and sends a `RendererChangeEvent` to all registered listeners. If `generator` is `null`, this method throws an `IllegalArgumentException`.

Each legend item can (optionally) have a tool tip associated with it. If you require tool tips, you need to specify a tool tip generator:

```
→ public CategorySeriesLabelGenerator getLegendItemToolTipGenerator();
```

Returns the tool tip generator for the legend items created by this renderer. The default is `null`.

```
→ public void setLegendItemToolTipGenerator(CategorySeriesLabelGenerator generator);
```

Sets the tool tip generator for the legend items created by this renderer, and sends a `RendererChangeEvent` to all registered listeners. If you set this to `null`, no tool tips will be generated.

Similarly, each legend item can (optionally) have a URL associated with it (these are used only in HTML image maps). If you require URLs, you need to specified a URL generator:

```
→ public CategorySeriesLabelGenerator getLegendItemURLGenerator();
```

Returns the URL generator for the legend items created by this renderer. The default is `null`. The URLs are used to provide hyperlinks in HTML image maps.

```
→ public void setLegendItemURLGenerator(CategorySeriesLabelGenerator generator);
```

Sets the URL generator for the legend items created by this renderer, and sends a `RendererChangeEvent` to all registered listeners. If you set this to `null`, no URLs will be generated.

### 36.2.7 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj);
```

Tests this renderer for equality with an arbitrary object. Note that the test does NOT take into account the plot that the renderer is assigned to.

Instances of this class are cloneable and serializable.

### 36.2.8 Notes

If you are implementing your own renderer, you do not have to use this base class, but it does save you some work.

---

<sup>1</sup>The `dataset` argument is passed in because the renderer itself has no state information to know which dataset it is rendering.

## 36.3 AreaRenderer

### 36.3.1 Overview

A renderer that represents the items in a [CategoryDataset](#) using a polygon that fills the area between the x-axis and the data points—an example is shown in figure 36.1. This renderer is designed for use with the [CategoryPlot](#) class.

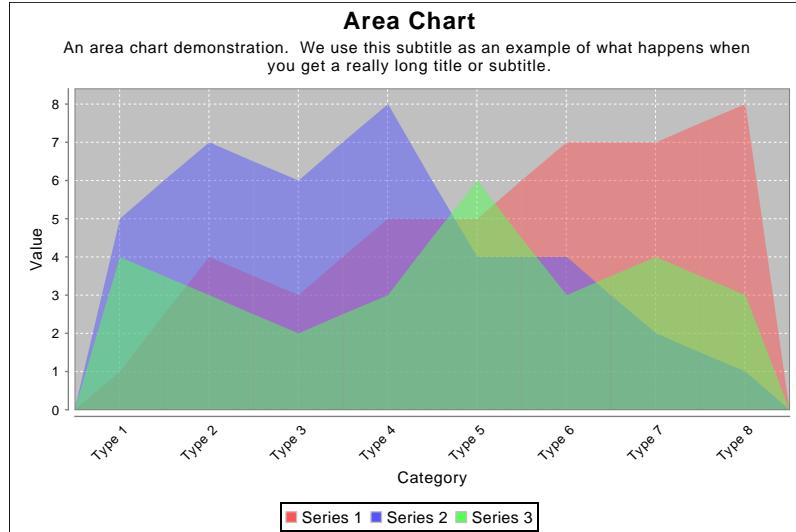


Figure 36.1: An area chart (see [AreaChartDemo1.java](#))

### 36.3.2 Constructor

To create a new renderer:

```
➔ public AreaRenderer();
Creates a new renderer with default attributes.
```

The [ChartFactory.createAreaChart\(\)](#) method can also be used to create a chart that uses this type of renderer.

### 36.3.3 General Attributes

To control how the end points of the area chart are represented:

```
➔ public AreaRendererEndType getEndType();
Returns a token indicating how the ends of the area drawn by this renderer are handled (never null). The default value is AreaRendererEndType.TAPER.
```

```
➔ public void setEndType(AreaRendererEndType type);
Sets the token that controls how the end points are drawn on the area chart and sends a RendererChangeEvent to all registered listeners. If type is null, this method throws an IllegalArgumentException.
```

Other attributes are inherited from [AbstractCategoryItemRenderer](#).

### 36.3.4 Other Methods

The following methods are typically called by JFreeChart—you shouldn't need to call them directly:

```
► public LegendItem getLegendItem(int datasetIndex, int series);
Overridden to use a custom graphic in the legend item.

► public void drawItem(Graphics2D g2, CategoryItemRendererState state,
Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis,
CategoryDataset dataset, int row, int column, int pass);
Draws one item from the dataset.
```

### 36.3.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 36.3.6 Notes

Some points to note:

- the `createAreaChart()` method in the `ChartFactory` class will create a default chart that uses this renderer;
- this class extends `AbstractCategoryItemRenderer`;
- a demo (`AreaChartDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[AbstractCategoryItemRenderer](#), [XYAreaRenderer](#).

## 36.4 BarRenderer

### 36.4.1 Overview

This renderer is used in conjunction with a `CategoryPlot` to create bar charts from data in a `CategoryDataset`. The renderer will handle plots with a vertical orientation (see figure 36.2) or a horizontal orientation (see figure 36.3).

The renderer will recognise the use of `GradientPaint` instances for series colors and use a special transformer to apply these to bar regions.

This class extends the `AbstractCategoryItemRenderer` base class.

### 36.4.2 Constructor

The constructor creates a new renderer with default settings:

```
► public BarRenderer();
Creates a new renderer with a default settings. By default, the renderer will draw outlines
around the bars, will have an item margin of 20% (this controls the amount of space allocated to
the gaps between bars within a single category), and will use a StandardGradientPaintTransformer
when a series color is an instance of GradientPaint.
```

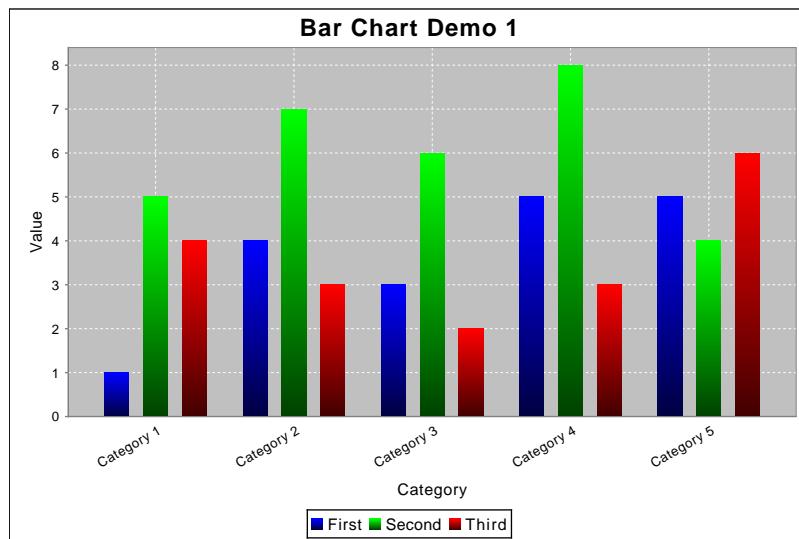


Figure 36.2: A vertical bar chart (see `BarChartDemo1.java`)

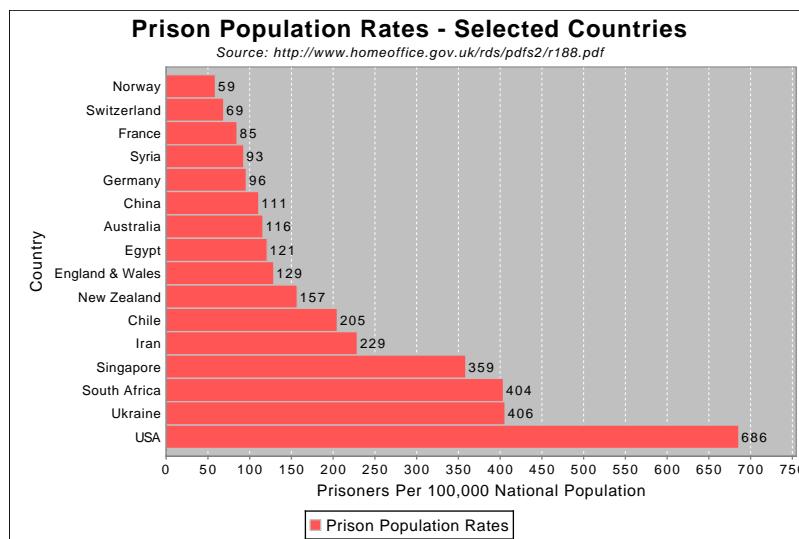


Figure 36.3: A horizontal bar chart (see `BarChartDemo5.java`)

### 36.4.3 The Base Value

By default, the renderer draws a bar between zero (the *base value*) and the data value of the item to be displayed. Some specialised bar charged require a non-zero base value—you can use the following methods to access/modify the base value:

► `public double getBase();`

Returns the base value for the bars. The default value is 0.0.

► `public void setBase(double base);`

Sets the base value for the bars and sends a `RendererChangeEvent` to all registered listeners.

The `includeBaseInRange` flag controls whether or not JFreeChart will include the base value when calculating the bounds for the chart's range axis:

► public boolean getIncludeBaseInRange(); [1.0.1]

Returns `true` if the base value should be included in the auto range calculation for the chart's range axis, and `false` otherwise. The default value is `true`.

► public void setIncludeBaseInRange(boolean include); [1.0.1]

Sets the flag that controls whether or not the base value is included in the auto range calculation for the range axis, and sends a `RendererChangeEvent` to all registered listeners.

#### 36.4.4 Controlling the Width of Bars

The renderer automatically calculates the width of the bars to fit the available space for the plot, so you cannot directly control how wide the bars are. However, the bar width is a function of the following attributes that you can control:

- the `lowerMargin`, `upperMargin` and `categoryMargin` attributes, all defined by the `CategoryAxis` (see figure 24.8.1 for more information about the purpose of these attributes);
- the `itemMargin` attribute belonging to the renderer (see below).

The `itemMargin` attribute controls the amount of space between bars *within a category*:

► public double getItemMargin();

Returns the item margin as a percentage of the overall length of the category axis (the default is 0.20, or twenty percent). This controls the amount of space that is allocated to the gaps between bars within the same category.

► public void setItemMargin(double percent);

Sets the item margin and sends a `RendererChangeEvent` to all registered listeners.

The dynamic bar width calculation can result in very wide bars if you have only a few data values in a chart. If you would like to specify a “cap” for the bar width, use the `maximumBarWidth` attribute:

► public double getMaximumBarWidth();

Returns the maximum bar width allowed, as a percentage of the length of the category axis. The default is 1.00 (100 percent) which means that the bar widths are never capped.

► public void setMaximumBarWidth(double percent);

Sets the maximum bar width as a percentage of the axis length and sends a `RendererChangeEvent` to all registered listeners. For example, setting this to 0.05 will ensure that the bars never exceed five percent of the length of the axis. This can improve the appearance of charts where there is a possibility that only one or two bars will be displayed.

#### 36.4.5 Bar Outlines

The `drawBarOutline` flag controls whether the bars drawn by the renderer are outlined:

► public boolean isDrawBarOutline();

Returns the flag that controls whether an outline is added to each bar drawn by this renderer. The default value is `false`.

► public void setDrawBarOutline(boolean draw);

Sets the flag that controls whether or not an outline is drawn around each bar and sends a `RendererChangeEvent` to all registered listeners. The `Paint` and `Stroke` used for the bar outline is specified using methods in the superclass.

#### 36.4.6 Gradient Paint Support

You can set the colour for the bars in a series using the `setSeriesPaint()` method inherited from the `AbstractRenderer` class.<sup>2</sup>

---

<sup>2</sup>Note that the `setSeriesFillPaint()` method does NOT change the bar colour—the fill paint is ignored by this renderer.

To provide better support for the use of `GradientPaint` objects to color the bars drawn by this renderer, you can specify a *transformer* that will dynamically adjust the `GradientPaint` to fit each bar:

```
➔ public GradientPaintTransformer getGradientPaintTransformer();
Returns the transformer used for GradientPaint instances. If this is null, any GradientPaint instance will be used in its raw form (i.e. with fixed coordinates), which you typically don't want.

➔ public void setGradientPaintTransformer(GradientPaintTransformer transformer);
Sets the transformer (null is permitted) used to transform GradientPaint instances and sends a RendererChangeEvent to all registered listeners.
```

The `BarChartDemo1.java` application, included in the JFreeChart demo collection, provides an example of the use of this attribute.

### 36.4.7 Item Labels

This renderer supports the display of item labels. For the most part, these are controlled using methods defined in the super class, but there are some settings that are specific to the bar renderer.

Due to the rectangular nature of the bars, the renderer calculates anchor points that are arranged as shown in figure 36.4. Note that the numbers correspond (roughly) to the position of the hours on a clock face.

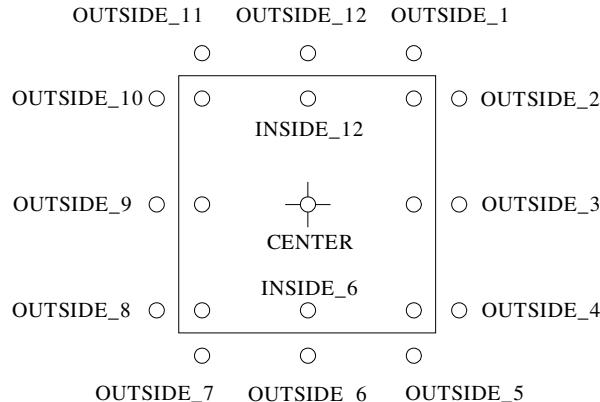


Figure 36.4: Item Label Anchors for Bars

When an item label is displayed inside a bar, the renderer will calculate if the bar is large enough to contain the text. If not, the renderer will check to see if a “fallback” label position has been specified. If there is a fallback position, the label is displayed there, and if there is no fallback position the label is not displayed at all. Two fallback positions can be specified, one for positive values and one for negative values (this covers the standard case where positive value labels that don’t fit within a bar should be displayed above the bar, and negative value labels that don’t fit within a bar should be displayed below the bar).

```
➔ public ItemLabelPosition getPositiveItemLabelPositionFallback();
Returns the fallback position for positive value labels that don't fit within a bar. This can be null, in which case the label won't be displayed at all.
```

```
→ public void setPositiveItemLabelPositionFallback(ItemLabelPosition position);
Sets the fallback position for positive item labels (null is permitted) and sends a RendererChangeEvent to all registered listeners. Set the fallback position to null if you prefer labels to be hidden if they don't fit within the bar.
```

```
→ public ItemLabelPosition getNegativeItemLabelPositionFallback();
Returns the fallback position for negative value labels that don't fit within a bar. This can be null, in which case the label won't be displayed at all.
```

```
→ public void setNegativeItemLabelPositionFallback(ItemLabelPosition position);
Sets the fallback position for negative item labels (null is permitted) and sends a RendererChangeEvent to all registered listeners. Set the fallback position to null if you prefer labels to be hidden if they don't fit within the bar.
```

### 36.4.8 Other Methods

This class implements all the methods in the `CategoryItemRenderer` interface.

```
→ public CategoryItemRendererState initialise(Graphics2D g2,
Rectangle2D dataArea, CategoryPlot plot, int rendererIndex, PlotRenderingInfo info);
This method is called by the plot at the start of every chart drawing run (you shouldn't need to call this method yourself). It initialises the renderer and creates a state object that will be passed to each invocation of the drawItem() method for this drawing run only.
```

```
→ public void drawItem(Graphics2D g2, CategoryItemRendererState state,
Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis,
ValueAxis rangeAxis, CategoryDataset dataset, int row, int column);
This method is called (by the plot) once for each item in the dataset. The renderer state is the same object that was created in the initialise() method.
```

For very small data values (relative to the axis range), you can have bars with a length of less than 1 pixel (on-screen)—when the value gets too small, the bar will disappear. If you want to ensure that a line is always drawn so that the small bar is visible, you can specify a minimum bar length with this method:

```
→ public void setMinimumBarLength(double min);
Sets the minimum length that will be used for a bar, specified in Java 2D units. You can set this to 1.0, for example, to ensure that very short bars do not disappear.
```

### 36.4.9 Internal Methods

The following methods are used internally by the renderer:

```
→ protected void calculateBarWidth(CategoryPlot plot, Rectangle2D dataArea,
int rendererIndex, CategoryItemRendererState state)
This method is called during the initialisation of each drawing run to calculate the width of each bar. The calculated value is stored in the renderer state so it doesn't need to be recalculated for every bar in the chart.
```

### 36.4.10 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object. This method returns true if and only if:


- obj is not null;
- obj is an instance of BarRenderer;
- obj has the same attributes as this renderer;

```

Instances of this class are `Cloneable` and `Serializable`.

### 36.4.11 Notes

Some points to note:

- this renderer inherits `seriesPaint` and `seriesFillPaint` attributes from `AbstractRenderer`. The bar colour is determined by the former—the `seriesFillPaint` attribute is never used by this renderer;
- the `ChartFactory` class uses this renderer when it constructs bar charts via the `createBarChart()` method;
- a range of demos (for example, `BarChartDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[BarRenderer3D](#), [StackedBarRenderer](#), [StackedBarRenderer3D](#).

## 36.5 BarRenderer3D

### 36.5.1 Overview

A renderer that draws items from a `CategoryDataset` using bars with a 3D effect—see figure 36.5.

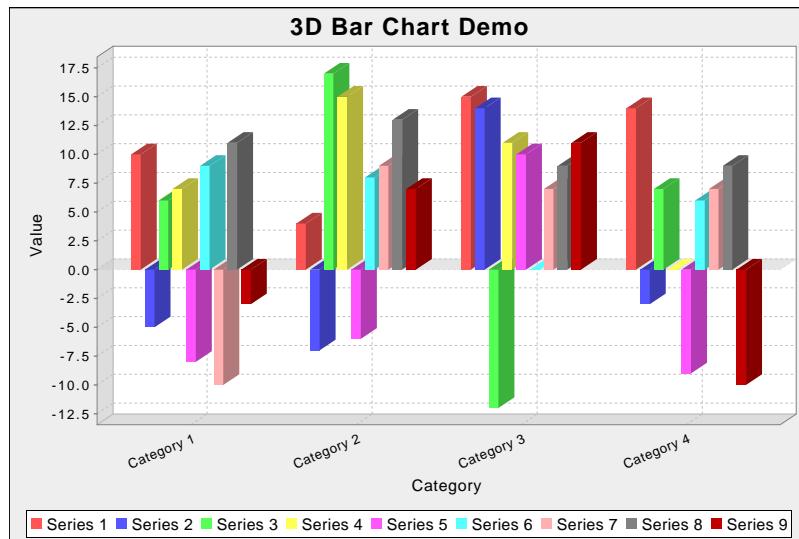


Figure 36.5: A bar chart with 3D effect (see `BarChart3DDemo1.java`)

This renderer is a subclass of `BarRenderer` and is designed for use with the `CategoryPlot` class.

### 36.5.2 Constructors

There are two constructors:

► `public BarRenderer3D();`

Equivalent to `BarRenderer(12.0, 8.0)`—see below.

► `public BarRenderer3D(double xOffset, double yOffset);`

Creates a new renderer with the specified offsets for the 3D effect (specified in Java2D units).

### 36.5.3 Attributes

To access the 3D offset (which is defined in the constructor):

```
↳ public double getXOffset();
```

Returns the x-offset for the 3D effect, in Java2D units. The default value is 12.0. This attribute can only be set via the class constructor.

```
↳ public double getYOffset();
```

Returns the y-offset for the 3D effect, in Java2D units. The default value is 8.0. This attribute can only be set via the class constructor.

The “wall paint” is the color used to highlight the inner wall of the data area in the plot:

```
↳ public Paint getWallPaint();
```

Returns the paint used to draw the inside walls of the plot area, highlighting the 3D effect. The default value is `Color(0xDD, 0xDD, 0xDD)`. This method never returns `null`.

```
↳ public void setWallPaint(Paint paint);
```

Sets the paint used to draw the inside walls of the plot area, highlighting the 3D effect, and sends a `RendererChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

### 36.5.4 Other Methods

Several methods are overridden to add support for the 3D effect. JFreeChart calls these methods, you won’t normally call them directly:

```
↳ public CategoryItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea,
CategoryPlot plot, int rendererIndex, PlotRenderingInfo info);
```

Overridden for the purpose of adjusting the `dataArea`, some of which is taken away by the 3D effect.

```
↳ public void drawItem(Graphics2D g2, CategoryItemRendererState state, Rectangle2D dataArea,
CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis, CategoryDataset dataset,
int row, int column, int pass);
```

Draws a single bar with 3D effect.

```
↳ public void drawBackground(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
```

Draws the background area, taking into account the 3D effect.

```
↳ public void drawOutline(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
```

Draws the plot outline, taking into account the 3D effect.

```
↳ public void drawDomainGridline(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
double value);
```

Draws a domain gridline, taking into account the 3D effect.

```
↳ public void drawRangeGridline(Graphics2D g2, CategoryPlot plot, ValueAxis axis,
Rectangle2D dataArea, double value);
```

Draws a range gridline, taking into account the 3D effect.

```
↳ public void drawRangeMarker(Graphics2D g2, CategoryPlot plot, ValueAxis axis,
Marker marker, Rectangle2D dataArea);
```

Draws a range marker, taking into account the 3D effect.

### 36.5.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
↳ public boolean equals(Object obj);
```

Tests this renderer for equality with an arbitrary object, returning `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `BarRenderer3D`;
- `obj` has the same x- and y-offset and wall paint settings;
- `super.equals(obj)` returns `true`.

Instances of this class are `Cloneable` (`PublicCloneable`) and `Serializable`.

### 36.5.6 Notes

Some points to note:

- this class is a subclass of [BarRenderer](#) and implements the [CategoryItemRenderer](#) interface.
- the 3D effect drawn by this renderer is sensitive to the order in which the data items are drawn (see section [33.3.8](#));
- a couple of demos ([BarChart3DDemo1.java](#) and [BarChart3DDemo2.java](#)) are included in the JFree-Chart demo collection.

#### See Also

[BarRenderer](#), [StackedBarRenderer3D](#), [CategoryAxis3D](#).

## 36.6 BoxAndWhiskerRenderer

### 36.6.1 Overview

A renderer that is used to create a box-and-whisker chart using data from a special dataset ([BoxAndWhiskerCategoryDataset](#)). A sample chart is shown in Figure 36.6.

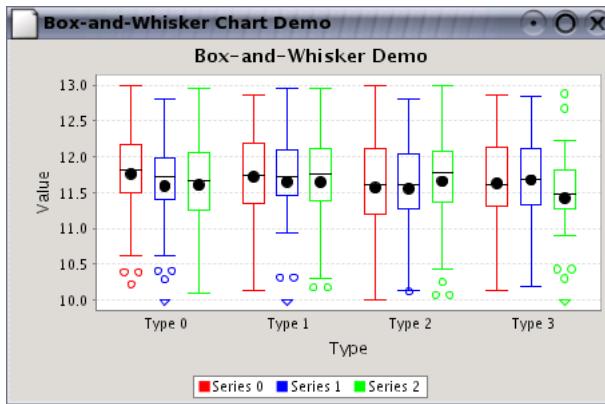


Figure 36.6: A chart generated with a [BoxAndWhiskerRenderer](#)

The chart indicates the following data items:

- the mean value is drawn as a filled circle (with a diameter equal to half the width of the box);
- the median is drawn as a short horizontal line;
- the box highlights the range from quartile 1 to quartile 3;
- the capped line extending from each end of the box indicates the full range of *regular* values;
- hollow circles are used to indicate outlier values;
- triangles at each extreme indicate the presence of *far out* values.

Note that a similar renderer ([XYBoxAndWhiskerRenderer](#)) is available for use with the [XYPlot](#) class.

### 36.6.2 Constructors

To create a new renderer:

```
→ public BoxAndWhiskerRenderer();  
Creates a new renderer.
```

### 36.6.3 Methods

To control the color of the median and mean indicators:

```
► public Paint getArtifactPaint();
Returns the Paint used to draw the median and mean indicators. The default is Color.black.

► public void setArtifactPaint(Paint paint);
Sets the Paint used to draw the median and mean indicators and sends a RendererChangeEvent to all registered listeners. If paint is null, this method throws an IllegalArgumentException.
```

To determine whether or not the boxes are filled:

```
► public boolean getFillBox();
Returns true if the boxes are filled, and false otherwise. The default is true.

► public void setFillBox(boolean flag);
Sets the flag that controls whether or not the boxes are filled, and sends a RendererChangeEvent to all registered listeners.
```

To control the spacing between items within a category:<sup>3</sup>

```
► public double getItemMargin();
Returns the item margin as a percentage of the overall length of the category axis (the default is 0.20, or twenty percent). This controls the amount of space that is allocated to the gaps between items within the same category.

► public void setItemMargin(double margin);
Sets the item margin and sends a RendererChangeEvent to all registered listeners.
```

The method that creates legend items is overridden:

```
► public LegendItem getLegendItem(int datasetIndex, int series);
Returns a legend item for the specified series. This method is overridden to return a box as the legend item shape.
```

### 36.6.4 Notes

Some points to note:

- a demo application (`BoxAndWhiskerDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[XYBoxAndWhiskerRenderer](#).

## 36.7 CategoryItemRenderer

### 36.7.1 Overview

A *category item renderer* is an object that is assigned to a [CategoryPlot](#) and assumes responsibility for drawing the visual representation of individual data items in a dataset. This interface defines the methods that must be provided by all category item renderers—the plot will only use the methods defined in this interface.

A number of different renderers have been developed, allowing different chart types to be generated easily. The following table lists the renderers that have been implemented to date:

---

<sup>3</sup>The spacing between categories is controlled by the [CategoryAxis](#).

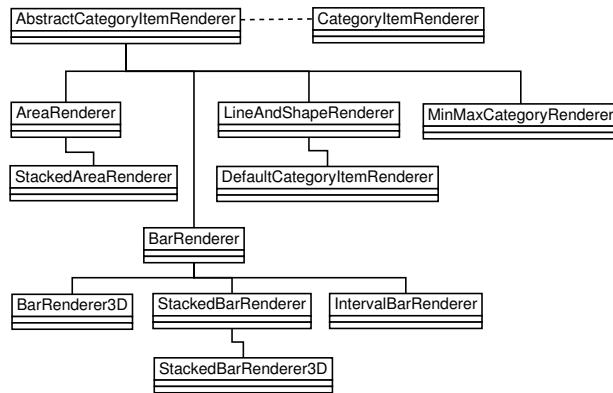


Figure 36.7: Category item renderers

Class:	Description:
<a href="#">AreaRenderer</a>	Used to create area charts.
<a href="#">BarRenderer</a>	Represents data using bars (anchored at zero).
<a href="#">BarRenderer3D</a>	Represents data using bars (anchored at zero) with a 3D effect.
<a href="#">BoxAndWhiskerRenderer</a>	A box-and-whisker plot.
<a href="#">CategoryStepRenderer</a>	Connects data values using a stepped line.
<a href="#">GanttRenderer</a>	For displaying simple Gantt charts.
<a href="#">GroupedStackedBarRenderer</a>	Stacked bar charts with custom groupings.
<a href="#">IntervalBarRenderer</a>	Draws intervals using bars. This renderer can be used to create simple Gantt charts.
<a href="#">LayeredBarRenderer</a>	For layered bar charts.
<a href="#">LevelRenderer</a>	Displays levels (usually overlaid on top of a regular bar chart).
<a href="#">LineAndShapeRenderer</a>	Draws lines and/or shapes to represent data.
<a href="#">LineRenderer3D</a>	A line chart with 3D effect.
<a href="#">MinMaxCategoryRenderer</a>	For plotting min-max ranges.
<a href="#">StackedAreaRenderer</a>	For stacked area charts.
<a href="#">StackedBarRenderer</a>	Used to create a stacked bar charts.
<a href="#">StackedBarRenderer3D</a>	For stacked bar charts with a 3D effect.
<a href="#">StatisticalBarRenderer</a>	For bar charts with an error indicator.
<a href="#">StatisticalLineAndShapeRenderer</a>	For line charts with an error indicator.

The [AbstractCategoryItemRenderer](#) is a useful base class for implementing this interface, if you are developing your own renderer.

### 36.7.2 General Methods

To find the plot that the renderer is currently assigned to:

► public [CategoryPlot](#) getPlot();

Returns the plot that the renderer is currently assigned to.

► public void setPlot([CategoryPlot](#) plot);

Sets the plot that the renderer is currently assigned to. This method is called by the [CategoryPlot](#) class, you shouldn't need to call this yourself.

The following method returns the range of values that will be spanned by the visual representation of the specified dataset, as drawn by this renderer:

► public [Range](#) findRangeBounds([CategoryDataset](#) dataset);

Returns the range of values that will be spanned by the visual representation of the data values in the specified dataset, as drawn by this renderer. This method is used to determine an appropriate default range for the y-axis on a plot.

The interface defines an initialisation method:

```
➔ public CategoryItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea,
CategoryPlot plot, Integer index, PlotRenderingInfo info);
```

This method is called exactly once at the start of every chart redraw. The method returns a state object that the plot will pass to the `drawItem()` method for each data item that the renderer needs to draw. Thus, it gives the renderer a chance to precalculate any information it might require later when rendering individual data items.

The *pass count* refers to the number of passes that the renderer makes through the dataset. The majority of renderers use just a single pass, but some renderers make two or more passes, drawing the data items in several layers:

```
➔ public int getPassCount();
```

Returns the number of passes through the dataset that need to be made by this renderer.

The most important method is the one that actually draws a data item:

```
➔ public void drawItem(...);
```

Draws one item on a category plot. The `CategoryPlot` class will iterate through the data items, passing them to the renderer one at a time.

```
➔ public LegendItem getLegendItem(int datasetIndex, int series);
```

Returns a legend item for the specified series.

### 36.7.3 The Paint and Outline Paint

This interface assumes that the renderer stores values for the paint and outline paint attributes on a per-series basis, with a default value that can be used when no value is specified for a particular series. By convention, the `CategoryPlot` will always call the following method to obtain the paint value for a data item:

```
➔ public Paint getItemPaint(int row, int column);
```

Returns the paint to use for the specified data item. This method should never return `null`.

The per-series paint settings can be controlled via the following methods:

```
➔ public Paint getSeriesPaint(int series);
```

Returns the paint for the specified series (possibly `null`).

```
➔ public void setSeriesPaint(int series, Paint paint);
```

Sets the paint for the specified series (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

The default value (to be used in the case that there is no value specified for a particular series) is controlled via the following methods:

```
➔ public Paint getBasePaint();
```

Returns the default paint, typically used when no per-series paint is defined. This method should never return `null`.

```
➔ public void setBasePaint(Paint paint);
```

Sets the default paint to be used when no per-series paint is defined. If `paint` is `null`, this method should throw an `IllegalArgumentException`. If the new `paint` value is different to the existing value, this method should send a `RendererChangeEvent` to all registered listeners.

Similarly for the outline paint, the `CategoryPlot` will always call the following method to obtain the outline paint for a data item:

```
➔ public Paint getItemOutlinePaint(int row, int column);
```

Returns the outline paint to use for the specified data item. This method should never return `null`.

The per-series outline paint settings can be controlled via the following methods:

```
➔ public Paint getSeriesOutlinePaint(int series);
```

Returns the paint for the specified series (possibly `null`).

```
→ public void setSeriesOutlinePaint(int series, Paint paint);
Sets the paint for the specified series (null is permitted) and sends a RendererChangeEvent to all registered listeners.
```

The default outline paint value (to be used in the case that there is no value specified for a particular series) is controlled via the following methods:

```
→ public Paint getBaseOutlinePaint();
Returns the default outline paint, typically used when no per-series outline paint is defined. This method should never return null.
```

```
→ public void setBaseOutlinePaint(Paint paint);
Sets the default outline paint to be used when no per-series outline paint is defined. If paint is null, this method should throw an IllegalArgumentException. If the new paint value is different to the existing value, this method should send a RendererChangeEvent to all registered listeners.
```

### 36.7.4 The Stroke

The style of lines (if any) drawn by the renderer is controlled by the stroke attribute. A renderer should always call the following method to determine the stroke on a per-item basis:

```
→ public Stroke getItemStroke(int row, int column);
Returns the stroke for the specified data item. This method should never return null.
```

For the convenience of developers using the renderer API, the interface assumes that a renderer stores stroke attributes on a per-series basis (at least), with a default value that is used when no per-series stroke is defined:

```
→ public Stroke getSeriesStroke(int series);
Returns the stroke for the specified series, or null if no stroke is defined.
```

```
→ public void setSeriesStroke(int series, Stroke stroke);
Sets the stroke for the specified series and sends a RendererChangeEvent to all registered listeners.
```

```
→ public Stroke getBaseStroke();
Returns the default stroke (never null). This is used when no other stroke setting is available.
```

```
→ public void setBaseStroke(Stroke stroke);
Sets the default stroke, to be used when no other stroke is available. If stroke is null, this method throws an IllegalArgumentException.
```

### 36.7.5 The Outline Stroke

The outline stroke controls the pen style used to draw the outline of any shapes drawn by the renderer (for example, the bars in a bar chart, or the circles/squares/triangles drawn at each data point by a `LineAndShapeRenderer`). A renderer should always call the following method to determine the outline stroke on a per-item basis:

```
→ public Stroke getItemOutlineStroke(int row, int column);
Returns the outline stroke for the specified item. This method should never return null.
```

For the convenience of developers using the renderer API, the interface assumes that a renderer stores outline stroke attributes on a per-series basis (at least), with a default value that is used when no per-series outline stroke is defined:

```
→ public Stroke getSeriesOutlineStroke(int series);
Returns the outline stroke for the specified series, or null if no outline stroke is defined.
```

```
→ public void setSeriesOutlineStroke(int series, Stroke stroke);
Sets the outline stroke for the specified series and sends a RendererChangeEvent to all registered listeners.
```

```
→ public Stroke getBaseOutlineStroke();
Returns the default outline stroke (never null). This is used when no other setting is available.
```

```
→ public void setBaseOutlineStroke(Stroke stroke);
Sets the default outline stroke, to be used when no other stroke is available. If stroke is null, this method throws an IllegalArgumentException.
```

### 36.7.6 The Shapes

Some renderers draw a shape to represent each data item:

► `public Shape getItemShape(int row, int column);`

Returns the shape for the specified data item. This method should never return `null`. The shape will be defined with the point (0, 0) at its centre, so that it can be easily translated into an arbitrary (x, y) position at rendering time.

For the convenience of developers using the renderer API, the interface assumes that shape attributes are stored on a per-series basis (at least) with a default shape that is used when no per-series shape is defined:

► `public Shape getSeriesShape(int series);`

Returns the shape for the specified series, or `null` if no shape is defined for the series.

► `public void setSeriesShape(int series, Shape shape);`

Sets the shape for the specified series (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

► `public Shape getBaseShape();`

Returns the default shape (never `null`). This is used when no other shape is available.

► `public void setBaseShape(Shape shape);`

Sets the default shape to be used when no other shape is available. If `shape` is `null`, this method throws an `IllegalArgumentException`.

### 36.7.7 Item Labels

An *item label* is a short text string that can be displayed near each data item in a chart. Whenever the renderer requires an item label, it obtains a label generator via the following method:

► `public CategoryItemLabelGenerator getLabelGenerator(int series, int item);`

Returns the label generator for the specified data item. In theory, this method could return a different generator for each item but, in practice, it will often return the same generator for every item (or one generator per series). The method can return `null` if no generator has been set for the renderer—in this case, no item labels will be displayed.

To set a generator for a particular series:

► `public CategoryItemLabelGenerator getSeriesItemLabelGenerator(int series);`

Returns the item label generator for the specified series, or `null`.

► `public void setSeriesItemLabelGenerator(int series, CategoryItemLabelGenerator generator);`

Sets the item label generator for the specified series (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

To control the default item label generator:

► `public CategoryItemLabelGenerator getBaseItemLabelGenerator();`

Returns the default item label generator (possibly `null`).

► `public void setBaseItemLabelGenerator(CategoryItemLabelGenerator generator);`

Sets the default item label generator (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

To set a generator that will be used for all data items in the chart (note that this override feature has been deprecated):

► `public void setItemLabelGenerator(CategoryItemLabelGenerator generator); [Deprecated, 1.0.6]`

Sets the item label generator that will be used for ALL data items in the chart, and sends a `RendererChangeEvent` to all registered listeners. Set this to `null` if you prefer to set the generator on a “per series” basis.

### 36.7.8 The Item Label Font

To determine the font used to display each item label (if visible), a renderer will normally call the following method:

► `public Font getItemLabelFont(int row, int column);`

Returns the font to use for the item label for the specified data item. This method should never return `null`.

For the convenience of developers using the renderer API, the interface assumes that a renderer defines the item label font on a per-series basis (at least) with a default value that is used if no per-series font is defined:

► `public Font getSeriesItemLabelFont(int series);`

Returns the item label font for the specified series, or `null`.

► `public void setSeriesItemLabelFont(int series, Font font);`

Sets the item label font for the specified series (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

► `public Font getBaseItemLabelFont();`

Returns the default item label font (never `null`).

► `public void setBaseItemLabelFont(Font font);`

Sets the default item label font and sends a `RendererChangeEvent` to all registered listeners. If `font` is `null`, this method throws an `IllegalArgumentException`.

### 36.7.9 The Item Label Paint

To determine the paint used to display each item label (if visible), a renderer will normally call the following method:

► `public Paint getItemLabelPaint(int row, int column);`

Returns the paint used to draw the item label for the specified data item. This method should never return `null`.

For the convenience of developers using the renderer API, the interface assumes that a renderer defines the item label paint on a per-series basis (at least) with a default value that is used if no per-series paint is defined:

► `public Paint getSeriesItemLabelPaint(int series);`

Returns the item label paint for the specified series, or `null`.

► `public void setSeriesItemLabelPaint(int series, Paint paint);`

Sets the item label paint for the specified series (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

► `public Paint getBaseItemLabelPaint();`

Returns the default item label paint (never `null`).

► `public void setBaseItemLabelPaint(Paint paint);`

Sets the default item label paint and sends a `RendererChangeEvent` to all registered listeners. If `paint` is `null`, this method throws an `IllegalArgumentException`.

### 36.7.10 The Item Label Position

The position of the item labels is specified with two attributes, one for data items with positive values and another for data items with negative values.

For positive data items, the renderer will normally determine the item label position by calling the following method:

► `public ItemLabelPosition getPositiveItemLabelPosition(int row, int column);`

Returns the item label position for the specified data item. This method should never return `null`.

For the convenience of developers using the renderer API, the interface assumes that the item label position is defined on a per-series basis (at least) with a default value that is used if no per-series position is defined:

- `public ItemLabelPosition getSeriesPositiveItemLabelPosition(int series);`  
Returns the item label position (possibly `null`) for positive value items in the specified series.
- `public void setSeriesPositiveItemLabelPosition(int series, ItemLabelPosition position);`  
Equivalent to `setSeriesPositiveItemLabelPosition(series, position, true)`—see the next method.
- `public void setSeriesPositiveItemLabelPosition(int series, ItemLabelPosition position, boolean notify);`  
Sets the item label position for positive value items in the specified series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

To control the default value:

- `public ItemLabelPosition getBasePositiveItemLabelPosition();`  
Returns the default item label position (never `null`) for positive value data items.
- `public void setBasePositiveItemLabelPosition(ItemLabelPosition position);`  
Equivalent to `setBasePositiveItemLabelPosition(position, true)`—see the next method.
- `public void setBasePositiveItemLabelPosition(ItemLabelPosition position, boolean notify);`  
Sets the default item label position for positive value data items and, if requested, sends a `RendererChangeEvent` to all registered listeners.

For negative data items, the renderer will normally determine the item label position by calling the following method:

- `public ItemLabelPosition getNegativeItemLabelPosition(int row, int column);`  
Returns the item label position for the specified data item. This method should never return `null`.

For the convenience of developers using the renderer API, the interface assumes that the item label position is defined on a per-series basis (at least) with a default value that is used if no per-series position is defined:

- `public ItemLabelPosition getSeriesNegativeItemLabelPosition(int series);`  
Returns the item label position (possibly `null` for negative value items in the specified series).
- `public void setSeriesNegativeItemLabelPosition(int series, ItemLabelPosition position);`  
Equivalent to `setSeriesNegativeItemLabelPosition(series, position, true)`—see the next method.
- `public void setSeriesNegativeItemLabelPosition(int series, ItemLabelPosition position, boolean notify);`  
Sets the item label position for negative value items in the specified series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

To control the default value:

- `public ItemLabelPosition getBaseNegativeItemLabelPosition();`  
Returns the default item label position (never `null`) for negative value data items.
- `public void setBaseNegativeItemLabelPosition(ItemLabelPosition position);`  
Equivalent to `setBaseNegativeItemLabelPosition(position, true)`—see the next method.
- `public void setBaseNegativeItemLabelPosition(ItemLabelPosition position, boolean notify);`  
Sets the default item label position for negative value data items and, if requested, sends a `RendererChangeEvent` to all registered listeners.

### 36.7.11 Item Label Visibility

To determine whether or not an item label should be displayed for a data item, the renderer calls the following method:

- ▶ `public boolean isItemLabelVisible(int row, int column);`  
Returns `true` if an item label should be displayed for the specified data item, and `false` otherwise.
- ▶ `public boolean isSeriesItemLabelsVisible(int series);`  
Performs a lookup for the given series to determine whether or not the item labels are visible for the specified series. This method looks at the override, per-series and default flags and returns the appropriate value.<sup>4</sup>

For the convenience of developers using the renderer API, it is assumed that the renderer stores visibility flags on a per-series basis (at least) with a default setting. To control the visibility of item labels for a particular series:

- ▶ `public void setSeriesItemLabelsVisible(int series, boolean visible);`  
Sets a flag that controls whether or not item labels are visible for the specified series.
- ▶ `public void setSeriesItemLabelsVisible(int series, Boolean visible);`  
Equivalent to `setSeriesItemLabelsVisible(series, visible, true)`—see the next method.
- ▶ `public void setSeriesItemLabelsVisible(int series, Boolean visible, boolean notify);`  
Sets a flag that controls whether or not item labels are visible for the specified series and sends a `RendererChangeEvent` to all registered listeners. If `visible` is `null`, the `baseItemLabelsVisible` flag determines the visibility.

The default visibility flag is controlled via the following methods:

- ▶ `public Boolean getBaseItemLabelsVisible();`  
Returns the default value of the flag for item label visibility. A `null` value should be interpreted as `Boolean.FALSE` (this is an error in the API design, the return value should have been a `boolean` primitive).
- ▶ `public void setBaseItemLabelsVisible(boolean visible);`  
Equivalent to `setBaseItemLabelsVisible(Boolean.valueOf(visible))`—see the next method.
- ▶ `public void setBaseItemLabelsVisible(Boolean visible);`  
Equivalent to `setBaseitemLabelsVisible(visible, true)`—see the next method.
- ▶ `public void setBaseItemLabelsVisible(Boolean visible, boolean notify);`  
Sets the default value of the item label visibility flag, and sends a `RendererChangeEvent` to all registered listeners. You should not set this to `null`, but if you do the `null` value will be interpreted as `Boolean.FALSE`.

An override flag is available, but this has been deprecated and you should avoid using it:

- ▶ `public void setItemLabelsVisible(boolean visible); [Deprecated, 1.0.6]`  
Sets the flag that controls whether or not item labels are visible for all series drawn by this renderer. If you prefer to set the visibility on a *per series* basis, you need to set this flag to `null` (see the next method).
- ▶ `public void setItemLabelsVisible(Boolean visible); [Deprecated, 1.0.6]`  
Sets the flag that controls whether or not item labels are visible for all series drawn by this renderer. Set this to `null` if you prefer to set the visibility on a *per series* basis.

---

<sup>4</sup>This method is an error in the API—it should return a `Boolean` flag for just the per-series value. For compatibility reasons, this won't be fixed in the 1.0.x series of JFreeChart releases.

### 36.7.12 Tooltips

A *tool tip* is a short text string that is displayed temporarily in a GUI while the mouse pointer hovers over a particular item in a chart. Whenever the renderer requires a text string for a tool tip, it calls the following method:

► public `CategoryToolTipGenerator` `getToolTipGenerator(int series, int item)`;  
 Returns the tool tip generator for the specified data item (possibly `null`).

For the convenience of developers using the renderer API, the interface assumes that the tool tip generator is defined on a per-series basis (at least) with a default generator that is used if no other is available:

► public `CategoryToolTipGenerator` `getSeriesToolTipGenerator(int series)`;  
 Returns the tooltip generator (possibly `null`) for the data items in the specified series.

► public void `setSeriesToolTipGenerator(int series, CategoryToolTipGenerator generator)`;  
 Sets the tool tip generator (`null` is permitted) for the specified series and sends a `RendererChangeEvent` to all registered listeners.

► public `CategoryToolTipGenerator` `getBaseToolTipGenerator()`;  
 Returns the default tool tip generator (possibly `null`).

► public void `setBaseToolTipGenerator(CategoryToolTipGenerator generator)`;  
 Sets the default tool tip generator (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

There is an override setting, but this has been deprecated and you should avoid using it:

► public `CategoryToolTipGenerator` `getToolTipGenerator()`; [Deprecated 1.0.6]  
 Returns the tool top generator (possibly `null`) that will be used for ALL data items.

► public void `setToolTipGenerator(CategoryToolTipGenerator generator)`; [Deprecated 1.0.6]  
 Sets the tool tip generator that will be used for ALL data items in the chart, and sends a `RendererChangeEvent` to all registered listeners.

### 36.7.13 Item URLs

The `ChartEntity` objects created by the renderer for each data item can have a URL associated with them. To provide flexibility, URLs are generated using a plugin object (similar to the tooltip generator).

*URLs are only used in HTML image maps at present. If you are not generating HTML image maps, then you should leave the URL generators set to null.*

A renderer will normally obtain the URL generator for a data item by calling the following method:

► public `CategoryURLGenerator` `getItemURLGenerator(int series, int item)`;  
 Returns the URL generator for the specified data item. This method may return `null`, in which case no URL will be associated with the data item.

For the convenience of developers using the renderer API, the interface assumes that the item URL generator is defined on a per-series basis (at least) with a default generator that is used if no per-series generator is defined.

► public `CategoryURLGenerator` `getSeriesItemURLGenerator(int series)`;  
 Returns the URL generator (possibly `null`) for the specified series.

► public void `setSeriesItemURLGenerator(int series, CategoryURLGenerator generator)`;  
 Sets the URL generator for the specified series (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

► public `CategoryURLGenerator` `getBaseItemURLGenerator()`;  
 Returns the default URL generator (possibly `null`).

► public void `setBaseItemURLGenerator(CategoryURLGenerator generator)`;  
 Sets the default URL generator (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

### 36.7.14 Series Visibility

All renderers maintain a set of visibility flags to control whether or not a series will be drawn by the renderer. Be aware that some renderers ignore these flags.

At rendering time, JFreeChart will check the visibility of each item using the following method:

```
► public boolean getItemVisible(int series, int item);
```

Returns `true` if the specified item will be drawn by the renderer, and `false` otherwise.

The default implementation of `getItemVisible(int, int)` will simply return the visibility flag for the specified `series`—but you can subclass any renderer and override `getItemVisible()` if you want some different behaviour.

The following method checks the visibility of a series by first checking the per-series flags and, if no flag is defined for the specified series, then checking the base flag value:

```
► public boolean isSeriesVisible(int series);
```

Returns `true` if the specified series will be drawn by the renderer, and `false` otherwise.

A set of visibility flags can be defined on a per-series basis. A flag may be set to `null`, in which case the base flag value will be used.

```
► public Boolean getSeriesVisible(int series);
```

Returns the visibility flag for the specified series. This may be `null`.

```
► public void setSeriesVisible(int series, Boolean visible);
```

Equivalent to `setSeriesVisible(series, visible, true)`—see the next method.

```
► public void setSeriesVisible(int series, Boolean visible, boolean notify);
```

Sets the visibility flag for the specified series and, if requested, sends a `RendererChangeEvent` to all registered listeners. The `visible` argument may be `null`.

The base visibility flag is used for any series with no specific visibility setting:

```
► public boolean getBaseSeriesVisible();
```

Returns the default visibility for all series.

```
► public void setBaseSeriesVisible(boolean visible);
```

Equivalent to `setBaseSeriesVisible(visible, true)`—see the next method.

```
► public void setBaseSeriesVisible(boolean visible, boolean notify);
```

Sets the default visibility for all series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

### 36.7.15 Miscellaneous Methods

The following methods perform drawing tasks that the plot needs to complete, but by delegating these to the renderer it provides an opportunity for the renderer to customise the drawing if necessary.

```
► public void drawBackground(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
```

Draws the background of the plot area (the area enclosed by the axes). A simple implementation for this method is to simply call the `drawBackground()` method in the `CategoryPlot` class—however, the renderers with a 3D effect will do something different.

```
► public void drawOutline(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
```

Draws an outline for the plot area. Most renderers will use a standard implementation (for example, a call to the `drawOutline()` method in the plot), but those renderers that have a 3D effect will implement this method differently.

```
► public void drawDomainGridline(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea, double value);
```

Draws a gridline perpendicular to the domain axis.

```
► public void drawRangeGridline(Graphics2D g2, CategoryPlot plot, ValueAxis axis, Rectangle2D dataArea, double value);
```

Draws a gridline perpendicular to the range axis.

```

→ public void drawDomainMarker(Graphics2D g2, CategoryPlot plot, CategoryAxis axis,
CategoryMarker marker, Rectangle2D dataArea);
Draws a marker against the specified axis.

→ public void drawRangeMarker(Graphics2D g2, CategoryPlot plot, ValueAxis axis,
Marker marker, Rectangle2D dataArea);
Draws a marker against the specified axis.

```

### 36.7.16 Change Listeners

All renderers must support a change notification mechanism, which is designed to allow the `CategoryPlot` (or any other “change listener”) to receive notification whenever any attribute of the renderer changes. To register a change listener with the renderer:

```

→ public void addChangeListener(RendererChangeListener listener);
Registers a listener so that it receives notification of changes to this renderer.

→ public void removeChangeListener(RendererChangeListener listener);
Deregisters a listener so that it no longer receives notification of changes to this renderer.

```

There probably aren’t many situations where you will need to register a listener with the renderer. The `CategoryPlot` class registers itself so that it is notified of changes to any of its renderers and, in turn, passes on a change notification to its own listeners.

### 36.7.17 Notes

Some points to note:

- this interface defines attributes that are common to most renderers. However, some renderers will not use all the attributes defined here (for example, the `BarRenderer` class never uses the `shape` attribute);
- classes that implement the `CategoryItemRenderer` interface are used by the `CategoryPlot` class. They cannot be used by the `XYPlot` class (which uses implementations of the `XYItemRenderer` interface).

#### See Also

[CategoryPlot](#), [AbstractCategoryItemRenderer](#).

## 36.8 CategoryItemRendererState

### 36.8.1 Overview

This class records state information for a `CategoryItemRenderer` during the process of drawing a chart. Recall that the plot uses a renderer to draw the individual data items in a chart. In the plot’s `render()` method, a call is made to the renderer’s `initialise()` method, which returns a state object. Subsequently, for every call the plot makes to the renderer’s `drawItem()` method, it passes in the same state object (which can be updated with new state information during the rendering).

This scheme is designed to allow two or more different threads to use a single renderer to draw a chart to different output targets simultaneously.

### 36.8.2 Constructors

To create a new state instance:

```

→ public CategoryItemRendererState(PlotRenderingInfo info);
Creates a new state object with a reference to the specified info for plot rendering (which may
be null).

```

In general, this class is instantiated in the renderer’s `initialize()` method.

### 36.8.3 Methods

To access the bar width (only used by some renderers):

↳ `public double getBarWidth();`

Returns the bar width, in Java2D units.

↳ `public void setBarWidth(double width);`

Sets the bar width. This method is intended for internal use (by the renderer), you shouldn't call this method directly.

To access the running total for the current series:

↳ `public double getSeriesRunningTotal();`

Returns the running total for the current series.

↳ `void setSeriesRunningTotal(double total);`

Sets the running total for the current series. This method is intended for internal use (by the renderer), you shouldn't call this method directly (actually, it is package private so you can't in any case).

### 36.8.4 Equals, Cloning and Serialization

As this class is intended to represent temporary state information, it is neither cloneable nor serializable, and it does not override the `equals()` method.

#### See Also

[RendererState](#).

## 36.9 CategoryStepRenderer

### 36.9.1 Overview

A renderer that draws a stepped line (or stepped lines) connecting the data items in a `CategoryDataset`—see figure 36.8 for an example. This renderer is designed for use with the `CategoryPlot` class.

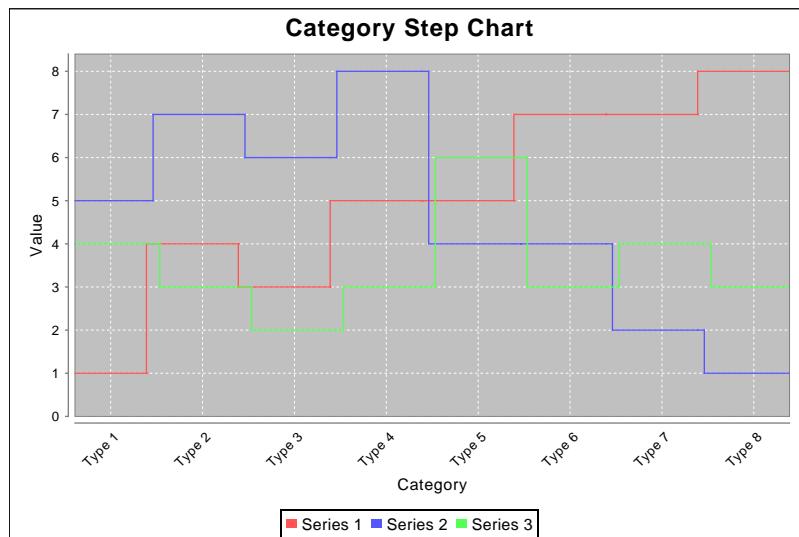


Figure 36.8: A step chart (see `CategoryStepChartDemo1.java`)

### 36.9.2 Constructor

To create a new renderer:

- `public CategoryStepRenderer();`  
Equivalent to `CategoryStepRenderer(false)`—see the next constructor.
- `public CategoryStepRenderer(boolean stagger);`  
Creates a new renderer. If `stagger` is `true`, the vertical steps for each series are offset slightly from one another.

### 36.9.3 Methods

A flag controls whether or not the stepped lines are offset slightly from one another:

- `public boolean getStagger();`  
Returns the flag that controls whether or not the stepped line for each series is offset from the other stepped lines (to avoid the vertical lines overlapping). The initial value is set via the constructors. In the sample chart (see figure 36.8) this flag is set to `true`.
- `public void setStagger(boolean shouldStagger);`  
Sets the flag that controls whether or not the stepped lines for the series are offset from one another, and sends a `RendererChangeEvent` to all registered listeners.

### 36.9.4 Other Methods

All of the following methods are used internally by JFreeChart—you won’t normally call them directly:

- `public LegendItem getLegendItem(int datasetIndex, int series);`  
Overrides the inherited method to generate a legend item with a graphic that is appropriate for this style of renderer.
- `protected CategoryItemRendererState createState(PlotRenderingInfo info);`  
Overrides the inherited method to return a state object with a reusable `Line2D` for drawing purposes.
- `protected void drawLine(Graphics2D g2, State state, PlotOrientation orientation, double x0, double y0, double x1, double y1);`  
Draws a line connecting the specified coordinates (which are in Java2D space). The x and y values will be switched if the orientation is horizontal.
- `public void drawItem(...);`  
Draws one item from the dataset.

### 36.9.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this renderer for equality with an arbitrary object. This method returns `true` if:
  - `obj` is not `null`;
  - `obj` is an instance of `CategoryStepRenderer`;
  - `obj` has the same attributes as this renderer.

Instances of this class are `Cloneable` (`PublicCloneable`) and `Serializable`.

### 36.9.6 Notes

Some points to notes:

- there is no method in the `ChartFactory` class to create a chart using this renderer. Instead, you must manually assign a renderer instance to a `CategoryPlot` instance;
- a demo (`CategoryStepChartDemo1.java`) is included in the JFreeChart demo collection.

**See Also**[XYStepRenderer](#).

## 36.10 DefaultCategoryItemRenderer

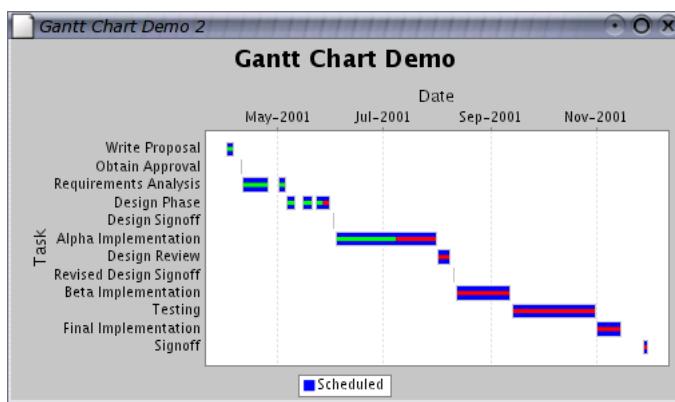
### 36.10.1 Overview

This class is an alias for the [LineAndShapeRenderer](#) class.

## 36.11 GanttRenderer

### 36.11.1 Overview

A renderer that is used to draw simple Gantt charts—an example is shown in figure 36.9.



*Figure 36.9: A Gantt chart*

The renderer is used with the [CategoryPlot](#) class and accesses data via the [GanttCategoryDataset](#) interface. The [createGanttChart\(\)](#) method in the [ChartFactory](#) class will create a [JFreeChart](#) instance that uses this renderer.

### 36.11.2 Methods

The renderer can highlight the “percentage complete” for a task, provided that this information is specified in the dataset. The colors used for this indicator are set with the following methods:

- **public void setCompletePaint(Paint paint);**  
Sets the Paint used to draw the portion of the task that is completed and sends a [RendererChangeEvent](#) to all registered listeners.
- **public void setIncompletePaint(Paint paint);**  
Sets the Paint used to draw the portion of the task that is not yet completed and sends a [RendererChangeEvent](#) to all registered listeners.

The width of the “percentage complete” indicator can be controlled by specifying the start and end percentage values relative to the width (not length!) of the task bars:

- **public void setStartPercent(double percent);**  
Sets the start position for the indicator as a percentage of the width of the task bar (for example, 0.30 is thirty percent)
- **public void setEndPercent(double percent);**  
Sets the end position for the indicator as a percentage of the width of the task bar (for example, 0.70 is seventy percent)

As an example, by setting the start and end percentages in the above methods to 0.30 and 0.70 (say), the middle forty percent of the task bar is occupied by the “percentage complete” indicator.

### 36.11.3 Notes

Some points to note:

- this class extends `IntervalBarRenderer`;
- you can enable or disable bar outlines using the `setDrawBarOutline()` method inherited from the `BarRenderer` class;
- two demo applications (`GanttDemo1.java` and `GanttDemo2.java`) are included in the JFreeChart demo distribution.

## 36.12 GroupedStackedBarRenderer

### 36.12.1 Overview

This renderer is used to draw grouped and stacked bar charts using data from a `CategoryDataset` (see figure 36.10).

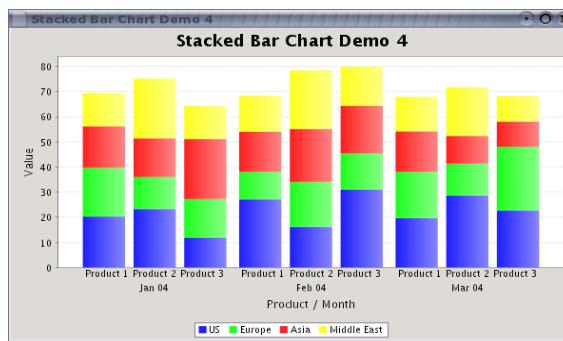


Figure 36.10: A grouped and stacked bar chart

This class extends the `StackedBarRenderer` class.

### 36.12.2 Constructor

To create a new renderer:

```
➔ public GroupedStackedBarRenderer();
```

Creates a new renderer with default settings. By default, all series are mapped to a single group—you can change this using the `setSeriesToGroupMap()` method.

### 36.12.3 Mapping Series To Groups

This renderer requires you to specify the mapping between series and groups using the following method:

```
➔ public void setSeriesToGroupMap(KeyToGroupMap map);
```

Sets the map that controls which series are grouped together.

Refer to the source code for `StackedBarChartDemo4` for an example of this.

### 36.12.4 Other Methods

The following method is called by JFreeChart when determining the axis range that will display ALL the data in the dataset. Due to the stacking performed by this renderer, the range will depend on the way that the series are grouped together:

```
➔ public Range getRangeExtent(CategoryDataset dataset);
    Returns the range of data values in the dataset, after taking into account the stacking that is
    performed by this renderer.
```

### 36.12.5 Notes

Some points to note:

- there is a demo (`StackedBarChartDemo4.java`) included in the JFreeChart demo collection.

## 36.13 IntervalBarRenderer

### 36.13.1 Overview

A renderer that draws bars to represent items from an `IntervalCategoryDataset`—see figure 36.11.

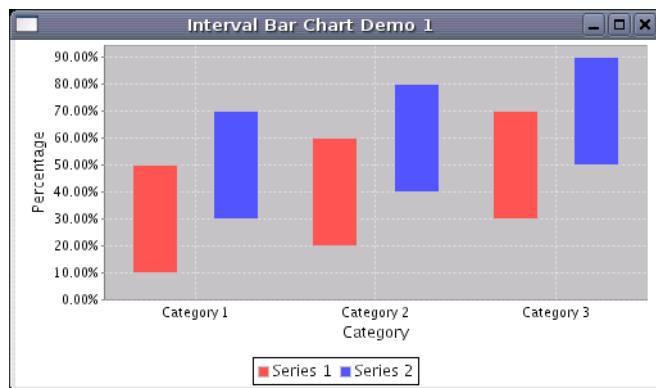


Figure 36.11: A chart that uses an `IntervalBarRenderer`

This renderer is used with the `CategoryPlot` class, and is an extension of `BarRenderer`.

### 36.13.2 Constructors

This class has a single constructor:

```
➔ public IntervalBarRenderer();
Creates a new renderer instance. After the renderer is created, you can customise it using the
methods inherited from its ancestor classes.
```

### 36.13.3 Methods

The following methods are called by the `CategoryPlot` class during chart rendering, you won't normally call them directly:

```
➔ public void drawItem(Graphics2D g2, CategoryItemRendererState state,
    Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis,
    CategoryDataset dataset, int row, int column, int pass);
Handles the drawing of a single item. If dataset is an instance of
IntervalCategoryDataset, the bars are rendered for the interval defined by the dataset. Otherwise,
the method passes control back to the super class to draw a regular bar.
```

```
→ protected void drawInterval(Graphics2D g2, CategoryItemRendererState state,
    Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis,
    ValueAxis rangeAxis, IntervalCategoryDataset dataset, int row, int column);
Handles the drawing of a single interval (called from the drawItem() method).
```

Many other methods are inherited from [BarRenderer](#).

### 36.13.4 Notes

Some points to note:

- the [IntervalCategoryToolTipGenerator](#) interface can be used to generate tooltips with this renderer;
- a demo ([IntervalBarChartDemo1](#)) is included in the [JFreeChart demo collection](#).

#### See Also

[DefaultIntervalCategoryDataset](#), [GanttRenderer](#).

## 36.14 LayeredBarRenderer

### 36.14.1 Overview

A renderer that draws layered bars to represent items from an [CategoryDataset](#)—see figure 36.12 for an example.

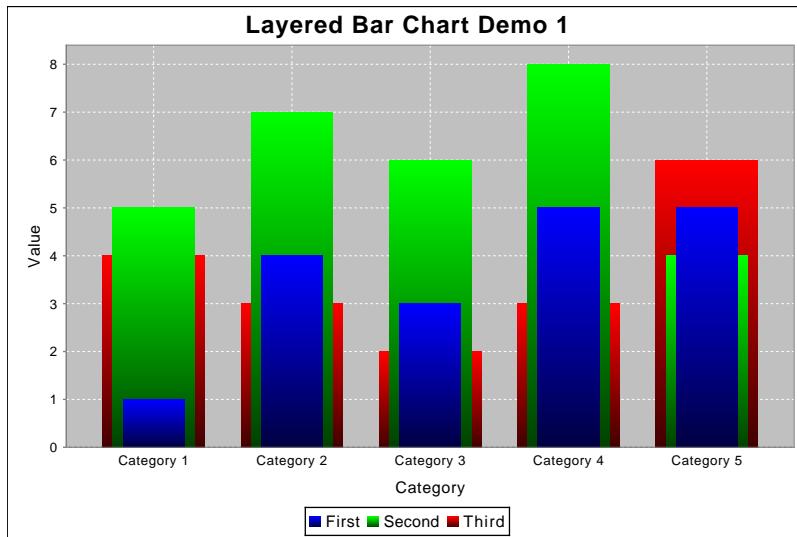


Figure 36.12: A layered bar chart (see [LayeredBarChartDemo1.java](#))

### 36.14.2 Constructors

To create a new renderer:

```
→ public LayeredBarRenderer();
Creates a new renderer with default settings.
```

### 36.14.3 Methods

With this renderer, the bar width varies by series. Most of the time, the default widths are acceptable, but you can specify a custom width if necessary:

```
➔ public double getSeriesBarWidth(int series);
Returns the bar width as a percentage of the default bar width (where 1.0 is 100%).
```

```
➔ public void setSeriesBarWidth(int series, double width);
Sets the bar width as a percentage of the default bar width. Bear in mind that the default bar width decreases as the series index increases.
```

```
➔ protected void calculateBarWidth(CategoryPlot plot, Rectangle2D dataArea, int rendererIndex,
CategoryItemRendererState state);
Updates the state with the bar width calculated for the current series/renderer.
```

```
➔ public void drawItem(Graphics2D g2, CategoryItemRendererState state, Rectangle2D dataArea,
CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis, CategoryDataset data, int
row, int column, int pass);
Draws a bar to represent one data item.
```

```
➔ protected void drawHorizontalItem(Graphics2D g2, CategoryItemRendererState state, Rectangle2D
dataArea, CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis, CategoryDataset
data, int row, int column);
Draws a horizontal bar to represent a data item.
```

```
➔ protected void drawVerticalItem(Graphics2D g2, CategoryItemRendererState state, Rectangle2D
dataArea, CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis, CategoryDataset
data, int row, int column);
Draws a vertical bar to represent a data item.
```

### 36.14.4 Notes

Some points to note:

- a demo ([LayeredBarChartDemo1.java](#)) is included in the JFreeChart demo distribution.

## 36.15 LevelRenderer

### 36.15.1 Overview

A renderer that draws horizontal lines to represent items from an `CategoryDataset`. The lines occupy the same width along the axis that a bar drawn by the `BarRenderer` class would occupy—for example, see figure 36.13.

### 36.15.2 General Attributes

This renderer defines a couple of attributes in addition to those it inherits from its superclass (`AbstractCategoryItemRenderer`).

To control the gap between items:

```
➔ public double getItemMargin();
Returns the item margin as a percentage of the overall length of the category axis (the default
is 0.20, or twenty percent). This controls the amount of space that is allocated to the gaps
between items within the same category.
```

```
➔ public void setItemMargin(double percent);
Sets the item margin and sends a RendererChangeEvent to all registered listeners.
```

To set a cap for the maximum width of each item:

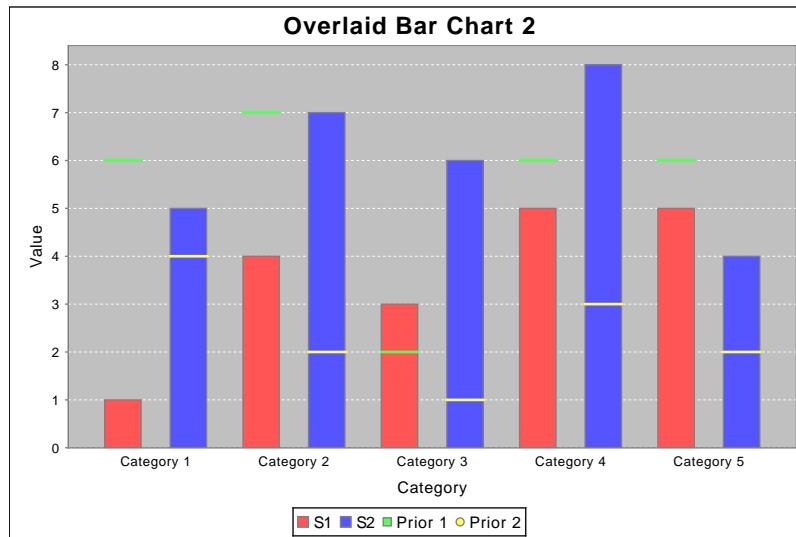


Figure 36.13: A chart that uses a `LevelRenderer`

► `public double getMaximumItemWidth();`

Returns the maximum item width allowed, as a percentage of the length of the category axis. The default is 1.00 (100 percent) which means that the item widths are never capped.

► `public void setMaximumItemWidth(double percent);`

Sets the maximum item width as a percentage of the axis length and sends a `RendererChangeEvent` to all registered listeners. For example, setting this to 0.05 will ensure that the items never exceed five percent of the length of the axis. This can improve the appearance of charts where there is a possibility that only one or two items will be displayed.

### 36.15.3 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`

Tests this renderer for equality with an arbitrary object. Returns `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `LevelRenderer`;
- both renderers have equal field values.

This renderer is `Cloneable` and `Serializable`.

### 36.15.4 Notes

Some points to note:

- the item widths for this renderer are intended to match those calculated by the `BarRenderer` class;
- a demo for this renderer (`OverlaidBarChartDemo2.java`) is included in the JFreeChart demo collection.

## 36.16 LineAndShapeRenderer

### 36.16.1 Overview

A renderer that displays data items by drawing a shape at each data point and/or connecting data points with straight lines—see figure 36.14 for an example. The renderer works with a [CategoryPlot](#) and obtains data from a [CategoryDataset](#).

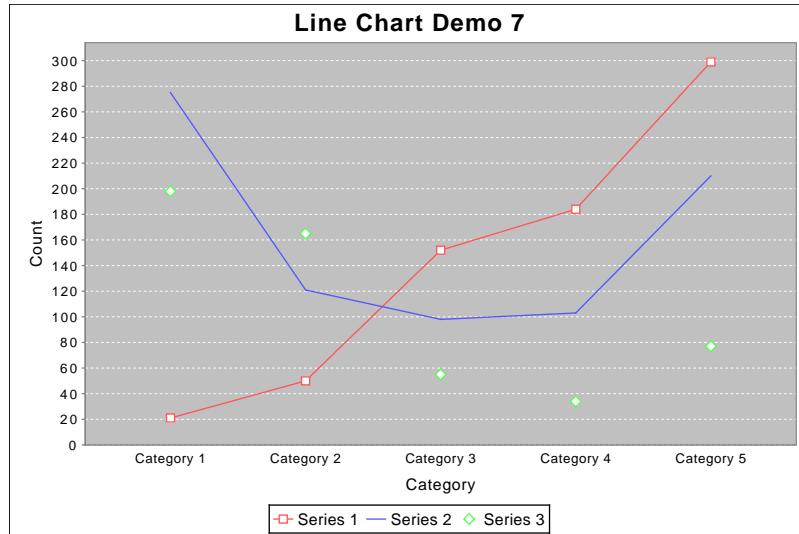


Figure 36.14: A chart that uses a `LineAndShapeRenderer`

### 36.16.2 Usage

You can create a chart that uses this renderer via the `createLineChart()` method in the [ChartUtilities](#) class.

Alternatively, you can install an instance of this renderer on an existing chart (provided that the chart uses a [CategoryPlot](#)) with the following code:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
LineAndShapeRenderer renderer = new LineAndShapeRenderer();
plot.setRenderer(renderer);
```

If you use the latter approach, be aware that the newly created renderer doesn't have tool-tip or item label generators installed (you can add these if you need them).

### 36.16.3 Constructors

The default constructor creates a renderer that draws both shapes and lines:

```
► public LineAndShapeRenderer();
Creates a new renderer that draws both shapes and lines.
```

A second constructor allows you to select shapes and/or lines:

```
► public LineAndShapeRenderer(boolean lines, boolean shapes);
Creates a new renderer that draws shapes and/or lines.
```

### 36.16.4 Line Visibility

To determine if a line should be drawn between the current item and the previous item, the following method is called by the renderer's drawing code:

► `public boolean getItemLineVisible(int series, int item);`

Returns `true` if a line should be drawn between the current item and the previous item, and `false` otherwise.

This method is called for every data item, even though the default implementation can access only the per-series settings provided by the renderer. You can override this method if you need to vary the line visibility on a per-item basis.

The line visibility settings are arranged into the standard three-layer mechanism for renderer attributes (see [35.2.2](#)). To get the top level (override) setting for the visibility of lines:

► `public Boolean getLinesVisible(); [Deprecated 1.0.7]`

Returns `Boolean.TRUE` if lines are visible for all series, `Boolean.FALSE` if lines are invisible for all series, and `null` (the default) if the lower level settings should be referenced instead.

► `public void setLinesVisible(Boolean visible); [Deprecated 1.0.7]`

Sets the override flag for the visibility of lines for all series and sends a `RendererChangeEvent` to all registered listeners. You should set this value to `null` (the default) if you don't require an override setting.

► `public void setLinesVisible(boolean visible); [Deprecated 1.0.7]`

As above.

If the override setting is `null`, the "per series" settings apply:

► `public Boolean getSeriesLinesVisible(int series);`

Returns `Boolean.TRUE` if lines are visible for the specified series, `Boolean.FALSE` if lines are invisible for the specified series, and `null` (the default) if the lower level settings should be referenced instead.

► `public void setSeriesLinesVisible(int series, Boolean flag);`

Sets the line visibility flag for the specified series and sends a `RendererChangeEvent` to all registered listeners. If you set the value for a series to `null`, the base value will be used instead.

► `public void setSeriesLinesVisible(int series, boolean visible);`

As above.

If neither the override nor the per series settings are set, the base level flag is used:

► `public boolean getBaseLinesVisible();`

Returns `true` if lines are visible, by default, for all series, and `false` otherwise.

► `public void setBaseLinesVisible(boolean flag);`

Sets the default value for line visibility and sends a `RendererChangeEvent` to all registered listeners.

The line color (`Paint`) and style (`Stroke`) settings are inherited from the `AbstractRenderer` class.

### 36.16.5 Shapes

The shapes displayed by the renderer (if it is configured to display shapes) are inherited from the `AbstractRenderer` class—see section [35.2.11](#).

### 36.16.6 Shape Visibility

The visibility of shapes can be controlled on a per-item basis. To determine if a shape should be drawn for an item, the renderer will call the following method:

► `public boolean getItemShapeVisible(int series, int item);`  
 Returns `true` if the shape should be drawn for this item, and `false` otherwise. The default implementation of this method does a look-up on the per-series settings for this renderer (see the methods below). Override this method if you want to control shape visibility on a per-item basis.

The override flag, if non-`null`, applies to all series:

► `public Boolean getShapesVisible(); [Deprecated 1.0.7]`  
 Returns the override flag for shape visibility. By default, this method returns `null`.  
 ► `public void setShapesVisible(Boolean visible); [Deprecated 1.0.7]`  
 Sets the override flag for shape visibility, and sends a `RendererChangeEvent` to all registered listeners. You should leave this flag set to `null` if you want the per-series flags to apply.  
 ► `public void setShapesVisible(boolean visible); [Deprecated 1.0.7]`  
 As above.

To control shape visibility on a per-series basis:

► `public Boolean getSeriesShapesVisible(int series);`  
 Returns a flag that indicates whether or not shapes are visible for a series. If this method returns `null`, the base setting will apply.  
 ► `public void setSeriesShapesVisible(int series, Boolean flag);`  
 Sets the flag that controls the visibility of shapes for a series, and sends a `RendererChangeEvent` to all registered listeners. You can set the flag to `null`, in which case the base flag will be used.  
 ► `public void setSeriesShapesVisible(int series, boolean visible);`  
 As above.

The base flag defines the default visibility when both the per-series and override flags are `null`:

► `public boolean getBaseShapesVisible();`  
 Returns `true` if shapes are visible, by default, for all series, and `false` otherwise.  
 ► `public void setBaseShapesVisible(boolean flag);`  
 Sets the default value for shape visibility and sends a `RendererChangeEvent` to all registered listeners.

### 36.16.7 Controlling Shape Outlines

If the renderer is configured to draw shapes, then the shapes can be drawn with or without outlines, according to the setting of the `drawOutlines` flag:

► `public boolean getDrawOutlines();`  
 Returns `true` if the renderer draws outlines around each shape, and `false` otherwise. The default value is `true`.  
 ► `public void setDrawOutlines(boolean flag);`  
 Sets the flag that controls whether or not outlines are drawn around shapes, and sends a `RendererChangeEvent` to all registered listeners.

The renderer uses one of two possible colors (inherited from `AbstractRenderer`) for the shape outlines: (a) the outline paint for the current series, or (b) the (regular) paint for the current series. The selection is determined by the `useOutlinePaint` flag:

► `public boolean getUseOutlinePaint();`  
 Returns `true` if the renderer draws shape outlines using the outline paint, and `false` if the regular series paint is used (the default).  
 ► `public void setUseOutlinePaint(boolean use);`  
 Sets the flag that controls which paint is used for the shape outlines and sends a `RendererChangeEvent` to all registered listeners.

The outline stroke is inherited from the `AbstractRenderer` class—see section 35.2.9.

### 36.16.8 Controlling Shape Filling

The renderer can fill each shape (with either the regular series paint or the series fill paint) or leave the shapes empty (usually only when shape outlines are drawn). The flags that control this are set using the “three layer, per series” approach common to many other renderer attributes.

► `public boolean getItemShapeFilled(int series, int item);`  
 Returns `true` if the shape for the specified item should be filled, and `false` if it should remain unfilled. This method simply calls the `getSeriesShapeFilled(int)` method—override it if you need to control the shape filling on a per item basis.

► `public boolean getSeriesShapesFilled(int series);`  
 Returns `true` if all shapes for the specified series should be filled, and `false` if they should remain unfilled.

An override flag can control shape filling for all series:

► `public Boolean getShapesFilled(); [Deprecated 1.0.7]`  
 Returns `Boolean.TRUE` if all shapes are filled, `Boolean.FALSE` if all shapes are unfilled, and `null` if the override setting does not apply.

► `public void setShapesFilled(boolean filled); [Deprecated 1.0.7]`  
 Sets the override flag for filling all shapes, and sends a `RendererChangeEvent` to all registered listeners.

► `public void setShapesFilled(Boolean filled); [Deprecated 1.0.7]`  
 Sets the override flag for filling all shapes, and sends a `RendererChangeEvent` to all registered listeners. If you set this to `null`, the override setting does not apply.

► `public void setSeriesShapesFilled(int series, boolean filled);`  
 Sets the flag that controls whether or not the shapes are filled for the specified series.

► `public void setSeriesShapesFilled(int series, Boolean filled);`  
 Sets the flag that controls whether or not the shapes are filled for the specified series (if `null`, the default setting applies).

► `public boolean getBaseShapesFilled();`  
 Returns the renderer’s default setting for filling shapes. This will be used only when the override setting is `null` and the per-series setting is `null`.

► `public void setBaseShapesFilled(boolean flag);`  
 Sets the default setting for filling shapes and sends a `RendererChangeEvent` to all registered listeners.

The renderer can use either the regular series paint to fill shapes or the series fill paint, according to the setting of the `useFillPaint` attribute:

► `public boolean getUseFillPaint();`  
 Returns `true` if the renderer should use the series fill paint to fill shapes, and `false` if it should use the regular series paint.

► `public void setUseFillPaint(boolean flag);`  
 Sets the flag that controls whether the series fill paint or the regular series paint is used to fill shapes.

Both the series fill paint and regular series paint settings are inherited from the `AbstractRenderer` class.

### 36.16.9 Rendering Methods

The following methods are used during the chart drawing process, most applications won’t call them directly:

► `public int getPassCount();`  
 Returns 2, to indicate that this renderer requires two passes through the dataset. Lines are drawn in the first pass, and shapes are drawn in the second pass.

```
→ public LegendItem getLegendItem(int datasetIndex, int series);
Returns a legend item for the specified series. The legend item will reflect the line and shape
visibility settings for the specified series.

→ public void drawItem(Graphics2D g2, CategoryItemRendererState state,
Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis,
ValueAxis rangeAxis, CategoryDataset dataset, int row, int column, int pass);
Draws a single item from the dataset. This method is called by the CategoryPlot class, you
don't normally need to call it directly.
```

### 36.16.10 Equals, Cloning and Serialization

This renderer overrides the `equals()` method:

```
→ public boolean equals(Object obj);
Returns true if this renderer is equal to the specified object, and false otherwise.
```

Instances of this class are `Cloneable` (`PublicCloneable`) and `Serializable`.

### 36.16.11 Notes

Some points to note:

- for line charts where the x-values are values (dates or numbers), use an `XYLineAndShapeRenderer`;
- if you set the fill and/or outline paint attributes for this renderer, be sure to check the current settings of the `useFillPaint` and `useOutlinePaint` attributes;

## 36.17 LineRenderer3D

### 36.17.1 Overview

A line renderer that uses a “pseudo-3D” effect to draw line charts on a `CategoryPlot` using data from a `CategoryDataset`.<sup>5</sup>

### 36.17.2 Constructor

To create a new renderer:

```
→ public LineRenderer3D();
Creates a new default renderer.
```

### 36.17.3 General Attributes

The 3D effect is controlled by offsets that can be configured with the following methods:

```
→ public double getXOffset();
Returns the x-offset for the 3D effect. The offset is measured in Java2D units. The default
value is 12.0.

→ public void setXOffset(double xOffset);
Sets the x-offset (in Java2D units) for the 3D effect, and sends a RendererChangeEvent to all
registered listeners.

→ public double getYOffset();
Returns the y-offset for the 3D effect. The offset is measured in Java2D units. The default
value is 8.0.
```

---

<sup>5</sup>Note that we discourage the use of this renderer, since the 3D effect obscures the information content in the chart.

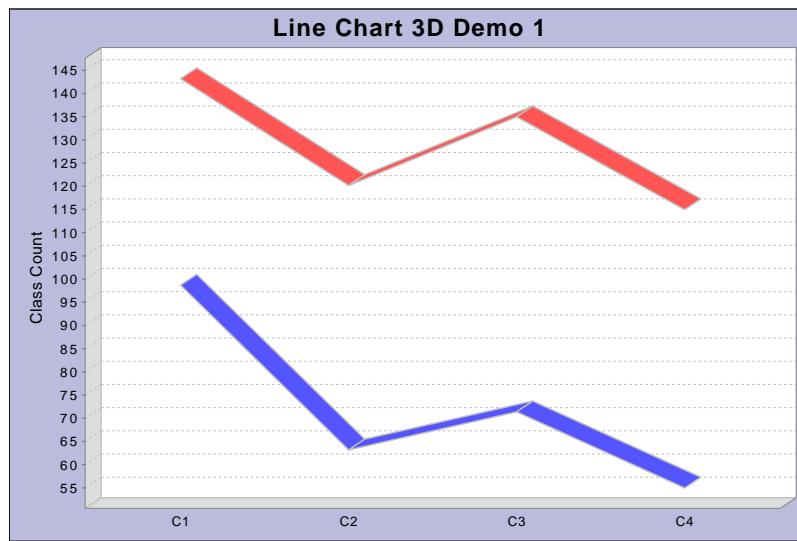


Figure 36.15: A line chart with 3D effect (see `LineChart3DDemo1.java`)

```
→ public void setYOffset(double yOffset);
Sets the y-offset (in Java2D units) for the 3D effect, and sends a RendererChangeEvent to all registered listeners.
```

The colour of the “sides” (or “walls”) of the plot background area can be controlled with the following methods:

```
→ public Paint getWallPaint();
Returns the paint used to colour the offset part of the data area.

→ public void setWallPaint(Paint paint);
Sets the paint used to colour the offset part of the data area, and sends a RendererChangeEvent to all registered listeners.
```

#### 36.17.4 Drawing Methods

Various drawing methods are overridden to incorporate the 3D effect—you won’t normally call any of these methods directly:

```
→ public void drawBackground(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
Draws the plot background, taking into account the 3D offset defined by the renderer.

→ public void drawOutline(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea);
Draws the plot outline, taking into account the 3D offset defined by the renderer.

→ public void drawDomainGridline(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,
double value);
Draws a domain gridline, adjusted for the 3D offset.

→ public void drawRangeGridline(Graphics2D g2, CategoryPlot plot, ValueAxis axis,
Rectangle2D dataArea, double value);
Draws a range gridline, adjusted for the 3D offset.

→ public void drawRangeMarker(Graphics2D g2, CategoryPlot plot, ValueAxis axis,
Marker marker, Rectangle2D dataArea);
Draws a range marker, adjusted for the 3D offset.

→ public void drawItem(Graphics2D g2, CategoryItemRendererState state, Rectangle2D dataArea,
CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis, CategoryDataset dataset,
int row, int column, int pass);
Draws the visual representation of one data item on the chart.
```

### 36.17.5 Equals, Cloning and Serialization

This renderer overrides the `equals()` method:<sup>6</sup>

```
➔ public boolean equals(Object obj);  
Tests this renderer for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

#### See Also

[LineAndShapeRenderer](#).

## 36.18 MinMaxCategoryRenderer

### 36.18.1 Overview

A renderer that plots data from a `CategoryDataset` with:

- an icon for each data item;
- special icons for the minimum and maximum value items within each category;
- a line connecting the minimum and maximum value items within each category.

An example is shown in figure 36.16. This renderer extends `AbstractCategoryItemRenderer`.

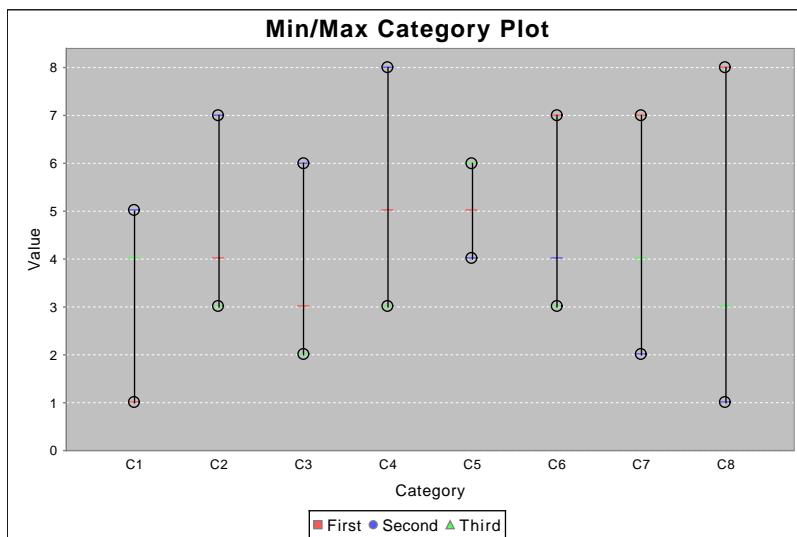


Figure 36.16: A chart that uses a `MinMaxCategoryRenderer`

### 36.18.2 Constructor

To create a new renderer:

```
➔ public MinMaxCategoryRenderer();  
Creates a new renderer with default settings.
```

---

<sup>6</sup>As of JFreeChart 1.0.4.

### 36.18.3 General Attributes

The renderer draws a line between the minimum and maximum value items within each *category*. The **Paint** and **Stroke** for this line are controlled via the following methods:

► **public Paint getGroupPaint();**

Returns the paint used to draw the line between the minimum and maximum value items within each category. The default value is **Color.black**. This method never returns **null**.

► **public void setGroupPaint(Paint paint);**

Sets the paint used to draw the line between the minimum and maximum value items within each category, and sends a **RendererChangeEvent** to all registered listeners. If **paint** is **null**, this method throws an **IllegalArgumentException**.

► **public void setGroupStroke(Stroke groupStroke);**

Returns the stroke used to draw the line between the minimum and maximum value items within each category. The default value is **BasicStroke(1.0f)**. This method never returns **null**.

► **public Stroke getGroupStroke();**

Sets the stroke used to draw the line between the minimum and maximum value items within each category, and sends a **RendererChangeEvent** to all registered listeners. If **stroke** is **null**, this method throws an **IllegalArgumentException**.

This renderer highlights data items by drawing an icon for each data value. The same icon is used for all series:

► **public Icon getObjectIcon();**

Returns the **Icon** that is displayed for each data item (note that a special icon is displayed for the data items with the minimum and maximum values in each category). The default icon is a horizontal line. This method never returns **null**.

► **public void setObjectIcon(Icon icon);**

Sets the **Icon** that is displayed for each data item and sends a **RendererChangeEvent** to all registered listeners. If **icon** is **null**, this method throws an **IllegalArgumentException**.

The maximum and minimum value items within each category are marked with special icons:

► **public Icon getMaxIcon();**

Returns the **Icon** that is displayed for the maximum value data item within each category. The default icon is a hollow circle. This method never returns **null**.

► **public void setMaxIcon(Icon icon);**

Sets the **Icon** that is displayed for the maximum value data item and sends a **RendererChangeEvent** to all registered listeners. If **icon** is **null**, this method throws an **IllegalArgumentException**.

► **public Icon getMinIcon();**

Returns the **Icon** that is displayed for the minimum value data item within each category. The default icon is a hollow circle. This method never returns **null**.

► **public void setMinIcon(Icon minIcon);**

Sets the **Icon** that is displayed for the minimum value data item and sends a **RendererChangeEvent** to all registered listeners. If **icon** is **null**, this method throws an **IllegalArgumentException**.

By default, the renderer does not connect the data points in a series (with a line), however this is configurable via the following methods:

► **public boolean isDrawLines();**

Returns **true** if lines are drawn to connect the items within a series, and **false** otherwise. The default value is **false**.

► **public void setDrawLines(boolean draw);**

Sets the flag that controls whether or not lines are drawn to connect the items within a series. If the new value of the flag is different to the old, a **RendererChangeEvent** is sent to all registered listeners.

If the lines between items are drawn, they use the **paint** and **stroke** attributes inherited from the **AbstractRenderer** class.

### 36.18.4 Drawing Methods

Each item is drawn using the following method:

```
→ public void drawItem(Graphics2D g2, CategoryItemRendererState state, Rectangle2D dataArea,
CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis, CategoryDataset dataset, int
row, int column, int pass);
Draws the specified item. This method is called by the CategoryPlot instance during chart
rendering, you won't normally call it yourself.
```

### 36.18.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj); [1.0.7]
Tests this method for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 36.18.6 Notes

Some points to note:

- a demo (`MinMaxCategoryPlotDemo1.java`) is included in the JFreeChart demo collection;

#### See Also

[AbstractCategoryItemRenderer](#), [StatisticalLineAndShapeRenderer](#).

## 36.19 ScatterRenderer

### 36.19.1 Overview

A renderer that draws shapes on a `CategoryPlot` for all the items in a `MultiValueCategoryDataset`—see figure 36.17 for an example. This class was first introduced in JFreeChart version 1.0.7.

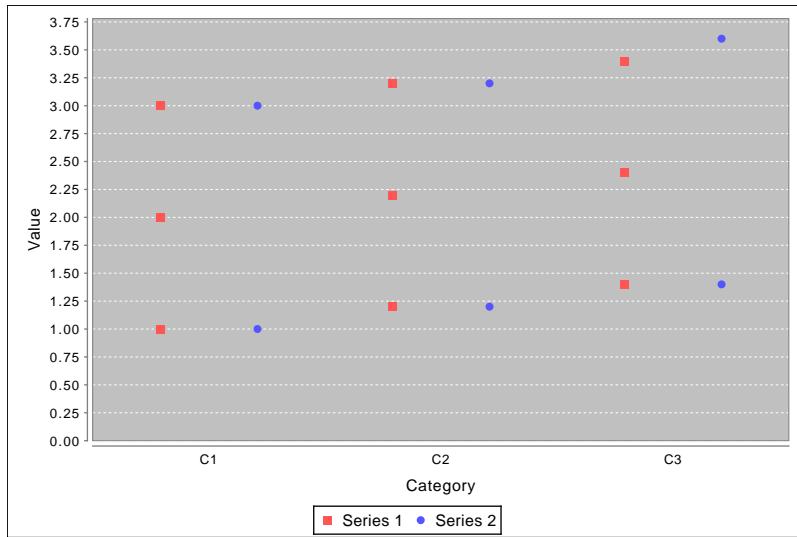


Figure 36.17: Sample chart (see `ScatterRendererDemo1.java`)

### 36.19.2 Constructor

To create a new instance:

```
➔ public ScatterRenderer(); [1.0.7]
Creates a new instance with default attributes.
```

### 36.19.3 General Attributes

To control whether the points for each series are offset from one another:

```
➔ public boolean getUseSeriesOffset(); [1.0.7]
Returns the flag that controls whether or not the data points for the series within each category
are offset from one another. The default value is true.

➔ public void setUseSeriesOffset(boolean offset); [1.0.7]
Sets the flag that controls whether or not series items are offset from one another, and sends a
 to all registered listeners.
```

The item margin controls the spacing between items within a category. It mirrors the behaviour of the item margin for the `BarRenderer` class, so that it is possible to align the shapes plotted by this renderer with bars plotted by another renderer:

```
➔ public double getItemMargin(); [1.0.7]
Returns the item margin, as a percentage of the axis length. The default value is 0.20 (twenty
percent).

➔ public void setItemMargin(double margin); [1.0.7]
Sets the item margin (as a percentage of the axis length) and sends a RendererChangeEvent to
all registered listeners.
```

The shapes for each data item can be drawn with or without outlines:

```
➔ public boolean getDrawOutlines(); [1.0.7]
Returns the flag that controls whether or not the shape outlines are drawn. The default value
is false.

➔ public void setDrawOutlines(boolean flag); [1.0.7]
Sets the flag that controls whether or not the shape outlines are drawn, and sends a RendererChangeEvent
to all registered listeners.
```

The outlines can be drawn using either the regular series paint (the default) or the renderer's outline paint:

```
➔ public boolean getUseOutlinePaint(); [1.0.7]
Returns the flag that controls whether the series outline paint settings are used to draw the
outlines. The default value is false, which means the regular series paint is used.

➔ public void setUseOutlinePaint(boolean use); [1.0.7]
Sets the flag that controls whether the renderer's outline paint or regular paint is used to draw
shape outlines, and sends a RendererChangeEvent to all registered listeners.
```

### 36.19.4 Per Series Attributes

Flags that control whether or not the shapes are filled are defined on a per-series basis for this renderer, with a base or default setting for any series that has no explicit setting.

```
➔ public boolean getItemShapeFilled(int series, int item); [1.0.7]
Returns a flag that determines whether or not the shapes for the specified item are filled. By
default, this method does a lookup on the per-series settings (see the following methods), using
the base setting if no per-series setting is specified.
```

Flags determining whether or not shapes are filled can be set on a per-series basis:

```
➔ public boolean getSeriesShapesFilled(int series); [1.0.7]
Returns a flag for the specified series that controls whether or not the renderer fills the shapes
it draws.
```

► `public void setSeriesShapesFilled(int series, boolean filled); [1.0.7]`  
 Equivalent to `setSeriesShapeFilled(series, Boolean.valueOf(filled))`—see the next method.

► `public void setSeriesShapesFilled(int series, Boolean filled); [1.0.7]`  
 Sets a flag for the specified series that controls whether or not the renderer fills the shapes it draws, and sends a `RendererChangeEvent` to all registered listeners.

► `public boolean getBaseShapesFilled(); [1.0.7]`  
 Returns the default flag that controls whether or not shapes are filled. The default value is `true`.

► `public void setBaseShapesFilled(boolean flag); [1.0.7]`  
 Sets the base flag that controls whether or not the renderer fills shapes, and sends a `RendererChangeEvent` to all registered listeners.

To control whether or not the renderer's fill paint is used:

► `public boolean getUseFillPaint(); [1.0.7]`  
 Returns a flag that controls whether or not the renderer's fill paint is used to fill shapes. The default value is `false`.

► `public void setUseFillPaint(boolean flag); [1.0.7]`  
 Sets the flag that controls whether the renderer's fill paint or regular paint is used to fill shapes, and sends a `RendererChangeEvent` to all registered listeners.

### 36.19.5 Other Methods

The other methods defined (or overridden) in this renderer are intended for internal use by JFreeChart—typically you won't need to call these methods directly.

To draw one item<sup>7</sup> from the dataset:

► `public void drawItem(Graphics2D g2, CategoryItemRendererState state, Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis, CategoryDataset dataset, int row, int column, int pass); 1[0.7]`  
 Draws the visual representation of one data item from the dataset. Note that this method expects `dataset` to be an instance of `MultiValueCategoryDataset`.

To fetch a legend item:

► `public LegendItem getLegendItem(int datasetIndex, int series); [1.0.7]`  
 Returns a legend item representing the specified series.

### 36.19.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj); [1.0.7]`  
 Tests this renderer for equality with an arbitrary object.

Instances of this renderer are `Cloneable` and `Serializable`.

### 36.19.7 Notes

Some points to note:

- this renderer requires a dataset that implements the `MultiValueCategoryDataset` interface;
- a demo (`ScatterRendererDemo1.java`) is included in the JFreeChart demo collection.

---

<sup>7</sup>Bear in mind that an “item” is a list of values, so the renderer may draw multiple shapes for the item.

## 36.20 StackedAreaRenderer

### 36.20.1 Overview

A renderer that draws a “stacked” form of area chart from the data in a `CategoryDataset`. An example is shown in figure 36.18.

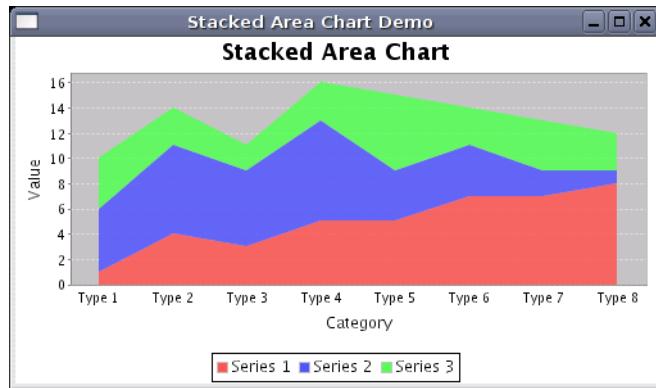


Figure 36.18: A chart that uses a `StackedAreaRenderer`

### 36.20.2 Constructors

This renderer has only the default constructor:

```
→ public StackedAreaRenderer();
Creates a new renderer instance.
```

### 36.20.3 Methods

The super class (`AreaRenderer`) has methods that can be used to customise this renderer. The methods added by this class are intended to be called by other JFreeChart classes, you won’t normally need to call these methods yourself.

```
→ public Range findRangeBounds(CategoryDataset dataset);
Returns the range of values that this renderer requires to display all the items from the dataset.

→ public void drawItem(Graphics2D g2, CategoryItemRendererState state,
Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis,
ValueAxis rangeAxis, CategoryDataset dataset,
int row, int column, int pass);
Draws one item from the dataset.
```

### 36.20.4 Notes

Some points to note:

- a demo (`StackedAreaChartDemo.java`) is included in the JFreeChart demo distribution.

## 36.21 StackedBarRenderer

### 36.21.1 Overview

A renderer that draws stacked bar charts (this class extends the `BarRenderer` class). An example is shown in figure 36.19. This renderer works with a `CategoryPlot` and any dataset that implements the `CategoryDataset` interface.

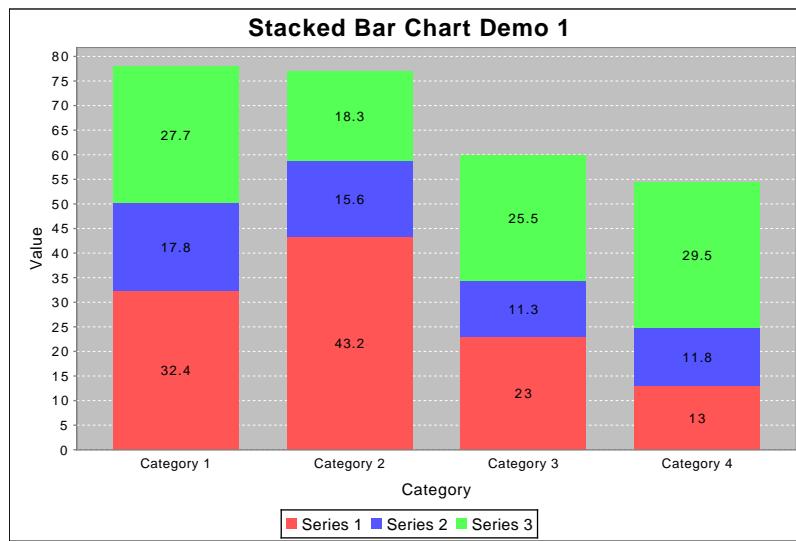


Figure 36.19: A chart that uses a `StackedBarRenderer`

### 36.21.2 Constructors

This renderer has two constructors:

► `public StackedBarRenderer();`

Equivalent to `StackedBarRenderer(false)`—see the next constructor.

► `public StackedBarRenderer(boolean renderAsPercentages);`

Creates a new renderer instance. If `renderAsPercentages` is `true`, each bar will represent a percentage, and all the bars within a category will sum to 100%.

### 36.21.3 General Attributes

Most attributes for this renderer are inherited from the `BarRenderer` class.

#### 36.21.4 Displaying Bars as Percentage Values

The `renderAsPercentages` flag controls the style of chart drawn. If it is set to `true`, the bars all add to 100 % within each category.

► `public boolean getRenderAsPercentages();`

Returns the flag that controls whether each bar represents the data value or its percentage of the category total. The initial value is specified in the constructor.

► `public void setRenderAsPercentages(boolean asPercentages);`

Sets the flag that controls whether each bar represents the data value or its percentage of the category total. A `RendererChangeEvent` is sent to all registered listeners.

### 36.21.5 Other Methods

These methods are used internally by JFreeChart, you won't normally need to call them directly.

► `public int getPassCount();`

Returns 2, the number of times the renderer needs to pass through the dataset for rendering. The second pass is used to draw item labels, if they are visible.

► `public Range findRangeBounds(CategoryDataset dataset);`

Returns the range of values required by the renderer to ensure that all items in the dataset are visible. This is used to set the default axis range.

```
→ public void drawItem(Graphics2D g2, CategoryItemRendererState state,
    Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis,
    ValueAxis rangeAxis, CategoryDataset data, int row, int column, int pass);
    Draws one item from the dataset.
```

### 36.21.6 Equals, Cloning and Serialization

To test the renderer for equality with another object:

```
→ public boolean equals(Object obj);
    Tests this renderer for equality with an arbitrary object.
```

This renderer is `Cloneable` and `Serializable`.

### 36.21.7 Notes

Some points to note:

- to control the space between the bars, see the `setCategoryMargin()` method in the `CategoryAxis` class.
- a demo (`StackedBarChartDemo1.java`) is included in the JFreeChart demo distribution.

#### See Also

[BarRenderer](#), [StackedBarRenderer3D](#).

## 36.22 StackedBarRenderer3D

### 36.22.1 Overview

A renderer that draws stacked bars with a 3D effect. An example is shown in figure 36.20. This renderer works with a `CategoryPlot` class and uses data from any `CategoryDataset`.

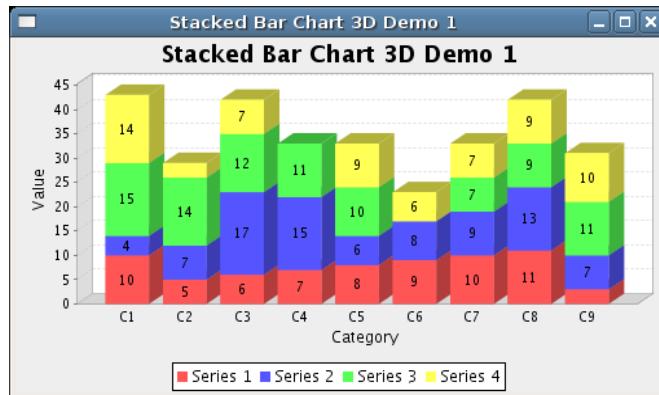


Figure 36.20: A chart that uses a `StackedBarRenderer3D`

### 36.22.2 Constructors

To create a new renderer:

```
→ public StackedBarRenderer3D();
```

Creates a new renderer with the `renderAsPercentages` flag set to `false`. All other defaults are set by the super class (`BarRenderer3D`).

```

→ public StackedBarRenderer3D(double xOffset, double yOffset);
Creates a new renderer with the specified offsets for the 3D effect.

→ public StackedBarRenderer3D(boolean renderAsPercentages);
Creates a new renderer with the specified value for the renderAsPercentages flag. All other
defaults are set by the super class (BarRenderer3D). [Since 1.0.2]

→ public StackedBarRenderer3D(double xOffset, double yOffset, boolean renderAsPercentages);
Creates a new renderer with the specified offsets for the 3D effect and the specified value for
the renderAsPercentages flag. [Since 1.0.2]

```

### 36.22.3 Methods

The `renderAsPercentages` flag controls whether the values are displayed as percentages that stack up to 100%—see figure 36.21 for an example:

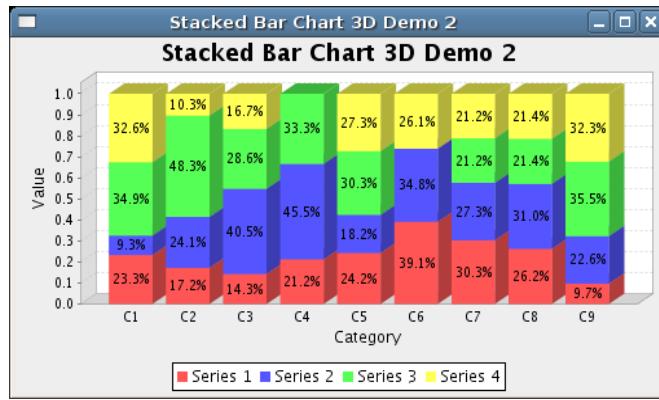


Figure 36.21: A `StackedBarRenderer3D` with `renderAsPercentages` set

```

→ public boolean getRenderAsPercentages();
Returns true if the renderer displays values in percentage terms, and false if the actual values
are displayed. [Since 1.0.2]

→ public void setRenderAsPercentages(boolean asPercentages);
Sets the flag that controls whether the renderer displays values in percentage terms or as
outright values and sends a RendererChangeEvent to all registered listeners. [Since 1.0.2]

```

The remaining methods are called by JFreeChart, you won't normally need to call them yourself:

```

→ public int getPassCount();
Returns 2, the number of passes through the dataset required by the renderer. The second pass
is used to draw the item labels, if they are visible.

→ public Range findRangeBounds(CategoryDataset dataset);
Returns the range of values required by the renderer to display all the items in the dataset.
This is used to set the default axis range. If the renderAsPercentages flag is true, the range is
0.0 to 1.0, otherwise it is the range that covers the sum of the stacked values.

→ public void drawItem(Graphics2D g2, CategoryItemRendererState state,
Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis,
ValueAxis rangeAxis, CategoryDataset data, int row, int column, int pass);
Draws one item from the dataset.

```

### 36.22.4 Equals, Cloning and Serialization

To test this renderer for equality with an arbitrary object:

```
→ public boolean equals(Object obj);  
Returns true if and only if:
```

- `obj` is not `null`;
- `obj` is an instance of `StackedBarRenderer3D`;
- all the fields in this renderer match those in `obj`.

Instances of this class are `Cloneable` and `Serializable`.

### 36.22.5 Notes

Some points to note:

- when using this renderer, you need to ensure that the plot is using axes that support the 3D effect—see `CategoryAxis3D` and `NumberAxis3D`. This is because the size of the data area is slightly reduced to make space for the 3D effect, and the axes need to take this into account;
- a demo (`StackedBarRenderer3DDemo1.java`) is included in the JFreeChart demo distribution.

#### See Also

[BarRenderer3D](#), [StackedBarRenderer](#).

## 36.23 StatisticalBarRenderer

### 36.23.1 Overview

A renderer that draws bars for each data value and then overlays a standard deviation indicator. An example is shown in figure 36.22. This renderer works with a `CategoryPlot` and requires a `StatisticalCategoryDataset`.

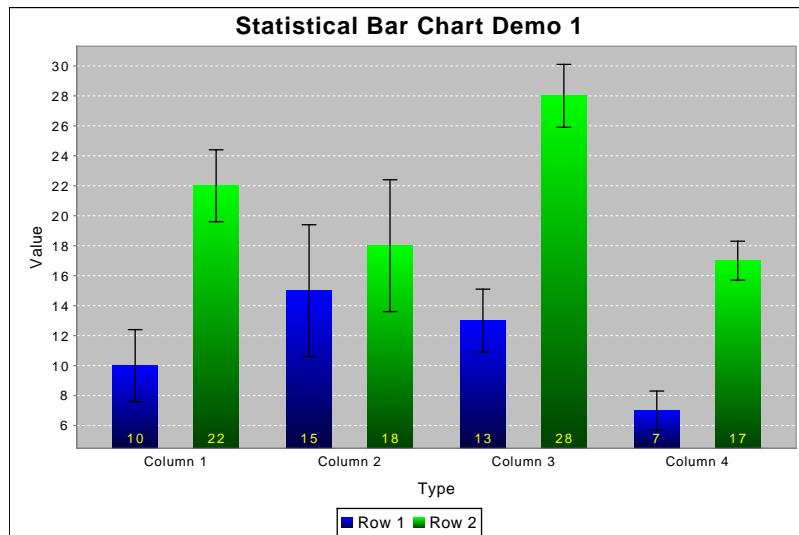


Figure 36.22: A chart that uses a `StatisticalBarRenderer`

### 36.23.2 Constructors

This renderer has only the default constructor:

```
→ public StatisticalBarRenderer();
```

Creates a new renderer instance. The `errorIndicatorPaint` defaults to `Color.gray`. Other defaults are inherited from `BarRenderer`.

### 36.23.3 Attributes

This class inherits most of its attributes from the [BarRenderer](#) class.

To control the colour of the error indicators:

```
→ public Paint getErrorIndicatorPaint();
Returns the paint used to display the error indicator for each bar. If this is null then the item
outline paint is used instead. The default value is Color.gray.

→ public void setErrorIndicatorPaint(Paint paint);
Sets the paint used to display the error indicator for each bar, then sends a RendererChangeEvent
to registered listeners. You can set this to null, in which case the item outline paint will be
used for the error indicators instead.
```

To control the stroke used to draw the error indicators:

```
→ public Stroke getErrorIndicatorStroke(); [1.0.8]
Returns the stroke used to draw the error indicators. The default value is BasicStroke(1.0f).

→ public void setErrorIndicatorStroke(Stroke stroke); [1.0.8]
Sets the stroke used to draw the error indicator for each bar, and sends a RendererChangeEvent
to all registered listeners. You can set this attribute to null, in which case the item outline
stroke will be used instead.
```

### 36.23.4 Other Methods

The renderer overrides the `drawItem()` method, which is called by JFreeChart when a chart is being drawn (normally you won't need to call this method yourself):

```
→ public void drawItem(Graphics2D g2, CategoryItemRendererState state,
Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis,
ValueAxis rangeAxis, CategoryDataset data, int row, int column, int pass);
Draws one item from the dataset.
```

### 36.23.5 Equals, Cloning and Serialization

To test the renderer for equality with another object:

```
→ public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object.
```

This class is `Cloneable` and `Serializable`.

### 36.23.6 Notes

Some points to note:

- a demo (`StatisticalBarChartDemo1.java`) is included in the JFreeChart demo distribution.

#### See Also

[StatisticalLineAndShapeRenderer](#).

## 36.24 StatisticalLineAndShapeRenderer

### 36.24.1 Overview

A renderer that draws lines and/or shapes for each data value and then overlays a standard deviation indicator. An example is shown in figure 36.23. This renderer works with a `CategoryPlot` and requires a `StatisticalCategoryDataset`.

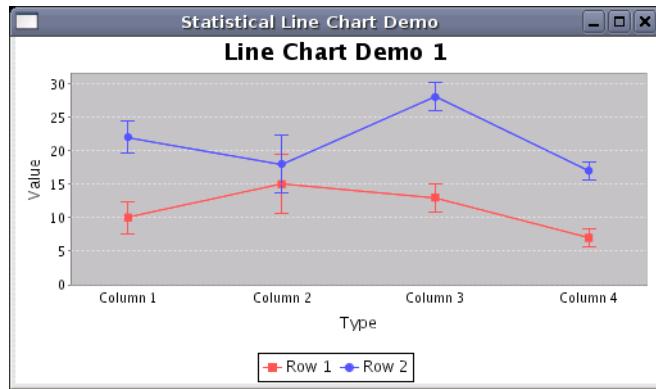


Figure 36.23: A chart that uses a `StatisticalLineAndShapeRenderer`

### 36.24.2 Constructors

This renderer has two constructors:

```
→ public StatisticalLineAndShapeRenderer();
Creates a new renderer instance. By default, both lines and shapes are visible. The errorIndicatorPaint defaults to null, which means the series paint will be used. Other defaults are inherited from LineAndShapeRenderer.
```

```
→ public StatisticalLineAndShapeRenderer(boolean linesVisible,
boolean shapesVisible);
Creates a new renderer instance with lines and/or shapes visible as requested. The errorIndicatorPaint defaults to null, which means the series paint will be used. Other defaults are inherited from LineAndShapeRenderer.
```

### 36.24.3 Attributes

In addition to the attributes inherited from `LineAndShapeRenderer`, this class defines an `errorIndicatorPaint` attribute:

```
→ public Paint getErrorIndicatorPaint();
Returns the paint used to display the error indicator for each item. If this is null then the item paint is used instead (that is, the error indicator will use the same color as the line/shape for the item).
```

```
→ public void setErrorIndicatorPaint(Paint paint);
Sets the paint used to display the error indicator for each item, then sends a RendererChangeEvent to registered listeners. You can set this to null, in which case the item paint will be used for the error indicators instead.
```

### 36.24.4 Other Methods

The renderer overrides the `drawItem()` method, which is called by JFreeChart when a chart is being drawn (normally you won't need to call this method yourself):

```
→ public void drawItem(Graphics2D g2, CategoryItemRendererState state,
Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis,
ValueAxis rangeAxis, CategoryDataset data, int row, int column, int pass);
Draws one item from the dataset.
```

### 36.24.5 Equals, Cloning and Serialization

To test the renderer for equality with another object:

```
➔ public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object.
```

This class is `Cloneable` and `Serializable`.

### 36.24.6 Notes

Some points to note:

- a demo (`StatisticalLineChartDemo1.java`) is included in the JFreeChart demo distribution.

## 36.25 WaterfallBarRenderer

### 36.25.1 Overview

A renderer for drawing “waterfall” charts on a `CategoryPlot` using data from a `CategoryDataset`. A waterfall chart highlights the difference between two values and the components that make up that difference. An example is shown in figure 36.24.

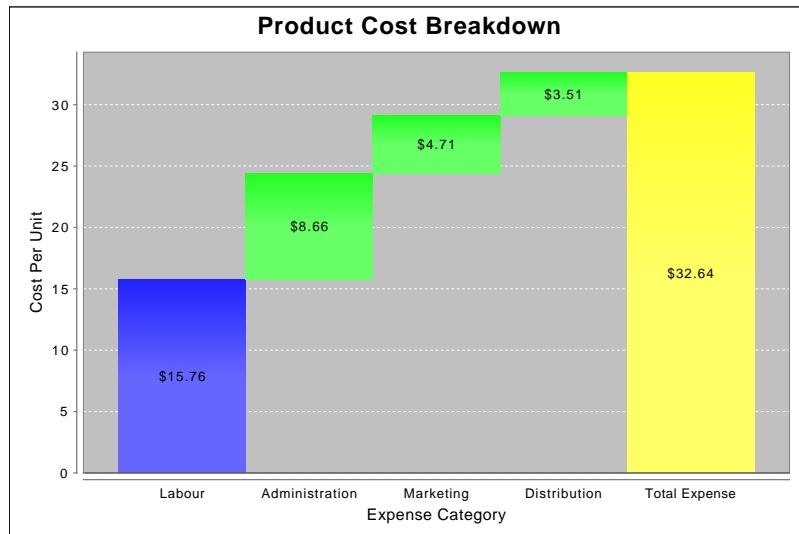


Figure 36.24: Sample chart (see `WaterfallChartDemo1.java`)

### 36.25.2 Constructors

This renderer has two constructors:

```
➔ public WaterfallBarRenderer();
Creates a new renderer with default colors. The defaults are blue for the first value/bar, yellow
for the last value/bar, green for intermediate values that are positive and red for intermediate
values that are negative.

➔ public WaterfallBarRenderer(Paint firstBarPaint,
Paint positiveBarPaint, Paint negativeBarPaint, Paint lastBarPaint);
Creates a new renderer with the specified colors. An IllegalArgumentException is thrown if any
of these is null.
```

### 36.25.3 Methods

This renderer defines the following methods to control the color of the bars it draws:

- ▶ `public Paint getFirstBarPaint();`  
Returns the paint used for the first bar drawn—this will never be `null`.
- ▶ `public void setFirstBarPaint(Paint paint);`  
Sets the paint used for the first bar, and sends a `RendererChangeEvent` to all registered listeners. An `IllegalArgumentException` is thrown if the supplied argument is `null`.
- ▶ `public Paint getLastBarPaint();`  
Returns the paint used for the last bar drawn—this will never be `null`.
- ▶ `public void setLastBarPaint(Paint paint);`  
Sets the paint used for the last bar drawn by the renderer, and sends a `RendererChangeEvent` to all registered listeners. An `IllegalArgumentException` is thrown if the supplied argument is `null`.
- ▶ `public Paint getPositiveBarPaint();`  
Returns the paint used for intermediate bars that have a positive value—this will never be `null`.
- ▶ `public void setPositiveBarPaint(Paint paint);`  
Sets the paint used for the intermediate bars representing positive values, and sends a `RendererChangeEvent` to all registered listeners. An `IllegalArgumentException` is thrown if the supplied argument is `null`.
- ▶ `public Paint getNegativeBarPaint();`  
Returns the paint used for intermediate bars that have a negative value—this will never be `null`.
- ▶ `public void setNegativeBarPaint(Paint paint);`  
Sets the paint used for the intermediate bars representing negative values, and sends a `RendererChangeEvent` to all registered listeners. An `IllegalArgumentException` is thrown if the supplied argument is `null`.

Further methods for customising the renderer are inherited from the `BarRenderer` class.

### 36.25.4 Other Methods

The renderer has a couple of other methods that will be called by the `CategoryPlot` class when it is drawing the chart—you won't typically call these methods directly.

- ▶ `public Range findRangeBounds(CategoryDataset dataset);`  
Returns the range of values that this renderer needs to display all the data in the specified dataset.
- ▶ `public void drawItem(Graphics2D g2, CategoryItemRendererState state, Rectangle2D dataArea, CategoryPlot plot, CategoryAxis domainAxis, ValueAxis rangeAxis, CategoryDataset dataset, int row, int column, int pass);`  
Draws one item from the dataset.

### 36.25.5 Equals, Cloning and Serialization

To test an object for equality with this renderer:

- ▶ `public boolean equals(Object obj);`  
Tests this renderer for equality with an arbitrary object.

This renderer can be `Cloneable` and `Serializable`.

### 36.25.6 Notes

Some points to note:

- a “shortcut” has been taken in the implementation of this renderer: the value for the last bar could be derived from the values of the other bars, but instead the renderer expects the final value to be part of the dataset. This means that you need to ensure that the final value corresponds to the sum of the preceding values (although this is not enforced).
- the `createWaterfallChart()` method in the `ChartFactory` class can be used to create a “ready made” chart;
- a demo (`WaterfallChartDemo1.java`) is included in the JFreeChart demo distribution.

# Chapter 37

## Package: `org.jfree.chart.renderer.xy`

### 37.1 Overview

This package contains interfaces and classes that are used to implement renderers for the `XYPlot` class. Renderers offer a lot of scope for changing the appearance of your charts, either by changing the attributes of an existing renderer, or by implementing a completely new renderer. JFreeChart provides many different renderers, and most of them are easily customised.

### 37.2 AbstractXYItemRenderer

#### 37.2.1 Overview

A base class (extending `AbstractRenderer`) that can be used for creating new `XYItemRenderer` implementations. Subclasses include:

- `CandlestickRenderer`;
- `HighLowRenderer`;
- `StandardXYItemRenderer`;
- `WindItemRenderer`;
- `XYAreaRenderer`;
- `XYAreaRenderer2`;
- `XYBarRenderer`;
- `XYBlockRenderer`;
- `XYBoxAndWhiskerRenderer`;
- `XYBubbleRenderer`;
- `XYDifferenceRenderer`;
- `XYDotRenderer`;
- `XYLineAndShapeRenderer`;
- `XYStepAreaRenderer`;
- `YIntervalRenderer`.

### 37.2.2 Constructors

This class provides a default constructor:

► `protected AbstractXYItemRenderer();`

Creates a new renderer. This allocates storage for the label generator(s), tool tip generator(s) and the URL generator.

### 37.2.3 The Plot

Renderers are assigned to plots, and the following methods are used to provide a link from a renderer to the plot it has been assigned to:

► `public XYPlot getPlot();`

Returns the plot that the renderer is currently assigned to. This method returns `null` if the renderer has not been assigned to a plot yet.

► `public void setPlot(XYPlot plot);`

Sets the plot that the renderer is assigned to. This method is called by JFreeChart—you shouldn't need to call it yourself (and, in fact, setting this field incorrectly will have unpredictable results).

### 37.2.4 Item Labels

Item labels are small text strings (typically showing just the data value) displayed on or near each data item. Flags that control whether or not item labels are visible are inherited from the `AbstractRenderer` class—see section 35.2.15. Most renderers have support for item labels, but some still need this to be implemented.

To allow for customisation of the labels themselves, the renderer uses a generator to create the text for the labels. In most cases, you can probably use the same generator for all series, so it is sufficient just to set the base (or default) item label generator:

► `public XYItemLabelGenerator getBaseItemLabelGenerator();`

Returns the default item label generator (possibly `null`).

► `public void setBaseItemLabelGenerator(XYItemLabelGenerator generator);`

Sets the default item label generator (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

For greater control, you can specify a different generator for each series:

► `public XYItemLabelGenerator getSeriesItemLabelGenerator(int series);`

Returns the item label generator for the specified series. This method can return `null`.

► `public void setSeriesItemLabelGenerator(int series, XYItemLabelGenerator generator);`

Sets the generator for the specified series, and sends a `RendererChangeEvent` to all registered listeners. You can set this to `null` if you want the base generator to be used.

There is an option to specify an override generator that will be used for ALL series, although it is rare that it is necessary to use this:

► `public XYItemLabelGenerator getItemLabelGenerator(); [1.0.5]`

Returns the override generator for ALL series. The default value is `null`.

► `public void setItemLabelGenerator(XYItemLabelGenerator generator);`

Sets the override generator for ALL series. You can set this to `null`, in which case the per-series and base settings will determine the generator to be used.

JFreeChart itself will always call the following method to retrieve a generator to use for a particular data item—this provides a single override point if you want to take full control over the selection of a generator to use:

► `public XYItemLabelGenerator getItemLabelGenerator(int series, int item);`

Returns the item label generator for the specified data item. If this is `null`, no item label will be displayed.

### 37.2.5 Tool Tip Generators

The renderer can generate tool tips for each data item. These are supported for charts displayed in a [ChartPanel](#), as well as in HTML image maps. JFreeChart calls the following method to obtain a tool tip generator for each data item:

```
► public XYToolTipGenerator getToolTipGenerator(int series, int item);
```

Returns the tool tip generator for the specified data item. This method can return `null`, in which case no tool tip will be displayed for the data item.

In general, you can define a single default tool tip generator that can be used for all data items:

```
► public XYToolTipGenerator getBaseToolTipGenerator();
```

Returns the default tool tip generator (possibly `null`).

```
► public void setBaseToolTipGenerator(XYToolTipGenerator generator);
```

Sets the default tool tip generator (`null` is permitted) and sends a [RendererChangeEvent](#) to all registered listeners.

If necessary, however, you can specify a different tool tip generator for each series:

```
► public XYToolTipGenerator getSeriesToolTipGenerator(int series);
```

Returns the tool tip generator for the specified series, or `null` if no generator is defined for that series.

```
► public void setSeriesToolTipGenerator(int series, XYToolTipGenerator generator);
```

Sets the tool tip generator for a series, and sends a [RendererChangeEvent](#) to all registered listeners.

An override generator can be specified for ALL series, although it is rare that this is required:

```
► public XYToolTipGenerator getToolTipGenerator(); [1.0.5]
```

Returns the override generator to be used for ALL series. The default value is `null`.

```
► public void setToolTipGenerator(XYToolTipGenerator generator);
```

Sets the override generator to use for ALL series, and sends a [RendererChangeEvent](#) to all registered listeners.

### 37.2.6 URLs

URLs are used in the construction of HTML image maps (see the [ImageMapUtilities](#) class). To allow for customisation of the URLs generated for each data item, you can supply a URL generator to the renderer:

```
► public XYURLGenerator getURLGenerator();
```

Returns the URL generator (possibly `null`).

```
► public void setURLGenerator(XYURLGenerator urlGenerator);
```

Sets the URL generator (`null` is permitted) and sends a [RendererChangeEvent](#) to all registered listeners.

There are currently no per-series settings for this attribute.

### 37.2.7 Hot Spot

For tool tips and URLs, the “hot spot” is the area within which the tool tip or URL is active. In general, the renderer will determine the shape of the hot spot according to the visual representation of the data (for example, in a bar chart the hot spot is usually the entire bar). As a fall back, however, a generic hotspot will be calculated as a circle with the following radius:

```
► public int getDefaultEntityRadius();
```

Returns the default entity radius, in Java2D units. The default value is `3.0`.

```
► public void setDefaultEntityRadius(int radius);
```

Sets the default entity radius, in Java2D units. No change event is generated in this case.

### 37.2.8 Annotations

Annotations are small graphical items that can be drawn at specific locations on a plot. We provide the facility to add annotations to a renderer, so that the location of the annotation can be computed using the axes that the renderer is linked to:

```
→ public void addAnnotation(XYAnnotation annotation);
Equivalent to addAnnotation(annotation, Layer.FOREGROUND)—see the next method.

→ public void addAnnotation(XYAnnotation annotation, Layer layer);
Adds an annotation to the renderer, for drawing in the specified layer, and sends a RendererChangeEvent to all registered listeners.

→ public boolean removeAnnotation(XYAnnotation annotation);
Removes the specified annotation, returning true if the annotation is actually removed, and false if the annotation was not removed (for example, if the annotation wasn't in fact assigned to this renderer).

→ public void removeAnnotations();
Removes all annotations and sends a RendererChangeEvent to all registered listeners.
```

### 37.2.9 Legend Items

The renderer is responsible for creating legend items, so that the legend item for each series matches whatever visual representation the renderer creates. JFreeChart will call the following method to obtain a collection of legend items for all the series that the renderer is responsible for:

```
→ public LegendItemCollection getLegendItems();
Returns a collection of legend items (the collection may be empty). Note that a new collection is returned each time this method is called—modifying the returned collection has no effect on the renderer.
```

To create a legend item for a series (this method is called by the plot):

```
→ public LegendItem getLegendItem(int index, int series);
Returns a legend item that represents the specified series. The index argument tells the renderer which dataset it is rendering (only the plot tracks this)—0 for the primary dataset, or n+1 for a secondary dataset (where n is the index of the secondary dataset).
```

The series label for each legend item is created by a generator:

```
→ public XYSeriesLabelGenerator getLegendItemLabelGenerator();
Returns the series label generator for the legend items. This method never returns null. The default value is StandardXYSeriesLabelGenerator("0").

→ public void setLegendItemLabelGenerator(XYSeriesLabelGenerator generator);
Sets the legend item label generator and sends a RendererChangeEvent to all registered listeners. If generator is null, this method throws an IllegalArgumentException.
```

Similarly, the tool tip for each legend item is created by a generator:

```
→ public XYSeriesLabelGenerator getLegendItemToolTipGenerator();
Returns the tool tip generator for the legend items. The default value is null.

→ public void setLegendItemToolTipGenerator(XYSeriesLabelGenerator generator);
Sets the legend item tool tip generator and sends a RendererChangeEvent to all registered listeners. The generator argument can be null.
```

And finally, the URL for each legend item (only used in HTML image maps) is

```
→ public XYSeriesLabelGenerator getLegendItemURLGenerator();
Returns the generator that is responsible for creating URLs for each legend item. The default value is null.

→ public void setLegendItemURLGenerator(XYSeriesLabelGenerator generator);
Sets the legend item URL generator and sends a RendererChangeEvent to all registered listeners. The generator argument can be null.
```

### 37.2.10 The Pass Count

The *pass count* refers to the number of times the `XYPlot` scans through the dataset passing individual data items to the renderer for drawing. Most renderers require only a single pass through the dataset, but some will use a second pass to overlay shapes (for example) over previously drawn items.

The plot will call the following method to determine how many passes the renderer requires:

```
➔ public int getPassCount();
```

Returns 1 to indicate that the renderer requires only a single pass through the dataset.

Renderers that require more than one pass through the dataset should override this method.

### 37.2.11 Domain and Range Markers

A default method is supplied for displaying a *domain marker* as a line on the plot:

```
➔ public void drawDomainMarker(...);
```

Draws a line perpendicular to the domain axis to represent a `Marker`.

A default method is supplied for displaying a *range marker* as a line on the plot:

```
➔ public void drawRangeMarker(...);
```

Draws a line perpendicular to the range axis to represent a `Marker`.

Most renderers will use these methods by default, but some may override them.

### 37.2.12 Grid Bands

It is possible to fill the space between alternate grid lines with a different color to create a “band” effect.

### 37.2.13 Initialisation

Each time a chart is drawn, the plot will initialise the renderer by calling the following method:

```
➔ public XYItemRendererState initialise()
```

Initialises the renderer and returns a state object that the plot will pass to all subsequent calls to the `drawItem()` method. The state object is discarded once the chart is fully drawn.

### 37.2.14 Other Methods

Other methods include:

```
➔ public Range findDomainBounds(XYDataset dataset);
```

Returns the range that should be set for the domain axis in order that all the data in the specified `dataset` will be visible when drawn by this renderer.

```
➔ public Range findRangeBounds(XYDataset dataset);
```

Returns the range that should be set for the range axis in order that all the data in the specified `dataset` will be visible when drawn by this renderer. Some renderers (for example, those that stack values) will override this method.

```
➔ public DrawingSupplier getDrawingSupplier();
```

A convenience method that returns the drawing supplier for the plot that the renderer is assigned to, or `null` if the renderer is not currently assigned to a plot. The drawing supplier provides a single source for obtaining series colors (and other attributes), so that multiple renderers in a single plot do not end up using duplicate colors.

### 37.2.15 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);  
    Tests this renderer for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 37.2.16 Notes

Some points to note:

- this class provides a property change mechanism to support the requirements of the `XYItemRenderer` interface;

#### See Also

[XYItemRenderer](#), [XYPlot](#).

## 37.3 CandlestickRenderer

### 37.3.1 Overview

A *candlestick renderer* draws each item from an `OHLCDataset` as a box with lines extending from the top and bottom. Candlestick charts are typically used to display financial data—the box represents the open and closing prices, while the lines indicate the high and low prices for a trading period (often one day). This renderer is designed for use with the `XYPlot` class.

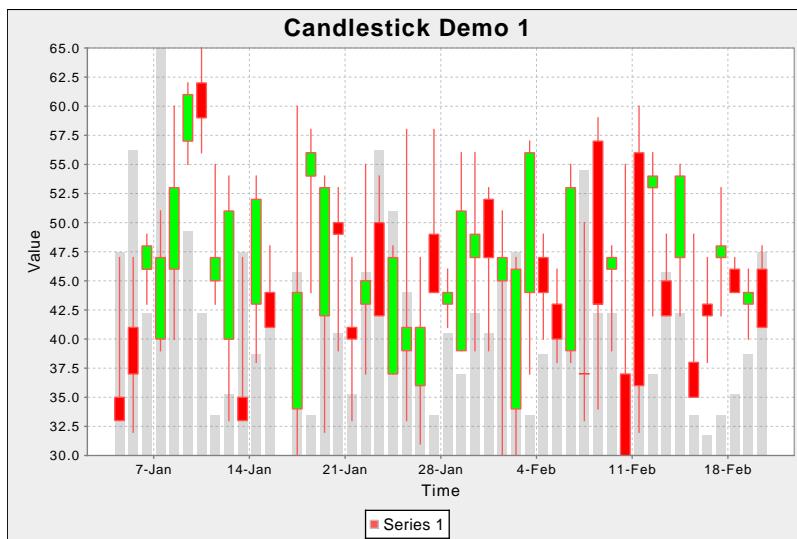


Figure 37.1: A sample chart (see `CandlestickChartDemo1.java`)

This renderer also has the ability to represent volume information in the background of the chart, although the same effect is sometimes better achieved with a second dataset, renderer and range axis.

### 37.3.2 Constructors

To create a new renderer:

```
→ public CandlestickRenderer();
Equivalent to CandlestickRenderer(-1.0) – see the next constructor.

→ public CandlestickRenderer(double candleWidth);
Equivalent to CandlestickRenderer(candleWidth, true, new HighLowItemLabelGenerator()) – see
the next constructor.

→ public CandlestickRenderer(double candleWidth, boolean drawVolume,
XYToolTipGenerator toolTipGenerator);
Creates a new renderer. If candleWidth is -1, the width will be calculated automatically. The
drawVolume flag controls whether or not volume bars are drawn in the background of the plot.
The toolTipGenerator is used to create tool tips, or you can leave this as null.
```

### 37.3.3 General Attributes

This class inherits many attributes from [AbstractXYItemRenderer](#). In addition, it defines some attributes of its own.

To control the width of the candles:

```
→ public double getCandleWidth();
Returns the width of the candles in Java2D units.

→ public void setCandleWidth(double width);
Sets the width of each candle in Java2D units, and sends a RendererChangeEvent to all registered
listeners. If the value is negative, then the renderer will automatically determine a width each
time the chart is redrawn.
```

To control the paint used to fill candles:

```
→ public Paint getUpPaint();
Returns the paint used to fill candles where the closing price is higher than the opening price
(that is, where the price has gone up). If this is null, the renderer will fill the candle using the
regular series paint.

→ public void setUpPaint(Paint paint);
Sets the fill color for candles where the closing price is higher than the opening price, and sends
a RendererChangeEvent to all registered listeners. You can set this to null, in which case the
renderer uses the regular series paint to fill the candles.
```

Similarly for the “down” paint:

```
→ public Paint getDownPaint();
Returns the paint used to fill candles where the closing price is lower than the opening price
(that is, where the price has gone down). If this is null, the renderer will fill the candle using the
regular series paint.

→ public void setDownPaint(Paint paint);
Sets the fill color for candles where the closing price is lower than the opening price, and sends
a RendererChangeEvent to all registered listeners. You can set this to null, in which case the
renderer uses the regular series paint to fill the candles.
```

This renderer can draw the outline of the candles using either the regular series paint or the series outline paint:

```
→ public boolean getUseOutlinePaint(); [1.0.5]
Returns the flag that controls whether the renderer uses the regular series paint or the series
outline paint to draw the outlines.

→ public void setUseOutlinePaint(boolean use); [1.0.5]
Sets the flag that controls whether the renderer uses the series outline paint, and sends a
RendererChangeEvent to all registered listeners.
```

To control whether or not volume bars are drawn in the background of the chart:

► `public boolean getDrawVolume(); [1.0.5]`

Returns the flag that controls whether or not the renderer draws volume bars in the background of the chart. The initial value is set in the constructor.

► `public void setDrawVolume(boolean flag);`

Sets the flag that controls whether or not volume bars are drawn in the background of the plot, and sends a `RendererChangeEvent` to all registered listeners.

To control the colour used to draw the volume bars (if they are visible):

► `public Paint getVolumePaint(); [1.0.7]`

Returns the paint (never `null`) used to fill the volume bars. The default value is `Color.gray`.

► `public void setVolumePaint(Paint paint); [1.0.7]`

Sets the paint used to fill the volume bars, and sends a `RendererChangeEvent` to all registered listeners. If paint is `null`, this method throws an `IllegalArgumentException`.

### 37.3.4 Auto Width Calculation

If the candle width attribute is set to `-1`, the renderer will automatically determine the candle width at drawing time.

► `public int getAutoWidthMethod();`

Returns an integer that indicates the type of auto-width calculation. The default value is `WIDTHMETHOD_AVERAGE`.

► `public void setAutoWidthMethod(int autoWidthMethod);`

Sets the type of auto width calculation and sends a `RendererChangeEvent` to all registered listeners. The valid types are:

- `WIDTHMETHOD_AVERAGE` – the default;

- `WIDTHMETHOD_SMALLEST` – bases the width on the smallest gap between consecutive x-values in the dataset;

- `WIDTHMETHOD_INTERVALDATA` – assumes that the dataset implements the `IntervalXYDataset` interface, and looks at the x-interval to determine the width.

After the candle width is calculated according to the width calculation method, it is then subject to three adjustments. First, the `autoWidthGap` is subtracted, then the width is multiplied by the `autoWidthFactor`, and finally the width is capped according to the `maxCandleWidthInMilliseconds`:

► `public double getAutoWidthGap();`

Returns the amount of space (measured in Java2D units) to leave *on each side* of every candle drawn by the renderer. The default value is `0.0`.

► `public void setAutoWidthGap(double autoWidthGap);`

Sets the amount of space to leave on each side of every candle, and sends a `RendererChangeEvent` to all registered listeners.

► `public double getAutoWidthFactor();`

Returns the factor by which the candle width is multiplied during the auto-width calculation. The default value is `4.5 / 7.0`.

► `public void setAutoWidthFactor(double autoWidthFactor);`

Sets the factor by which the candle width is multiplied during the auto-width calculation, and sends a `RendererChangeEvent` to all registered listeners.

► `public double getMaxCandleWidthInMilliseconds();`

The maximum candle width in milliseconds. The default value is equivalent to 20 hours, which is a reasonable value for displaying daily data.

► `public void setMaxCandleWidthInMilliseconds(double millis);`

Sets the maximum candle width in milliseconds (along the axis) and sends a `RendererChangeEvent` to all registered listeners.

### 37.3.5 Other Methods

These methods are called by the `XYPlot` class—you won't normally call them directly:

```
↳ public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot,
XYDataset dataset, PlotRenderingInfo info);
```

Initialises the renderer. This method is called once each time a chart is drawn.

```
↳ public void drawItem(...);
```

Draws a single item from the dataset.

### 37.3.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
↳ public boolean equals(Object obj);
```

Tests this renderer for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 37.3.7 Notes

Some points to note:

- this renderer expects the dataset to be an implementation of the `OHLCDataset` interface;
- a demo (`CandlestickChartDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[HighLowRenderer](#).

## 37.4 ClusteredXYBarRenderer

### 37.4.1 Overview

A renderer that draws bars from the items in an `IntervalXYDataset`—see figure ??.

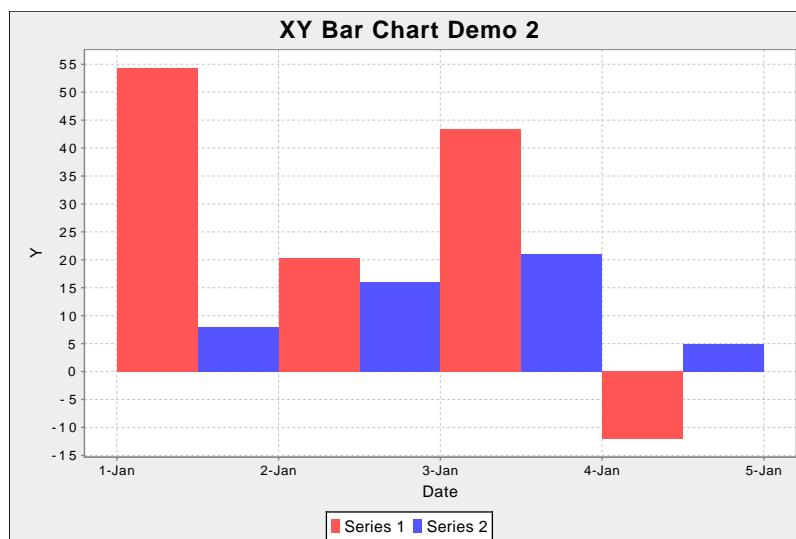


Figure 37.2: A sample chart (see `XYBarChartDemo2.java`)

This renderer is designed to work with an [XYPlot](#). It differs slightly from the [XYBarRenderer](#) class, in that it adjusts the position and width of each of the bars, making the assumption that the bars for all the series should be clustered within the same x-interval.

### 37.4.2 Constructors

Two constructors are defined:

```
► public ClusteredXYBarRenderer();
Equivalent to ClusteredXYBarRenderer(0.0, false)—see next constructor.
```

```
► public ClusteredXYBarRenderer(double margin, boolean centerBarAtStartValue);
Creates a new renderer. The margin (a percentage) indicates how much (if any) of each bar
should be trimmed off. The centerBarAtStartValue flag controls whether or not the cluster of
bars is shifted so that it centers around the starting x-value returned by the dataset.
```

### 37.4.3 Methods

The `drawItem()` method handles the rendering of a single item for the plot.

### 37.4.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
```

Tests this renderer for equality with an arbitrary object. Returns `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `ClusteredXYBarRenderer`;
- `obj` has the same values for the `margin` and `centerBarAtStartValue` attributes;
- `super.equals(obj)` returns `true`.

Instances of this class are `Cloneable` and `Serializable`.

### 37.4.5 Notes

Some points to note:

- this renderer casts the dataset to [IntervalXYDataset](#), so you should ensure that the plot is supplied with the correct type of data;
- a demo (`ClusteredXYBarRendererDemo1.java`) is included in the JFreeChart demo collection;
- it would probably be a good idea to merge this class with the [XYBarRenderer](#) class, but this hasn't been done yet.

#### See Also

[XYBarRenderer](#).

## 37.5 CyclicXYItemRenderer

### 37.5.1 Overview

A renderer for drawing “cyclic” charts.

#### See Also

[CyclicNumberAxis](#).

## 37.6 DefaultXYItemRenderer

### 37.6.1 Overview

This class is an alias for the [XYLineAndShapeRenderer](#) class.

## 37.7 DeviationRenderer

### 37.7.1 Overview

A subclass of [XYLineAndShapeRenderer](#) that can highlight a range of y-values in the background of a series that is typically rendered as a line—see figure 37.3 for an example. This class implements the [XYItemRenderer](#) interface, so it can be used with the [XYPlot](#) class. *This class was first introduced in JFreeChart version 1.0.5.*

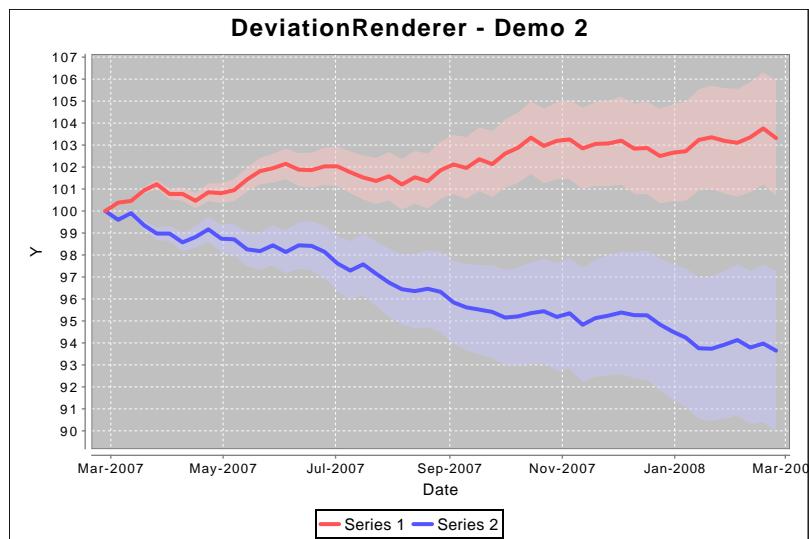


Figure 37.3: A sample chart (see `DeviationRendererDemo2.java`)

### 37.7.2 Constructor

To create a new renderer:

► `public DeviationRenderer(); [1.0.5]`

Equivalent to `DeviationRenderer(true, true)`—see the next constructor.

► `public DeviationRenderer(boolean lines, boolean shapes); [1.0.5]`

Creates a new renderer that draws lines and shapes as specified (these settings can be modified on a per-series basis using methods inherited from the [XYLineAndShapeRenderer](#) class).

### 37.7.3 General Attributes

Most attributes (the line style and color, fill color and so on) are controlled by methods inherited from [XYLineAndShapeRenderer](#).

The shading of the y-interval for each series is drawn using some transparency, which is user configurable with the following methods:

► `public float getAlpha(); [1.0.5]`

Returns the alpha transparency used when drawing the background shading for each series.

► **public void setAlpha(float alpha); [1.0.5]**

Sets the alpha transparency used when drawing the background shading for each series, and sends a `RendererChangeEvent` to all registered listeners. If `alpha` is not in the range `0.0f` to `1.0f`, this method will throw an `IllegalArgumentException`.

The (inherited) `drawSeriesLineAsPath` flag is not modifiable for this class:

► **public void setDrawSeriesLineAsPath(boolean flag); [1.0.5]**

This method is overridden to do nothing—this renderer ALWAYS draws the series line as a single path, so the flag inherited from `XYLineAndShapeRenderer` should never be changed.

### 37.7.4 Other Methods

The other methods in this renderer are typically used internally, you shouldn't need to call them directly.

The renderer makes three passes through the dataset, drawing the y-interval highlighting on the first pass, the lines on the second pass, and then drawing the shapes on the final pass. The number of passes is returned by the following method:

► **public int getPassCount(); [1.0.5]**

Returns 3.

When the renderer is initialised (at the start of every chart drawing), it creates a new state instance that will be passed to each invocation of the `drawItem()` method:

► **public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot, XYDataset dataset, PlotRenderingInfo info); [1.0.5]**

Initialises the renderer and returns a new state object.

► **protected boolean isItemPass(int pass); [1.0.5]**

Returns `true` if the specified pass is the one in which the shapes are drawn. This provides a mechanism for this renderer to control which pass the superclass uses to draw the shapes.

► **protected boolean isLinePass(int pass); [1.0.5]**

Returns `true` if these specified pass is the one in which the lines are drawn. This provides a mechanism for this renderer to control which pass the superclass uses to draw the lines.

► **public void drawItem(...); [1.0.5]**

Draws a single data item. In fact, with this implementation all the items are aggregated in the renderer state until the last item in the series is reached, then everything for that series is drawn at once.

### 37.7.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► **public boolean equals(Object obj); [1.0.5]**

Tests this renderer for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 37.7.6 Notes

Some points to note:

- this class was first introduced in JFreeChart 1.0.5;
- an `IntervalXYDataset` is required by this renderer (see `YIntervalSeriesCollection` for a useful dataset implementation);
- a couple of demos (`DeviationRendererDemo1-2.java`) are included in the JFreeChart demo collection.

## 37.8 HighLowRenderer

### 37.8.1 Overview

A *high-low renderer* draws each item in an [OHLCdataset](#) using lines to mark the “high-low” range for a trading period, plus small marks to indicate the “open” and “close” values.

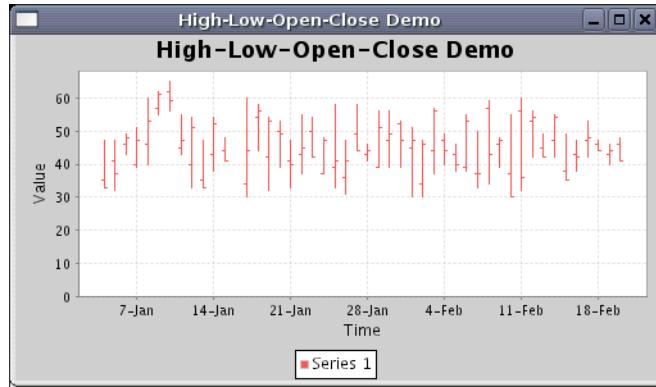


Figure 37.4: A chart that uses a *HighLowRenderer*

This renderer is designed for use with the [XYPlot](#) class. It requires an [OHLCdataset](#).

### 37.8.2 Constructors

To create a new renderer:

```
➔ public HighLowRenderer();
Creates a new renderer.
```

### 37.8.3 Methods

The renderer has flags that control whether or not the open and close ticks are drawn for each data value:

```
➔ public boolean getDrawOpenTicks();
Returns the flag that controls whether or not the open tick is drawn for each data value.

➔ public void setDrawOpenTicks(boolean draw);
Sets the flag that controls whether or not an open tick is drawn for each data value (the default value is true). A RendererChangeEvent is sent to all registered listeners.

➔ public boolean getDrawCloseTicks();
Returns the flag that controls whether or not the close tick is drawn for each data value.

➔ public void setDrawCloseTicks(boolean draw);
Sets the flag that controls whether or not a close tick is drawn for each data value (the default value is true). A RendererChangeEvent is sent to all registered listeners.
```

The paint used for the open and close ticks is the same as the series paint, but it can be overridden with the following methods:

```
➔ public Paint getOpenTickPaint();
Returns the paint (possibly null) used for the open tick mark.

➔ public void setOpenTickPaint(Paint paint);
Sets the paint used to draw the open tick mark for each data value. If this is null (the default) then the renderer's series paint is used instead.
```

```
→ public Paint getCloseTickPaint();
Returns the paint (possibly null) used for the close tick mark.

→ public void setCloseTickPaint(Paint paint);
Sets the paint used to draw the open tick mark for each data value. If this is null (the default)
then the renderer's series paint is used instead.
```

Finally, this class implements the `drawItem()` method defined in the `XYItemRenderer` interface:

```
→ public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea,
PlotRenderingInfo info, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
XYDataset dataset, int series, int item, CrosshairState crosshairState, int pass);
Draws a single item in the dataset. This method is called by the XYPlot class during chart
rendering—you won't normally call this method yourself.
```

### 37.8.4 Equals, Cloning and Serialization

This class overrides the `equals` method:

```
→ public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object. This method returns true if and only
if:


- obj is not null,
- obj is an instance of HighLowRenderer,
- obj and this renderer have the same field values.

```

Instances of this class are `Cloneable` and `Serializable`.

### 37.8.5 Notes

Some points to note:

- this renderer requires the dataset to be an instance of `OHLCDataSet`;
- the `createHighLowChart()` method in the `ChartFactory` class makes use of this renderer.

## 37.9 StackedXYAreaRenderer

### 37.9.1 Overview

A stacked area renderer that draws items from a `TableXYDataset`. An example is shown in figure 37.5.

### 37.9.2 Constructors

The following constructors are defined:

```
→ public StackedXYAreaRenderer();
Equivalent to StackedXYAreaRenderer(AREA)—see the next constructor.

→ public StackedXYAreaRenderer(int type);
Equivalent to StackedXYAreaRenderer(type, null, null)—see the next constructor.

→ public StackedXYAreaRenderer(int type, XYToolTipGenerator labelGenerator, XYURLGenerator
urlGenerator);
Creates a new renderer with the specified type, which should be one of:


- AREA – draws the area underneath the data points;
- AREA_AND_SHAPES – draws the area underneath the data points, and adds a shape at each
data point;
- SHAPES – draws shapes only;
- LINES – draws lines only;
- SHAPES_AND_LINES – draws shapes and lines.

```

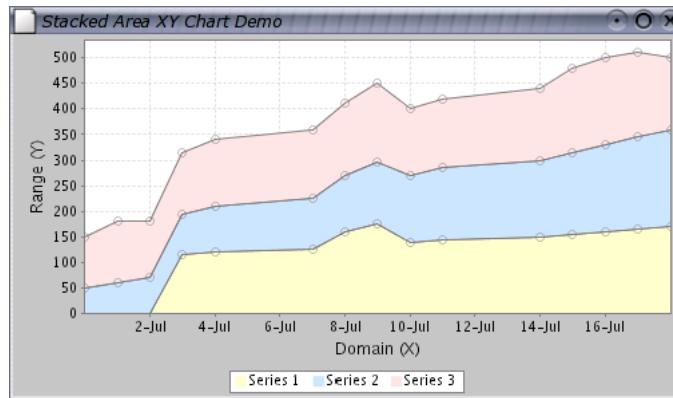


Figure 37.5: A chart created using `StackedXYAreaRenderer`

### 37.9.3 General Attributes

This class inherits most of its attributes from the `XYAreaRenderer` class.

To control the color of the shape outlines:

- ▶ `public Paint getShapePaint();`  
Returns the paint used to draw the shape outlines. The default value is `null`, which means the regular series paint is used to draw the shape outlines.
- ▶ `public void setShapePaint(Paint shapePaint);`  
Sets the paint used to draw the shape outlines, and sends a `RendererChangeEvent` to all registered listeners. If you set this attribute to `null`, the shape outlines are drawn using the series paint.

To control the stroke used to draw the shape outlines:

- ▶ `public Stroke getShapeStroke();`  
Returns the stroke used to draw the shape outlines. The default value is `null`, which means the regular series stroke is used instead.
- ▶ `public void setShapeStroke(Stroke shapeStroke);`  
Sets the stroke used to draw the shape outlines, and sends a `RendererChangeEvent` to all registered listeners. If you set this attribute to `null`, the shape outlines are drawn using the series stroke.

### 37.9.4 Other Methods

The other methods in this class are intended for use by JFreeChart—you won't normally call these methods directly:

- ▶ `public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot, XYDataset dataset, PlotRenderingInfo info);`  
Initialises the renderer and returns a state object that will be passed to each call to the `drawItem()` method.
- ▶ `public int getPassCount();`  
Returns 2, as this renderer requires two passes through the dataset.
- ▶ `public Range findRangeBounds(XYDataset dataset);`  
Returns the range that should be used on the range axis for this renderer to display all the values in the specified dataset. This takes into account that the values are stacked by this renderer.
- ▶ `public void drawItem(...);`  
Draws one item from the dataset.

### 37.9.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
    Tests this renderer for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 37.9.6 Notes

A couple of demos (`StackedXYAreaChartDemo1-2.java`) are included in the JFreeChart demo collection.

#### See Also

[StackedXYAreaRenderer2](#).

## 37.10 StackedXYAreaRenderer2

### 37.10.1 Overview

An `XYItemRenderer` that fills the area under a series, and stacks each series on top of the other—for example, see figure 37.6.

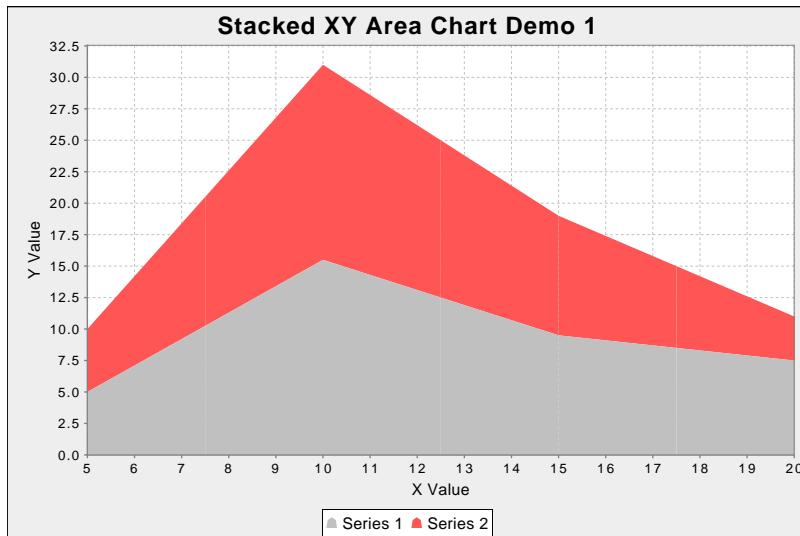


Figure 37.6: A sample chart (see `StackedXYAreaChartDemo1.java`)

The `ChartFactory` class has a `createStackedXYAreaChart()` that returns a ready-made `JFreeChart` instance that uses an instance of this renderer.

### 37.10.2 Constructors

To create a new instance:

```
➔ public StackedXYAreaRenderer2();
    Equivalent to StackedXYAreaRenderer2(null, null)—see the next constructor.

➔ public StackedXYAreaRenderer2(XYToolTipGenerator toolTipGenerator, XYURLGenerator urlGenerator);
    Creates a new renderer with the specified tool tip and URL generators (which may be null).
```

### 37.10.3 General Attributes

This class inherits most of its attributes from the [XYAreaRenderer2](#) class.

It also defines a flag that controls the rounding of x-coordinates, which can help to avoid vertical striping effects when drawing to the screen (with the sacrifice of a little accuracy):

- ▶ `public boolean getRoundXCoordinates(); [1.0.4]`  
Returns the flag that controls whether or not the x-coordinates (in drawing space) are rounded. The default value is `true`.
- ▶ `public void setRoundXCoordinates(boolean round); [1.0.4]`  
Sets the flag that controls whether or not the x-coordinates (in drawing space) are rounded, and sends a [RendererChangeEvent](#) to all registered listeners.

### 37.10.4 Other Methods

The other methods are intended for use by JFreeChart—you won’t normally call these methods yourself:

- ▶ `public int getPasswordCount();`  
Returns 1, as this renderer requires a single pass.
- ▶ `public Range findRangeBounds(XYDataset dataset);`  
Returns the range that should be used on the range axis for this renderer to display all the values in the specified dataset. This takes into account that the values are stacked by this renderer.
- ▶ `public void drawItem(...);`  
Handles the drawing of a single item from the dataset.

### 37.10.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- ▶ `public boolean equals(Object obj);`  
Tests this renderer for equality with an arbitrary object. If `obj` is `null`, this method returns `false`.

Instances of this class are `Cloneable` and `Serializable`.

### 37.10.6 Notes

Some points to note:

- a couple of demos (`StackedXYAreaChartDemo1-2.java`) are included in the JFreeChart demo collection.

#### See Also

[StackedXYAreaRenderer](#).

## 37.11 StackedXYBarRenderer

### 37.11.1 Overview

A renderer for drawing stacked bar charts on an [XYPlot](#) using data from a [TableXYDataset](#)—see figure 37.7 for an example. This class extends [XYBarRenderer](#).

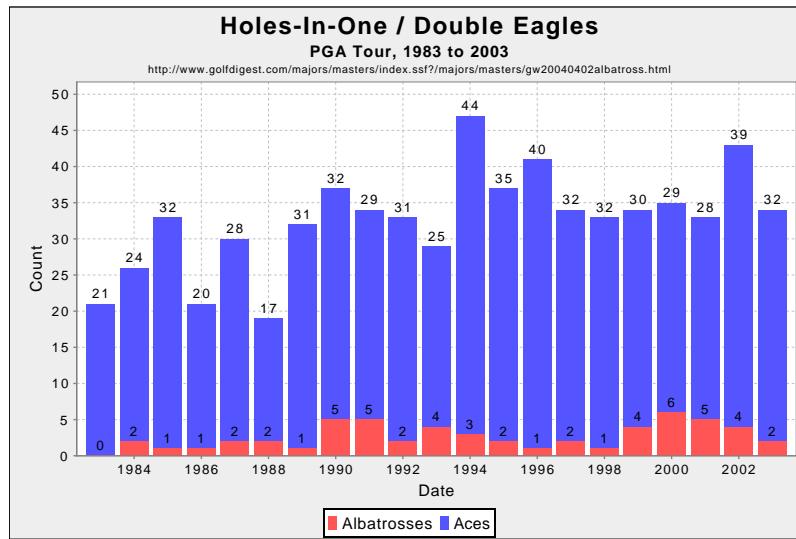


Figure 37.7: A sample chart (see `StackedXYBarChartDemo2.java`)

### 37.11.2 Constructors

There are two constructors:

- ▶ `public StackedXYBarRenderer();`  
Equivalent to `StackedXYBarRenderer(0.0)`—see the next constructor.
- ▶ `public StackedXYBarRenderer(double margin);`  
Creates a new instance with the specified `margin`. The margin is a percentage amount to trim off the width of each bar drawn by the renderer—for example, `0.10` is ten percent.

### 37.11.3 General Attributes

In addition to the attributes inherited from `XYBarRenderer`, this class defines a `renderAsPercentages` flag:

- ▶ `public boolean getRenderAsPercentages(); [1.0.5]`  
Returns the flag that controls whether the renderer displays the values as a percentage of the total. The default value is `false`.
- ▶ `public void setRenderAsPercentages(boolean asPercentages); [1.0.5]`  
Sets the flag that controls whether the renderer displays the values as percentage of the total across all series, and sends a `RendererChangeEvent` to all registered listeners.

### 37.11.4 Other Methods

The following methods are called by JFreeChart—you won't normally call these methods directly:

- ▶ `public int getPassCount();`  
Returns 2 to indicate that two passes are required by this renderer. The bars are drawn in the first pass, while the item labels (if any) are drawn in the second pass.
- ▶ `public Range findRangeBounds(XYDataset dataset);`  
Returns the range that should be set for the range axis in order to display all the values in the specified dataset (taking into account the fact that the renderer “stacks” values).
- ▶ `public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot, XYDataset data, PlotRenderingInfo info);`  
Initialises the renderer. This method is called by the `XYPlot` class, you won't normally need to call it yourself.

► public void drawItem(...);

Draws one item from the dataset. This method is called by the [XYPlot](#) class, you won't normally need to call it yourself.

### 37.11.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);

Tests this renderer for equality with an arbitrary object (`null` is permitted).

Instances of this class are `Cloneable` and `Serializable`.

### 37.11.6 Notes

Some points to note:

- this renderer requires a dataset that implements the [TableXYDataset](#) interface (which guarantees that all series share the same set of x-values, a requirement to allow values to be stacked).
- several demos ([StackedXYBarChartDemo1-3.java](#)) are included in the JFreeChart demo distribution.

#### See Also

[XYBarRenderer](#).

## 37.12 StandardXYItemRenderer

### 37.12.1 Overview

An [XYItemRenderer](#) for the [XYPlot](#) class that represents data by drawing lines between  $(x, y)$  data points. There is also a mechanism for drawing shapes or images at each at each  $(x, y)$  data point (with or without the lines).

*Special note: this renderer has been retained for historical reasons—in general, you should use the [XYLineAndShapeRenderer](#) instead.*

### 37.12.2 Constructors

To create a `StandardXYItemRenderer`:

► public StandardXYItemRenderer();

Equivalent to `StandardXYItemRenderer(LINES)`—see the next constructor.

► public StandardXYItemRenderer(int type);

Equivalent to `StandardXYItemRenderer(type, null, null)`—see next constructor.

► public StandardXYItemRenderer(int type, [XYToolTipGenerator](#) toolTipGenerator);

Equivalent to `StandardXYItemRenderer(type, toolTipGenerator, null)`—see next constructor.

► public StandardXYItemRenderer(int type, [XYToolTipGenerator](#) toolTipGenerator,

[XYURLGenerator](#) urlGenerator);

Creates a new renderer. The `type` argument should be one of: `LINE`, `SHAPE` or `SHAPES_AND_LINES`.

### 37.12.3 Lines

To control whether or not the renderer draws lines between data points:

► `public boolean getPlotLines();`

Returns the flag that controls whether or not the renderer draws lines between the data points.

The initial value is set according to the `type` argument in the constructor.

► `public void setPlotLines(boolean flag);`

Sets the flag that controls whether or not lines are plotted between data points, and sends a `RendererChangeEvent` to all registered listeners.

By default, this renderer draws lines between each pair of data items, one pair at a time. If you specify a dashed stroke for any series, this process of drawing the series as a sequence of line segments can distort (or even completely hide) the dashed appearance of the series line. One workaround for this is to set the following flag, which causes the renderer to store a complete path for the series, and draw the entire path only when the last item in the series is reached:

► `public boolean getDrawSeriesLineAsPath();`

Returns the flag that controls whether the series line is drawn as a single path, rather than a sequence of line segments.

► `public void setDrawSeriesLineAsPath(boolean flag);`

Sets the flag that controls whether the series line is drawn as a single path, and sends a `RendererChangeEvent` to all registered listeners.

This renderer can draw discontinuous lines:

► `public boolean getPlotDiscontinuous();`

Returns the flag that controls whether the discontinuous lines feature is being used. The default value is `false`.

► `public void setPlotDiscontinuous(boolean flag); [1.0.5]`

Sets the flag that controls whether or not the discontinuous lines feature should be used, and sends a `RendererChangeEvent` to all registered listeners.

The renderer compares the distance between the current and previous data points to the current gap threshold, then connects the points with a line only if the distance is below the threshold:

► `public double getGapThreshold();`

Returns the current gap threshold. The default value is `1.0`.

► `public void setGapThreshold(double t);`

Sets the gap threshold and sends a `RendererChangeEvent` to all registered listeners.

The gap threshold can be interpreted as an absolute distance along the x-axis, or a relative distance (compared to the range from the minimum x-value to the maximum x-value):

► `public UnitType getGapThresholdType();`

Returns a token that indicates whether the gap threshold is specified in absolute or relative terms. The default value is `UnitType.RELATIVE`. This method never returns `null`.

► `public void setGapThresholdType(UnitType thresholdType);`

Sets the token that indicates whether the gap threshold is specified in relative or absolute units, and sends a `RendererChangeEvent` to all registered listeners.

### 37.12.4 Shapes

This renderer can draw shapes to highlight individual data items. Each shape can be drawn as an outline, or filled—in either case, the renderer uses the regular series paint, not the outline or fill paint (which are ignored).<sup>1</sup> The shapes themselves are configurable using methods inherited from the `AbstractRenderer` class.

---

<sup>1</sup>The `XYLineAndShapeRenderer` is more flexible in this area.

## Visibility

To control the visibility of the shapes:<sup>2</sup>

► `public boolean getBaseShapesVisible();`

Returns the flag that controls whether or not the renderer draws shapes to highlight the data points. The initial value is set in the constructor according to the `type` argument.

► `public void setBaseShapesVisible(boolean flag);`

Sets the flag that controls whether or not the renderer draws shapes to highlight the data points, and sends a `RendererChangeEvent` to all registered listeners.

## Fill State

This renderer can fill shapes (using the regular series paint) or leave them empty—the following method is called for each item, to determine whether or not the shape for that item is filled:

► `public boolean getItemShapeFilled(int series, int item);`

Returns `true` if the shape for the specified data item should be filled, and `false` otherwise. By default, this method does a look-up on the base, per-series and override flags that you can set using other methods defined by this renderer—see below.

To control the default setting for the shape

► `public boolean getBaseShapesFilled();`

Returns the default flag that controls whether or not shapes are filled. This setting is used when there is no per-series or override flag set.

► `public void setBaseShapesFilled(boolean flag);`

Sets the default flag that controls whether or not shapes are filled, then sends a `RendererChangeEvent` to all registered listeners.

You can specify the shape fill status on a per-series basis:

► `public boolean getSeriesShapesFilled(int series);`

Returns a flag that determines whether or not the shapes for the given `series` are filled. This.

► `public void setSeriesShapesFilled(int series, Boolean flag);`

Sets the flag that determines whether or not the shape for the specified `series` are filled, then sends a `RendererChangeEvent` to all registered listeners.

An override flag can be used to control the fill status for all series:

► `public Boolean getShapesFilled(); [1.0.5]`

Returns the override flag that controls whether or not shapes are filled for all series. The default value is `null` (which means that the renderer looks at the per-series and base settings instead).

► `public void setShapesFilled(boolean filled);`

Equivalent to `setShapesFilled(Boolean.valueOf(filled))`—see the next method.

► `public void setShapesFilled(Boolean filled);`

Sets the override flag that controls whether or not shapes are filled for ALL series, and sends a `RendererChangeEvent` to all registered listeners. In general, you'll want to leave this set to `null`, so that the per-series or base settings apply.

## 37.12.5 Images

This renderer is capable of displaying small images at each data point, but only if you subclass the renderer and override the `getImage()` method, and also set the following flag:

► `public boolean getPlotImages();`

Returns the flag that controls whether the renderer draws an image to represent each data point.

---

<sup>2</sup>Attributes to control shape visibility on a per-series basis have never been added to this renderer, which is one reason for recommending the use of `XYLineAndShapeRenderer` ahead of this class.

```

→ public void setPlotImages(boolean flag);
Sets the flag that controls whether or not the renderer draws an image to represent each data
point, and sends a RendererChangeEvent to all registered listeners.

→ protected Image getImage(Plot plot, int series, int item, double x, double y);
Returns null always. Override this method and return an appropriate image for the specified
item, if you would like this renderer to display an image for each data point.

→ protected Point getImageHotspot(Plot plot, int series, int item, double x, double y,
Image image);
Returns the point within the image that is the “hot spot”.

```

### 37.12.6 Legend Items

To create a legend item for a series, JFreeChart will call the following method:

```

→ public LegendItem getLegendItem(int datasetIndex, int series);
Returns a legend item for the specified series that incorporates the shape returned by getLegend-
Line().

```

To control the small graphical item used to represent a line in the legend:

```

→ public Shape getLegendLine();
Returns the shape used to represent lines in the legend. The default value is Line2D.Double(-7.0,
0.0, 7.0, 0.0). This method never returns null.

→ public void setLegendLine(Shape line);
Sets the shape used to represent lines in the legend and sends a RendererChangeEvent to all
registered listeners. If line is null, this method throws an IllegalArgumentException. If you
supply a custom shape via this method, it should be centered about (0, 0) so that it can be
positioned accurately within the legend.

```

### 37.12.7 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```

→ public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object.

```

Instances of this class are `Cloneable` (`PublicCloneable`) and `Serializable`.

### 37.12.8 Notes

Some points to note:

- this class implements the `XYItemRenderer` interface;
- in general you should avoid using this renderer, and use `XYLineAndShapeRenderer` instead.

## 37.13 VectorRenderer

### 37.13.1 Overview

A renderer that displays vector data from a `VectorXYDataset`—see figure 37.8 for an example. This class implements the `XYItemRenderer` interface, so it can be used with the `XYPlot` class. **This class was first introduced in JFreeChart version 1.0.6.**

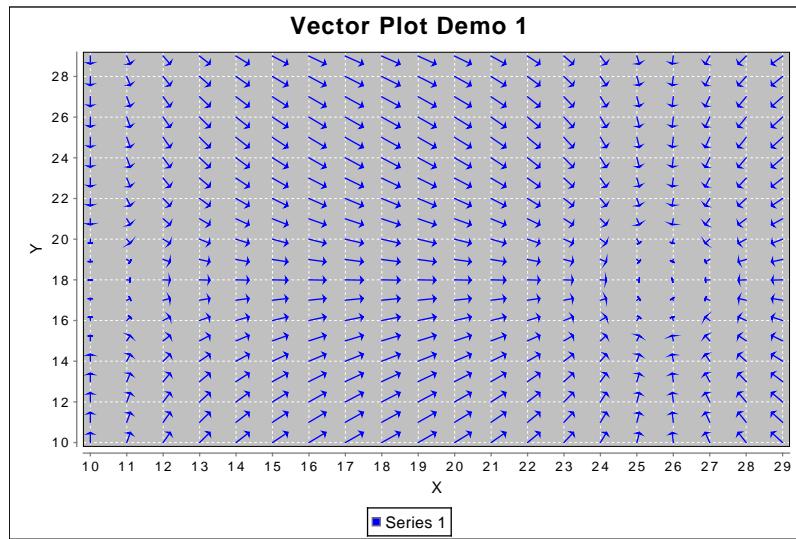
### 37.13.2 Constructor

To create a new renderer:

```

→ public VectorRenderer(); [1.0.6]
Creates a new renderer.

```



*Figure 37.8: A sample chart (see `VectorRendererDemo1.java`)*

### 37.13.3 Methods

The following methods are overridden to adjust the data bounds to take into account the length of the vectors:

```
➔ public Range findDomainBounds(XYDataset dataset);
    Returns the range required to display all the x-values in the dataset. This method is typically
    called by JFreeChart—you shouldn't need to call this method yourself.

➔ public Range findRangeBounds(XYDataset dataset);
    Returns the range required to display all the y-values in the dataset. This method is typically
    called by JFreeChart—you shouldn't need to call this method yourself.
```

The draw item method is overridden to implement the custom rendering performed by this class:

```
➔ public void drawItem(...);
    Draws a single item from the dataset. The XYPlot class will call this method as necessary—you
    don't need to call this method directly.
```

### 37.13.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj); [1.0.5]
    Tests this renderer for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 37.13.5 Notes

Some points to note:

- this class was first introduced in JFreeChart 1.0.6;
- a `VectorXYDataset` is required by this renderer (see `VectorSeriesCollection` for a useful dataset implementation);
- a demo (`VectorRendererDemo1.java`) is included in the JFreeChart demo collection.

## 37.14 WindItemRenderer

### 37.14.1 Overview

A renderer that [XYPlot](#) uses to draw wind plots.

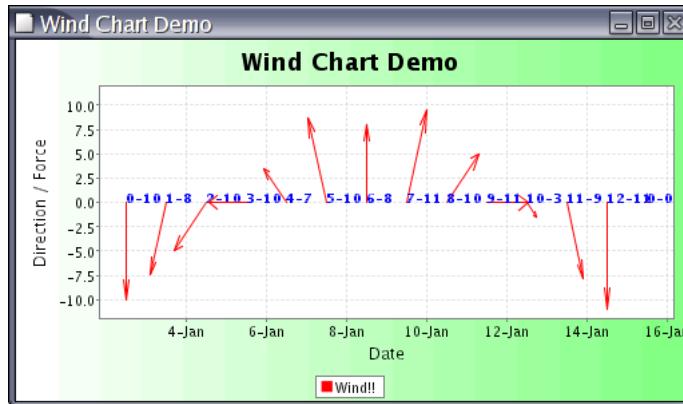


Figure 37.9: A sample chart using *WindItemRenderer*

### 37.14.2 Notes

Some points to note:

- this renderer requires a [WindDataset](#) for the data;
- a demo (`WindChartDemo1.java`) is included in the JFreeChart demo collection.

## 37.15 XYAreaRenderer

### 37.15.1 Overview

A renderer draws each item in an [XYDataset](#) using a polygon that fills the area between the x-axis and the data point—see figure 37.10 for an example. This renderer is designed to be used with the [XYPlot](#) class.

### 37.15.2 Constructors

The following constructors are defined:

```
► public XYAreaRenderer();
```

Equivalent to `XYAreaRenderer(AREA)`—see the next constructor.

```
► public XYAreaRenderer(int type);
```

Creates a new `XYAreaRenderer` using one of the following types: `SHAPES`, `LINE`s, `SHAPES_AND_LINES`, `AREA`, `AREA_AND_SHAPES`.

```
► public XYAreaRenderer(int type, XYToolTipGenerator toolTipGenerator,
XYURLGenerator urlGenerator);
```

Creates a new renderer with the specified tool tip generator and URL generator (either may be null).

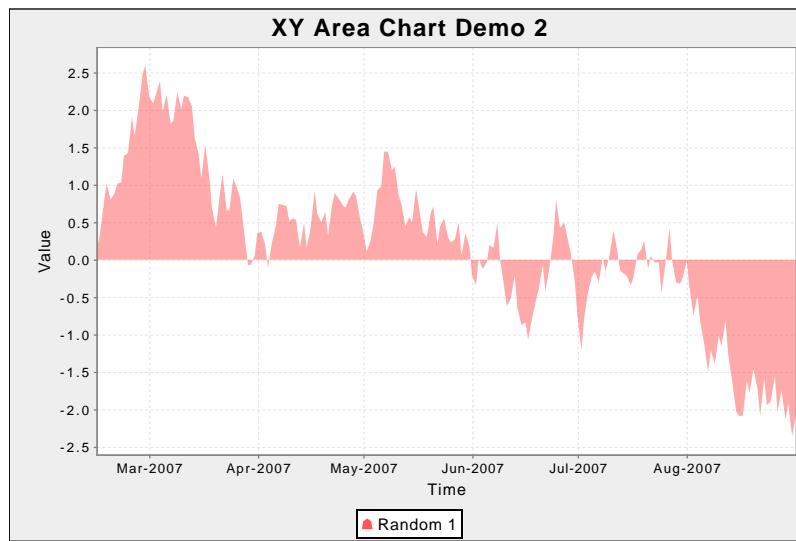


Figure 37.10: A chart using `XYAreaRenderer` (see `XYStepAreaRendererDemo1.java`)

### 37.15.3 General Attributes

Several flags control the rendering process. These flags are initialised in the constructor, and cannot be updated without creating a new renderer:

- `public boolean getPlotShapes();`  
Returns the flag that controls whether or not shapes are drawn at each data point.
- `public boolean getPlotLines();`  
Returns the flag that controls whether or not lines are drawn between each data point.
- `public boolean getPlotArea();`  
Returns a flag that controls whether or not the area is being filled for each series.

A flag controls whether or not outlines are drawn for the area representing each series:

- `public boolean isOutline();`  
Returns the flag that controls whether or not outlines are drawn.
- `public void setOutline(boolean show);`  
Sets the flag that controls whether or not outlines are drawn and sends a `RendererChangeEvent` to all registered listeners.

### 37.15.4 Legend Items

This renderer provides an option to modify the small graphic displayed in each legend item:

- `public Shape getLegendArea();`  
Returns the shape used for legend items (never `null`).
- `public void setLegendArea(Shape area);`  
Sets the shape used for legend items and sends a `RendererChangeEvent` to all registered listeners.  
If `area` is `null`, this method throws an `IllegalArgumentException`.

JFreeChart will call the following method to obtain a legend item for each series that is handled by this renderer:

- `public LegendItem getLegendItem(int datasetIndex, int series);`  
Returns a legend item for a series within a particular dataset. This method is implemented to use `getLegendArea()` for the small graphic displayed in the legend.

### 37.15.5 Other Methods

To initialise the renderer (this method is called by the plot, you won't normally need to call it yourself):

```
➔ public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea,
XYPlot plot, XYDataset data, PlotRenderingInfo info);
```

Initialises the renderer. The plot will call this method at the start of the drawing process, each time a chart is drawn.

```
➔ public void drawItem(Graphics2D g2, XYItemRendererState state,
Rectangle2D dataArea, PlotRenderingInfo info, XYPlot plot,
ValueAxis domainAxis, ValueAxis rangeAxis, XYDataset dataset,
int series, int item, CrosshairState crosshairState, int pass);
Draws a single item (this method is called by the plot).
```

### 37.15.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
```

Tests this renderer for equality with an arbitrary object. This method returns `true` if:

- `obj` is not `null`;
- `obj` is an instance of `XYAreaRenderer`;
- `obj` has the same attributes as this renderer.

Instances of this class are `Cloneable` and `Serializable`.

### 37.15.7 Notes

Some points to note:

- this class extends `AbstractXYItemRenderer`;
- for stacked area charts, use the `StackedXYAreaRenderer` class;
- this class uses code copied from the `StandardXYItemRenderer` class, and some additional work is required to eliminate the duplication;
- a couple of demos (`XYAreaChartDemo1-2.java`) are included in the JFreeChart demo collection.

#### See Also

[AreaRenderer](#), [StackedXYAreaRenderer](#).

## 37.16 XYBarRenderer

### 37.16.1 Overview

This renderer can be used within an `XYPlot` to draw bar charts with data from an `IntervalXYDataset`—see figure 37.11 for an example.

### 37.16.2 Constructors

To create a new instance:

```
➔ public XYBarRenderer();
```

Creates a new renderer. The margin defaults to 0.0 (see the next constructor).

```
➔ public XYBarRenderer(double margin);
```

Creates a new renderer with the specified margin (which is expressed as a percentage, for example 0.10 is ten percent).

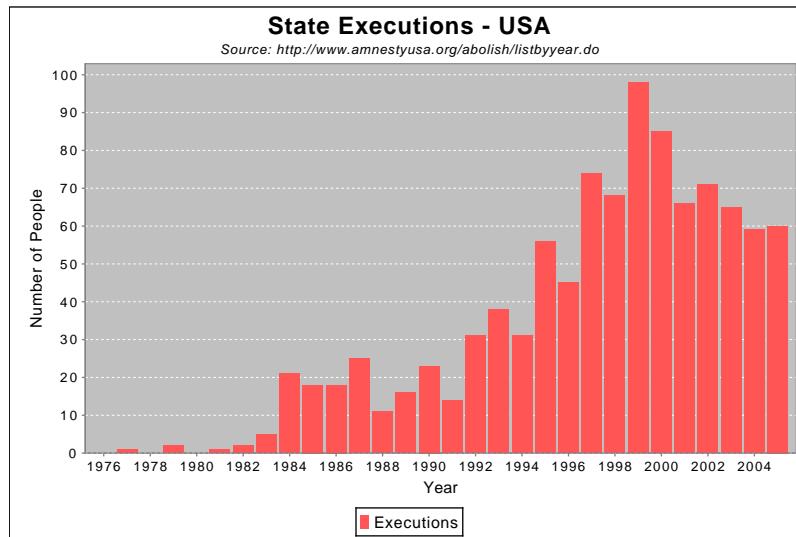


Figure 37.11: A sample chart (see XYBarChartDemo1.java)

### 37.16.3 General Attributes

To control the “margin” for the renderer:

► `public double getMargin();`

Returns the margin used by the renderer, as a percentage of the bar width (for example, 0.10 is ten percent).

► `public void setMargin(double margin);`

Sets the margin for the renderer and sends a `RendererChangeEvent` to all registered listeners. The margin is specified as a percentage of the bar width (for example, 0.10 is ten percent) and is the amount that is trimmed from the bar width before the bar is displayed.

To control whether or not outlines are drawn for each bar:

► `public boolean isDrawBarOutline();`

Returns a flag that controls whether or not bar outlines are drawn. The default value is `false`.

► `public void setDrawBarOutline(boolean draw);`

Sets a flag that controls whether or not bar outlines are drawn, and sends a `RendererChangeEvent` to all registered listeners.

To control the way that the length of the bars is determined:

► `public double getBase();`

Returns the base value for the bars (usually 0.0, but you can set it to any value).

► `public void setBase(double base);`

Sets the base value for the bars (defaults to 0.0). This setting is ignored if the `getUseYInterval()` method returns `true`.

► `public boolean getUseYInterval();`

Returns a flag that controls how the length of the bars is determined.

► `public void setUseYInterval(boolean use);`

Sets a flag that controls how the length of the bars is determined. If `true`, the y-interval from the dataset is used. If `false`, the y-value from the dataset determines one end of the bar, and the `getBase()` method determines the other end of the bar.

### 37.16.4 Gradient Paint Support

This renderer supports the use of `GradientPaint` for any series colour by using a transformer:

```
→ public GradientPaintTransformer getGradientPaintTransformer();
Returns the transformer used for GradientPaint instances. This method can return null, in
which case any GradientPaint instance will be used without being transformed.

→ public void setGradientPaintTransformer(GradientPaintTransformer transformer);
Sets the transformer used for GradientPaint instances and sends a RendererChangeEvent to all
registered listeners. If transformer is null, the renderer will use any GradientPaint instance
without transformation.
```

### 37.16.5 Other Methods

The following two methods are usually called by the `XYPlot`, you shouldn't need to call them directly:

```
→ public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot,
XYDataset dataset, PlotRenderingInfo info);
Initialises the renderer for drawing a chart.

→ public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea,
PlotRenderingInfo info, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
XYDataset dataset, int series, int item, CrosshairState crosshairState, int pass);
Draws one item from the dataset.
```

### 37.16.6 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

```
→ public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object (null is permitted). This method
returns true if and only if:


- obj is not null;
- obj is an instance of XYBarRenderer;
- both renderers have the same attributes (excluding the registered listeners).

```

Instances of this class are `Cloneable` and `Serializable`.

### 37.16.7 Notes

Some points to note:

- this renderer casts the dataset to `IntervalXYDataset`, so you should ensure that the plot is supplied with the correct type of data.

#### See Also

[ClusteredXYBarRenderer](#).

## 37.17 XYBlockRenderer

### 37.17.1 Overview

A renderer that draws coloured (or gray-scale) blocks to represent the z-values from an `XYZDataset`. For example see figure 37.12. The z-values are converted to colours using a `PaintScale`. This renderer was first introduced in JFreeChart version 1.0.4.

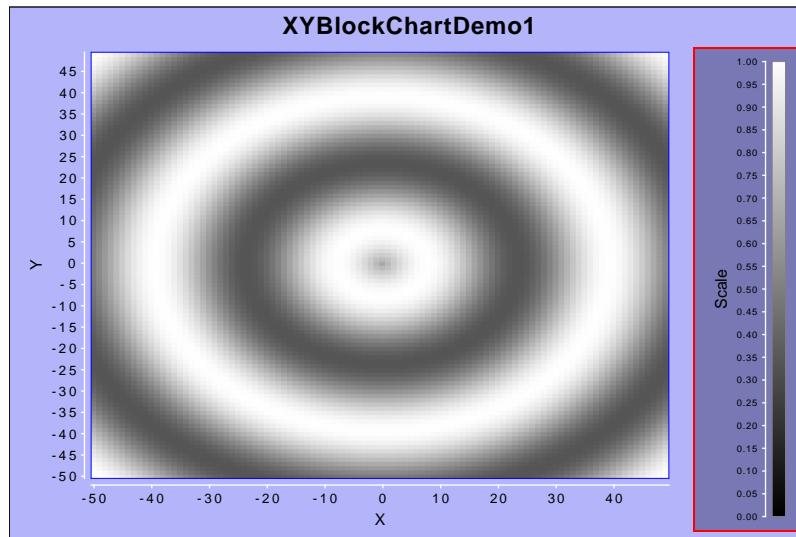


Figure 37.12: A sample chart (see `XYBlockChartDemo1.java`)

### 37.17.2 Constructors

To create a new renderer:

► `public XYBlockRenderer(); [1.0.4]`

Creates a new renderer with the following defaults:

- `blockWidth = 1.0;`
- `blockHeight = 1.0;`
- `blockAnchor = RectangleAnchor.CENTER;`
- `paintScale = new LookupPaintScale();.`

### 37.17.3 Attributes

To control the width and height of the blocks drawn by this renderer:

► `public double getBlockWidth(); [1.0.4]`

Returns the block width in data units (that is, measured against the domain axis). The default value is 1.0.

► `public void setBlockWidth(double width); [1.0.4]`

Sets the block width and sends a `RendererChangeEvent` to all registered listeners. If `width` is less than or equal to zero, this method throws an `IllegalArgumentException`.

► `public double getBlockHeight(); [1.0.4]`

Returns the block height in data units (that is, measured against the range axis). The default value is 1.0.

► `public void setBlockHeight(double height); [1.0.4]`

Sets the block height and sends a `RendererChangeEvent` to all registered listeners. If `height` is less than or equal to zero, this method throws an `IllegalArgumentException`.

Each block drawn by the renderer is aligned to its  $(x, y)$  location using an anchor point:

► `public RectangleAnchor getBlockAnchor(); [1.0.4]`

Returns the anchor point on the block that will be aligned to the  $(x, y)$  location on the plot. The default value is `RectangleAnchor.CENTER`.

```
→ public void setBlockAnchor(RectangleAnchor anchor); [1.0.4]
Sets the anchor point and sends a RendererChangeEvent to all registered listeners. If anchor is null, this method throws an IllegalArgumentException.
```

To access the paint scale used by the renderer:

```
→ public PaintScale getPaintScale(); [1.0.4]
Returns the paint scale used by this renderer (never null).

→ public void setPaintScale(PaintScale scale); [1.0.4]
Sets the paint scale used by this renderer and sends a RendererChangeEvent to all registered listeners. If scale is null, this method throws an IllegalArgumentException.
```

### 37.17.4 Other Methods

The axis ranges required by this renderer are slightly greater than the default ranges calculated on the basis of the x-values and y-value alone, because the renderer draws blocks at each data point. The following method overrides take this into account:

```
→ public Range findDomainBounds(XYDataset dataset); [1.0.4]
Returns the domain axis range required to display all the data in the specified dataset.

→ public Range findRangeBounds(XYDataset dataset); [1.0.4]
Returns the range axis range required to display all the data in the specified dataset.
```

Each item is drawn with a call to the following method (JFreeChart calls this method, you don't have to):

```
→ public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea,
PlotRenderingInfo info, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
XYZDataset dataset, int series, int item, CrosshairState crosshairState,
int pass); [1.0.4]
Draws the block for one item in the dataset.
```

### 37.17.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj); [1.0.4]
Tests this renderer for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 37.17.6 Notes

Some points to note:

- this renderer works with an `XYPlot`;
- the dataset must be an instance of `XYZDataset`, because the block colours are determined by the z-value in the dataset;
- several demos (`XYBlockChartDemo1-3.java`) are included in the JFreeChart demo collection.

#### See Also

[XYZDataset](#).

## 37.18 XYBoxAndWhiskerRenderer

### 37.18.1 Overview

A renderer that is used to create a box-and-whisker chart using data from an `BoxAndWhiskerXYDataset`. A sample chart is shown in Figure 37.13.

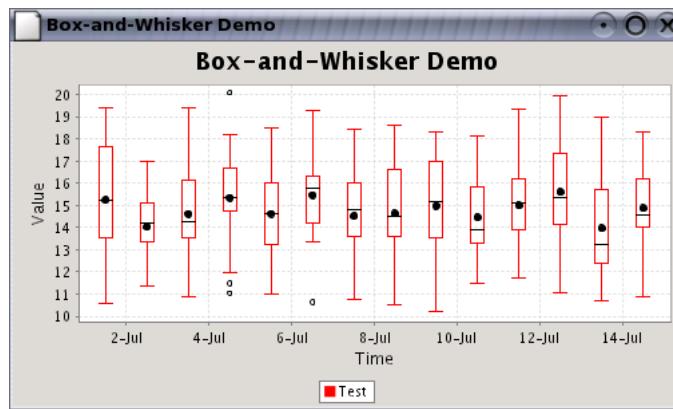


Figure 37.13: A chart generated with an `XYBoxAndWhiskerRenderer`.

### 37.18.2 Constructors

To create a new renderer:

- `public XYBoxAndWhiskerRenderer();`  
Creates a new renderer where the box width is calculated automatically.
- `public XYBoxAndWhiskerRenderer(double boxWidth);`  
Creates a new renderer with the specified box width.

### 37.18.3 Notes

Some points to note:

- for tool tips, you can use the `BoxAndWhiskerXYToolTipGenerator` class;
- there is a demo (`XYBoxAndWhiskerDemo1.java`) included in the JFreeChart demo collection.

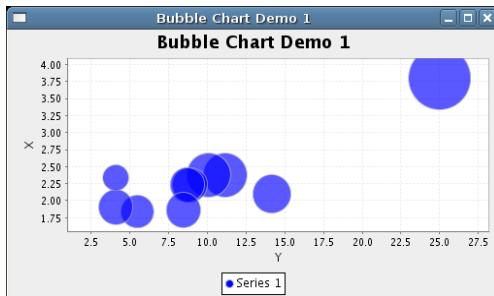
#### See Also

[BoxAndWhiskerRenderer](#).

## 37.19 XYBubbleRenderer

### 37.19.1 Overview

An `XYBubbleRenderer` displays items from an `XYZDataset` by drawing a bubble at each  $(x, y)$  point. The size (diameter) of the bubble is determined by the z-value from the dataset.



### 37.19.2 Constructors

The following constructors are defined:

```
↳ public XYBubbleRenderer();
```

Equivalent to `XYBubbleRenderer(SCALE_ON_BOTH_AXES)`—see the next constructor.

```
↳ public XYBubbleRenderer(int scaleType);
```

Creates a new renderer with the specified `scaleType`, which must be one of the following integer constants defined by this class:

- `SCALE_ON_BOTH_AXES`;
- `SCALE_ON_DOMAIN_AXIS`;
- `SCALE_ON_RANGE_AXIS`.

The width and height of the bubble is calculated from the dataset's z-value scaled against the specified axis.

### 37.19.3 Attributes

The `scaleType` attribute determines how the bubble is scaled relative to the domain and range axes:

```
↳ public int getScaleType();
```

Returns the method used to determine the size of the bubbles drawn by this renderer:

- `SCALE_ON_BOTH_AXES` – bubbles are drawn as ellipses where the width and height of the ellipse is determined by the z-value scaled against both axes;
- `SCALE_ON_DOMAIN_AXIS` – bubbles are drawn as circles where the diameter of the circle is determined by the z-value scaled against the domain axis;
- `SCALE_ON_RANGE_AXIS` – bubbles are drawn as circles where the diameter of the circle is determined by the z-value scaled against the range axis.

### 37.19.4 Other Methods

The following methods are called by JFreeChart—you won't normally call them directly.

```
↳ public LegendItem getLegendItem(int datasetIndex, int series);
```

Returns a legend item for the specified series. This method is overridden so that the legend item has a circle for the legend graphic.

```
↳ public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea,
PlotRenderingInfo info, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
XYDataset dataset, int series, int item, CrosshairState crosshairState, int pass);
```

Draws a bubble representing one item from the dataset.

### 37.19.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
↳ public boolean equals(Object obj);
```

Tests this renderer for equality with an arbitrary object. This method returns `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `XYBubbleRenderer`;
- `obj` has the same attributes as this renderer.

Instances of this class are `Cloneable` and `Serializable`.

### 37.19.6 Notes

Some notes:

- this class implements the `XYItemRenderer` interface and extends the `AbstractXYItemRenderer` class.
- a demo application (`BubbleChartDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[XYZDataset](#).

## 37.20 XYDifferenceRenderer

### 37.20.1 Overview

A renderer that highlights the difference between the items in two series by filling in the area between the lines for each series. The fill color alternates between a “positive” color (used when series 1 is greater than series 2) and a “negative” color (used when series 1 is less than series 2). Figure 37.14 shows an example.

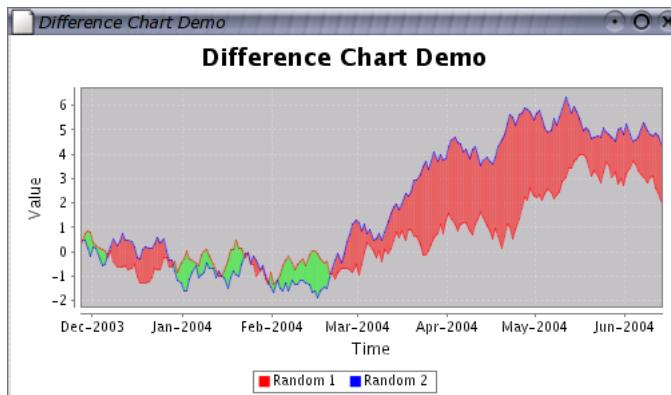


Figure 37.14: A chart generated with an `XYDifferenceRenderer`.

### 37.20.2 Usage

This renderer is designed for use with the `XYPlot` class. It expects an `XYDataset` that has exactly two series, with both series having the same set of x-values. The renderer does not handle `null` values.

There are two demos available: `DifferenceChartDemo1.java` and `DifferenceChartDemo2.java`.

### 37.20.3 Constructors

To create a new renderer:

```
► public XYDifferenceRenderer();
Creates a new renderer instance that uses Color.green for the positive paint, Color.red for the negative paint, and does not display shapes at each data point.

► public XYDifferenceRenderer(Paint positivePaint, Paint negativePaint,
boolean shapes);
Creates a new renderer instance with the given (non-null) colors. The shapes argument controls whether or not the renderer displays shapes at each point.
```

### 37.20.4 Accessor Methods

The following methods for accessing the attributes defined by this renderer (in addition to those inherited from `AbstractXYItemRenderer`):

- ▶ `public Paint getPositivePaint();`  
Returns the paint used to fill the area between series 1 and series 2 when the difference is positive (that is, the y-value in series 1 is greater than the corresponding y-value in series 2).
- ▶ `public void setPositivePaint(Paint paint);`  
Sets the paint used to fill the area between series 1 and series 2 when the difference is positive, and sends a `RendererChangeEvent` to all registered listeners.
- ▶ `public Paint getNegativePaint();`  
Returns the paint used to fill the area between series 1 and series 2 when the difference is negative (that is, the y-value in series 1 is less than the corresponding y-value in series 2).
- ▶ `public void setNegativePaint(Paint paint);`  
Sets the paint used to fill the area between series 1 and series 2 when the difference is negative, and sends a `RendererChangeEvent` to all registered listeners.
- ▶ `public boolean getShapesVisible();`  
Returns the flag that controls whether or not the renderer displays shapes at each data point.
- ▶ `public void setShapesVisible(boolean flag);`  
Sets the flag that controls whether or not the renderer displays shapes at each data point, and sends a `RendererChangeEvent` to all registered listeners.
- ▶ `public int getPassCount();`  
Returns 2, the number of passes required by this renderer to draw the data items. In the first pass, the “difference” area between the two series is filled with the specified colors. In the second pass, the series lines and item shapes are drawn.

A flag can be set to perform rounding on the x-coordinates to improve on-screen rendering:

- ▶ `public boolean getRoundXCoordinates(); [1.0.4]`  
Returns a flag that controls whether or not the x-coordinates are rounded to integers, which can prevent “striping” for charts displayed on-screen. The default value is `false`.
- ▶ `public void setRoundXCoordinates(boolean round); [1.0.4]`  
Sets the flag that controls whether or not the x-coordinates are rounded to integers and sends a `RendererChangeEvent` to all registered listeners.

As mentioned, the methods that set an attribute will send a `RendererChangeEvent` to all registered listeners. This will usually trigger a chain of events that will lead to the chart itself being repainted, if necessary.

### 37.20.5 Rendering Methods

The following methods are called by the `XYPlot`, you shouldn't need to call them directly:

- ▶ `public XYItemRendererState initialise(Graphics2D g2,  
Rectangle2D dataArea, XYPlot plot, XYDataset data, PlotRenderingInfo info);`  
Initialises the renderer.
- ▶ `public void drawItem(Graphics2D g2, XYItemRendererState state,  
Rectangle2D dataArea, PlotRenderingInfo info, XYPlot plot,  
ValueAxis domainAxis, ValueAxis rangeAxis, XYDataset dataset,  
int series, int item, CrosshairState crosshairState, int pass);`  
Draws an item—this method will be called for each item in the dataset.

### 37.20.6 Equals, Cloning and Serialization

This renderer overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
    Tests the renderer for equality with obj (which may be null).
```

This renderer can be cloned:

```
➔ public Object clone() throws CloneNotSupportedException;
    Returns a clone of the renderer. In typical usage, the specified exception will not be thrown,
    however it is possible to trigger the exception if some attribute of the renderer is not cloneable.
```

This renderer is serializable.

## 37.21 XYDotRenderer

### 37.21.1 Overview

A renderer that can be used by an `XYPlot` to display items from an `XYDataset`. The renderer fills a rectangle (a single pixel by default) at each  $(x, y)$  point—see figure 37.15 for an example.

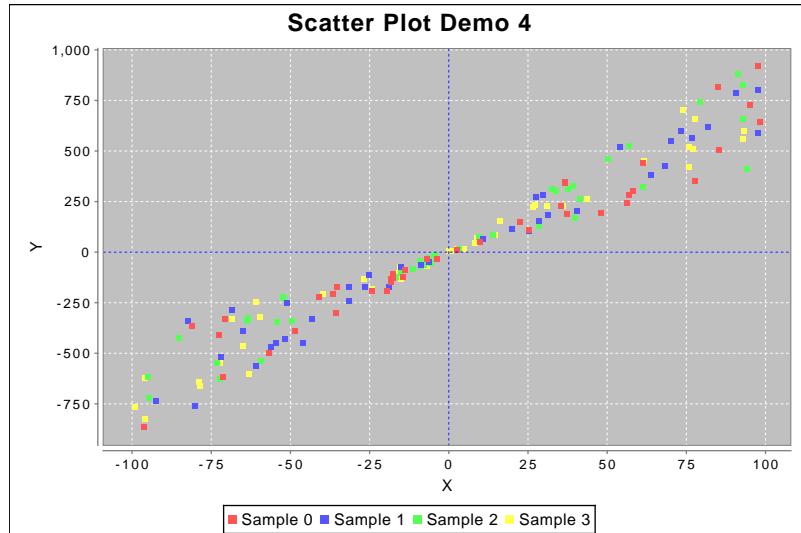


Figure 37.15: A sample chart (see `ScatterPlotDemo4.java`)

This renderer offers better performance (but less flexibility) than the `XYLineAndShapeRenderer` class, because it simply fills a rectangle for each data item, rather than filling and drawing a shape.

### 37.21.2 Constructor

The default constructor is the only constructor available:

```
➔ public XYDotRenderer();
    Creates a new renderer with the default dot size of 1 unit by 1 unit.
```

### 37.21.3 Methods

Accessor methods are provided for the “dot” (actually a rectangle) width and height:

```
➔ public int getDotWidth(); [1.0.2]
    Returns the current dot width. The default value is 1.
```

► public void setDotWidth(int w); [1.0.2]

Sets the dot width and sends a [RendererChangeEvent](#) to all registered listeners.

► public int getDotHeight(); [1.0.2]

Returns the current dot height. The default value is 1.

► public void setDotHeight(int h); [1.0.2]

Sets the dot height and sends a [RendererChangeEvent](#) to all registered listeners.

To control the shape used in the legend for each series:

► public Shape getLegendShape(); [1.0.7]

Returns the shape (never `null`) used in the legend to represent the series. The default value is `Rectangle2D.Double(-3.0, -3.0, 6.0, 6.0)`.

► public void setLegendShape(Shape shape); [1.0.7]

Sets the shape used in the legend to represent each series. If `shape` is `null`, this method throws an [IllegalArgumentException](#).

This class implements the `drawItem()` method defined in the [XYItemRenderer](#) interface. This method is usually called by the plot, you don't need to call it yourself. Many other methods are inherited from the [AbstractXYItemRenderer](#) base class.

### 37.21.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);

Tests this renderer for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`, so that chart's using this type of renderer are cloneable and serializable. This renderer also implements the [PublicCloneable](#) interface.

### 37.21.5 Notes

Some points to note:

- tooltips, item labels and URLs are NOT generated by this renderer (these features may be added in a future release);
- a demo application (`ScatterPlotDemo4.java`) is included in the JFreeChart demo collection.

#### See Also

[XYItemRenderer](#).

## 37.22 XYErrorRenderer

### 37.22.1 Overview

A renderer that extends [XYLineAndShapeRenderer](#) to display error bars about each data item. This renderer is designed to be used with an [XYPlot](#) to display items from an [IntervalXYDataset](#)—see figure 37.16 for an example. This class implements the [XYItemRenderer](#) interface, and was first added to JFreeChart at version 1.0.3.

### 37.22.2 Constructor

The default constructor is the only constructor available:

► public XYErrorRenderer();

Creates a new renderer. By default, error bars are drawn for both the x-values and the y-values, using the series paint.

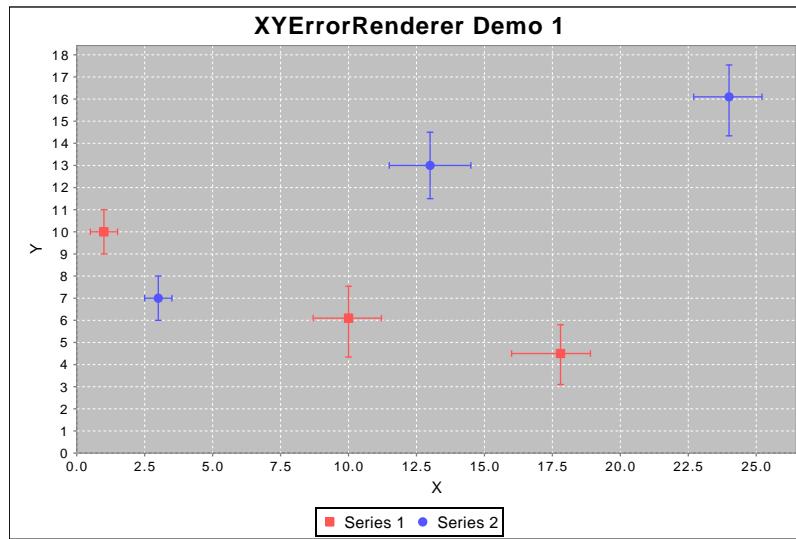


Figure 37.16: A sample chart (see `XYErrorRendererDemo1.java`)

### 37.22.3 General Attributes

To control whether or not error bars are drawn for the x-values:

```
→ public boolean getDrawXError();
Returns the flag that controls whether or not error bars are drawn for the x-values in the dataset. The default value is true.

→ public void setDrawXError(boolean draw);
Sets the flag that controls whether or not error bars are drawn for the x-values in the dataset, and sends a RendererChangeEvent to all registered listeners.
```

To control whether or not error bars are drawn for the y-values:

```
→ public boolean getDrawYError();
Returns the flag that controls whether or not error bars are drawn for the y-values in the dataset. The default value is true.

→ public void setDrawYError(boolean draw);
Sets the flag that controls whether or not error bars are drawn for the y-values in the dataset, and sends a RendererChangeEvent to all registered listeners.
```

To control the length of the caps at the end of the error bars:

```
→ public double getCapLength();
Returns the length of the caps at each end of the error bar (in Java2D units). The default value is 4.0.

→ public void setCapLength(double length);
Sets the length of the caps at each end of the error bar for each data value and sends a RendererChangeEvent to all registered listeners. The cap length is specified in Java2D units.
```

By default, the renderer draws error bars using the series paint. You can override this with the following methods:

```
→ public Paint getErrorPaint();
Returns the paint used to draw the error bars, or null if the renderer should use the series paint. The default value is null.

→ public void setErrorPaint(Paint paint);
Sets the paint used to draw the error bars and sends a RendererChangeEvent to all registered listeners. If you set this attribute to null, the error bars will be drawn using the series paint.
```

### 37.22.4 Other Methods

The following methods are typically called by JFreeChart—you won’t normally call them directly:

► public **Range** findDomainBounds(**XYDataset** dataset);

Returns the range required by this renderer to display all of the domain values in the dataset. If **dataset** is **null**, this method returns **null**. This method is overridden to include the x-interval when the dataset is an instance of **IntervalXYDataset**.

► public **Range** findRangeBounds(**XYDataset** dataset);

Returns the range required by this renderer to display all of the range values in the dataset. If **dataset** is **null**, this method returns **null**. This method is overridden to include the y-interval when the dataset is an instance of **IntervalXYDataset**.

### 37.22.5 Equals, Cloning and Serialization

This class overrides the **equals()** method:

► public boolean equals(Object obj);

Returns **true** if **obj** is equal to this instance.

Instances of this class are **Cloneable** and **Serializable**.

### 37.22.6 Notes

Some points to note:

- this renderer *requires* the dataset to be an instance of **IntervalXYDataset** in order to draw the error bars—in the event that the dataset is just a regular **XYDataset**, this renderer falls back to the behaviour of the **XYLineAndShapeRenderer** class;
- a couple of demos (**XYErrorRendererDemo1-2.java**) are included in the JFreeChart demo collection;
- this class was added to JFreeChart at version 1.0.3.

#### See Also

[XYItemRenderer](#).

## 37.23 XYItemRenderer

### 37.23.1 Overview

An *XY item renderer* is a plug-in class that works with an **XYPlot** and assumes responsibility for drawing individual data items in a chart. This interface defines the methods that every renderer must support.

A range of different renderers are supplied in the JFreeChart distribution. Figure 37.17 shows the class hierarchy.

As well as drawing the visual representation of a data item, the renderer is also responsible for generating tooltips (for charts displayed in a **ChartPanel**) and URL references for charts displayed in an HTML image map.

A summary of the available renderers is given in Table 37.1.

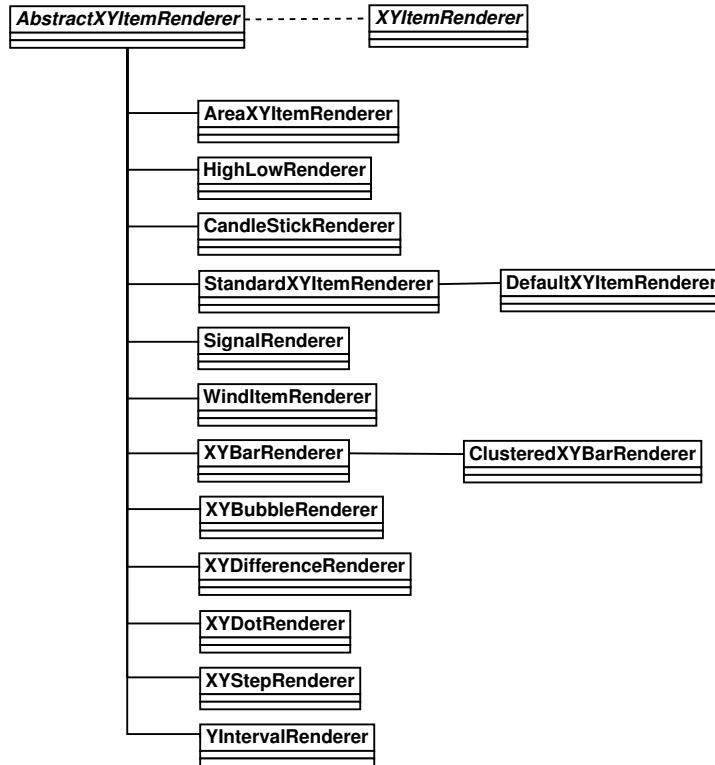


Figure 37.17: Renderer hierarchy

### 37.23.2 Core Methods

A renderer should be assigned to only one plot at a time. The following methods are used by JFreeChart to track the link between the renderer and the plot:

```

    ➔ public XYPlot getPlot();
    Returns the plot that this renderer is assigned to.

    ➔ public void setPlot(XYPlot plot);
    Sets the plot that this renderer is assigned to. JFreeChart calls this method, you shouldn't
    need to.
  
```

At certain times, the plot will ask the renderer to determine the axis bounds necessary to display all the data from a given dataset. Generally, the bounds will be the lowest and highest data values in the dataset, but depending on the type of rendering, a slightly different range may be required:

```

    ➔ public Range findDomainBounds(XYDataset dataset);
    Returns the bounds for a domain axis that this renderer requires to display all of the data in
    the specified dataset.

    ➔ public Range findRangeBounds(XYDataset dataset);
    Returns the bounds for a range axis that this renderer requires to display all of the data in the
    specified dataset.
  
```

The `initialise()` method is called once at the beginning of the chart drawing process, and gives the renderer a chance to initialise itself:

```

    ➔ public void initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot,
                           XYDataset dataset, ChartRenderingInfo info);
    Initialises the renderer. If possible, a renderer will pre-calculate any values that help to improve
    the performance of the drawItem() method.
  
```

Class:	Description:
<code>CandlestickRenderer</code>	Candlestick charts.
<code>ClusteredXYBarRenderer</code>	XY bar charts with automatic clustering.
<code>DeviationRenderer</code>	Line charts with deviation indicators.
<code>HighLowRenderer</code>	High-low-open-close charts.
<code>StackedXYAreaRenderer</code>	Stacked area charts.
<code>StackedXYBarRenderer</code>	Stacked bar charts.
<code>StandardXYItemRenderer</code>	Line charts and scatter plots.
<code>WindItemRenderer</code>	Wind charts.
<code>XYAreaRenderer</code>	Area charts.
<code>XYBarRenderer</code>	Bar charts with numerical domain values.
<code>XYBlockRenderer</code>	Heat map charts.
<code>XYBoxAndWhiskerRenderer</code>	Box-and-whisker charts.
<code>XYBubbleRenderer</code>	Bubble charts.
<code>XYDifferenceRenderer</code>	Difference charts.
<code>XYDotRenderer</code>	Scatter plots.
<code>XYErrorRenderer</code>	Line charts with error bars.
<code>XYLineAndShapeRenderer</code>	Line charts and scatter plots.
<code>XYStepRenderer</code>	Step charts.
<code>XYStepAreaRenderer</code>	Step charts.
<code>YIntervalRenderer</code>	Interval charts.

*Table 37.1: Classes that implement the `XYItemRenderer` interface*

The `drawItem()` method is responsible for drawing some representation of a particular data item within a plot:

```
► public void drawItem(Graphics2D g2, Rectangle2D dataArea, ChartRenderingInfo info,
XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
XYDataset dataset, int series, int item, CrosshairInfo info);
Draws a single data item on behalf of XYPlot.
```

```
► public int getPassCount();
Returns the number of passes (through the dataset) required by this renderer.
```

### 37.23.3 Series Visibility

A renderer should call the following method to determine if a data item is visible, before drawing it:

```
► public boolean getItemVisible(int series, int item);
Returns true if the data item should be drawn, and false otherwise. Very often, this method will be implemented to simply call isSeriesVisible(series).
```

```
► public boolean isSeriesVisible(int series);
Returns a flag that controls whether or not the items in the specified series are visible.
```

For convenience, the renderer defines the following methods that allow visibility flags to be specified on a per-series and default basis:

```
► public Boolean getSeriesVisible(int series);
Returns a flag (possibly null) that controls whether or not the items in the specified series are visible.
```

```
► public void setSeriesVisible(int series, Boolean visible);
Equivalent to setSeriesVisible(series, visible, true)—see the next method.
```

```
► public void setSeriesVisible(int series, Boolean visible, boolean notify);
Sets a flag (null is permitted) that controls whether or not the items in the specified series are visible. If visible is null, this typically means that the default flag value should apply.
```

A default series visibility flag is defined:

```
► public boolean getBaseSeriesVisible();
Returns the default series visibility flag. This default is typically used when no per-series flag has been defined.
```

```
→ public void setBaseSeriesVisible(boolean visible);
Equivalent to setBaseSeriesVisible(visible, true)—see the next method.

→ public void setBaseSeriesVisible(boolean visible, boolean notify);
Sets the default series visibility flag and, if requested, sends a RendererChangeEvent to all registered listeners.
```

### 37.23.4 The Paint and Outline Paint

This interface assumes that the renderer stores values for the paint and outline paint attributes on a per-series basis, with a default value that can be used when no value is specified for a particular series. By convention, the `XYPlot` will always call the following method to obtain the paint value for a data item:

```
→ public Paint getItemPaint(int row, int column);
Returns the paint to be used for the specified item. This method should never return null.
```

The per-series paint settings can be controlled via the following methods:

```
→ public Paint getSeriesPaint(int series);
Returns the paint for the specified series (possibly null).

→ public void setSeriesPaint(int series, Paint paint);
Sets the paint for the specified series (null is permitted) and sends a RendererChangeEvent to all registered listeners.
```

The default value (to be used in the case that there is no value specified for a particular series) is controlled via the following methods:

```
→ public Paint getBasePaint();
Returns the default paint, typically used when no per-series paint is defined. This method should never return null.

→ public void setBasePaint(Paint paint);
Sets the default paint to be used when no per-series paint is defined. If paint is null, this method should throw an IllegalArgumentException. If the new paint value is different to the existing value, this method should send a RendererChangeEvent to all registered listeners.
```

In a similar fashion, to obtain the outline paint for an item, the plot always call the following method:

```
→ public Paint getItemOutlinePaint(int row, int column);
Returns the paint to be used for the specified item.
```

The series outline paint settings can be controlled via the following methods:

```
→ public Paint getSeriesOutlinePaint(int series);
Returns the outline paint for the specified series (possibly null).

→ public void setSeriesOutlinePaint(int series, Paint paint);
Sets the outline paint (null permitted) for the specified series and sends a RendererChangeEvent to all registered listeners.

→ public Paint getBaseOutlinePaint();
Returns the default outline paint used when no per-series paint is defined. This method should never return null.

→ public void setBaseOutlinePaint(Paint paint);
Sets the default outline paint to be used when no per-series paint is defined. If paint is null, this method should throw an IllegalArgumentException. If the new paint value is different to the existing value, this method should send a RendererChangeEvent to all registered listeners.
```

### 37.23.5 The Stroke and Outline Stroke

All renderers maintain default and per-series settings for the stroke and outline stroke.

```
→ public Stroke getItemStroke(int series, int item);
Returns the stroke to use for the specified item in the dataset. This method should never return null.

→ public Stroke getSeriesStroke(int series);
Returns the stroke to use for the specified series. If this method returns null, the renderer will typically use the default stroke (see getBaseStroke()).

→ public void setSeriesStroke(int series, Stroke stroke);
Sets the stroke for the specified series and sends a RendererChangeEvent to all registered listeners. You can set the stroke to null, in which case the renderer will use the default stroke (see getBaseStroke()).
```

To control the default stroke:

```
→ public Stroke getBaseStroke();
Returns the default stroke (never null) for the renderer.

→ public void setBaseStroke(Stroke stroke);
Sets the default stroke for the renderer, and sends a RendererChangeEvent to all registered listeners. If stroke is null, this method should throw an IllegalArgumentException.
```

The renderer will call the following method to fetch the outline stroke for an item:

```
→ public Stroke getItemOutlineStroke(int row, int column);
Performs a lookup for the outline stroke. This method returns the per-series setting (see below) if it is not null, otherwise it returns the base outline stroke.
```

The per-series outline stroke can be determined using the following methods:

```
→ public Stroke getSeriesOutlineStroke(int series);
Returns the outline stroke for the specified series, or null.

→ public void setSeriesOutlineStroke(int series, Stroke stroke);
Sets the outline stroke (null is permitted) for the specified series, and sends a RendererChangeEvent to all registered listeners.
```

To control the default outline stroke:

```
→ public Stroke getBaseOutlineStroke();
Returns the default outline stroke (never null) for the renderer.

→ public void setBaseOutlineStroke(Stroke stroke);
Sets the default outline stroke for the renderer, and sends a RendererChangeEvent to all registered listeners. If stroke is null, this method should throw an IllegalArgumentException.
```

### 37.23.6 The Shapes

All renderers maintain a default shape and a set of per-series shapes, even though some renderers won't, in fact, require them.

```
→ public Shape getItemShape(int row, int column);
Returns a shape (never null) for the specified data item.
```

For the convenience of the developer interacting with this interface, it is assumed that renderers store shapes on a per-series basis, with a default shape to be used as the fallback.

```
→ public Shape getSeriesShape(int series);
Returns the shape (possibly null) for the specified series.

→ public void setSeriesShape(int series, Shape shape);
Sets the shape (null is permitted) for the specified series and sends a RendererChangeEvent to all registered listeners.
```

To control the default shape for the renderer:

► public Shape getBaseShape();

Returns the default shape (never `null`) for the renderer.

► public void setBaseShape(Shape shape);

Sets the default shape for the renderer, and sends a `RendererChangeEvent` to all registered listeners. If `shape` is `null`, this method should throw an `IllegalArgumentException`.

### 37.23.7 Item Labels

Item labels are small text items displayed near to the shape, line or bar representing a data item—the label will typically display the exact data value, but this is configurable by specifying a generator for the item labels. A renderer should always call the following method to obtain the item label generator for a particular data item:

► public XYItemLabelGenerator getItemLabelGenerator(int row, int column);

Returns the item label generator (possibly `null`) for the specified data item.

A typical renderer will allow a different item label generator to be specified for each series that the renderer handles, with a default item label generator for any series that doesn't have a generator specified. For convenience, the methods that support this are included in this interface:

► public XYItemLabelGenerator getSeriesItemLabelGenerator(int series);

Returns the item label generator (possibly `null`) for the specified series.

► public void setSeriesItemLabelGenerator(int series, XYItemLabelGenerator generator);

Sets the item label generator (`null` is permitted) for the specified series, and sends a `RendererChangeEvent` to all registered listeners.

The default item label generator is typically used when no other generator is available—if this is `null`, no item label is generated or displayed by the renderer:

► public XYItemLabelGenerator getBaseItemLabelGenerator();

Returns the default item label generator, which may be `null`.

► public void setBaseItemLabelGenerator(XYItemLabelGenerator generator);

Sets the default item label generator (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

### 37.23.8 The Item Label Font and Paint

When displaying an item label, the renderer will call the following methods to determine the font and paint to use for the label:

► public Font getItemLabelFont(int row, int column);

Returns the font (never `null`) used to display the item label for the specified data item.

► public Paint getItemLabelPaint(int row, int column);

Returns the paint (never `null`) used to display the item label for the specified data item.

A typical renderer will allow the font and paint settings to be specified on a per-series basis, with default settings to be used for any series without an explicit setting. For convenience, the methods that support this are included in this interface:

► public Font getSeriesItemLabelFont(int series);

Returns the font (possibly `null`) used to display the item labels for the specified series.

► public void setSeriesItemLabelFont(int series, Font font);

Sets the font (`null` is permitted) used to display the item labels for the specified series, and sends a `RendererChangeEvent` to all registered listeners.

To control the default font for item labels on the chart:

► public Font getBaseItemLabelFont();

Returns the default font (never `null`) for the item labels on the chart.

► public void setBaseItemLabelFont(Font font);  
 Sets the default font for the item labels on the chart, and sends a `RendererChangeEvent` to all registered listeners. If `font` is `null`, this method should throw an `IllegalArgumentException`.

To control the item label paint on a per-series basis:

► public Paint getSeriesItemLabelPaint(int series);  
 Returns the paint (possibly `null`) used to draw the item labels for the specified series.

► public void setSeriesItemLabelPaint(int series, Paint paint);  
 Sets the paint (`null` is permitted) used to draw the item labels for the specified series, and sends a `RendererChangeEvent` to all registered listeners.

To control the default paint for the item labels:

► public Paint getBaseItemLabelPaint();  
 Returns the default paint (never `null`) used to draw item labels (if these are visible).

► public void setBaseItemLabelPaint(Paint paint);  
 Sets the default paint used to draw item labels (if these are visible) and sends a `RendererChangeEvent` to all registered listeners.

### 37.23.9 The Item Label Position

The anchor position for item labels is specified in terms of a clock-face, with the renderer treating positive and negative data items differently (so that, for example, a bar renderer can place item labels for positive data items above the bar, and item labels for negative data items below the bar).

A renderer should always call the following method to find the anchor position for the item label for positive data items:

► public `ItemLabelPosition` getPositiveItemLabelPosition(int row, int column);  
 Returns the item label position that should be used for positive data items.

For the convenience of the developer interacting with this interface, it is assumed that the item label positions can be defined on a per series basis, with a default value to apply as the fall-back:

► public `ItemLabelPosition` getSeriesPositiveItemLabelPosition(int series);  
 Returns the item label position (possibly `null`) for the specified series.

► public void setSeriesPositiveItemLabelPosition(int series, `ItemLabelPosition` position);  
 Equivalent to `setSeriesPositiveItemLabelPosition(series, position, true)`—see the next method.

► public void setSeriesPositiveItemLabelPosition(int series, `ItemLabelPosition` position, boolean notify);  
 Sets the item label position (`null` is permitted) for the specified series and sends a `RendererChangeEvent` to all registered listeners.

The default position:

► public `ItemLabelPosition` getBasePositiveItemLabelPosition();  
 Returns the default item label position (never `null`) to be used when there is no per-series setting.

► public void setBasePositiveItemLabelPosition(`ItemLabelPosition` position);  
 Equivalent to `setBasePositiveItemLabelPosition(position, true)`—see the next method.

► public void setBasePositiveItemLabelPosition(`ItemLabelPosition` position, boolean notify);  
 Sets the default item label position (`null` is not permitted) and sends a `RendererChangeEvent` to all registered listeners.

The renderer will call the following method to get the position for any data item with a negative value:

► public `ItemLabelPosition` getNegativeItemLabelPosition(int row, int column);  
 Returns the item label position for the specified data item. The renderer will only call this method for data items with negative values.

For convenience, this interface assumes that the renderer supports defining item label positions on a per-series basis, with a default value for any series with no explicit setting, via the following methods:

- **public ItemLabelPosition getSeriesNegativeItemLabelPosition(int series);**  
Returns the item label position (possibly `null`) for any data item in the specified series that has a negative value.
- **public void setSeriesNegativeItemLabelPosition(int series, ItemLabelPosition position);**  
Equivalent to `setSeriesNegativeItemLabelPosition(series, position, true)`—see next method.
- **public void setSeriesNegativeItemLabelPosition(int series, ItemLabelPosition position, boolean notify);**  
Sets the item label position for the specified series and sends a `RendererChangeEvent` to all registered listeners.

To control the default item label position:

- **public ItemLabelPosition getBaseNegativeItemLabelPosition();**  
Returns the default item label position for data items that have a negative value.
- **public void setBaseNegativeItemLabelPosition(ItemLabelPosition position);**  
Equivalent to `setBaseNegativeItemLabelPosition(position, true)`—see the next constructor.
- **public void setBaseNegativeItemLabelPosition(ItemLabelPosition position, boolean notify);**  
Sets the default item label position for data items that have a negative value, and sends a `RendererChangeEvent` to all registered listeners.

### 37.23.10 The Item Label Visibility

To determine if an item label is visible for a particular data item, the renderer should always call the following method:

- **public boolean isItemLabelVisible(int row, int column);**  
Returns `true` if the item label for this data item should be displayed, and `false` otherwise.
- **public boolean isSeriesItemLabelsVisible(int series);**  
Returns `true` if item labels are visible for the specified series, and `false` otherwise.

To set the flag that controls whether or not item labels are visible for a series:

- **public void setSeriesItemLabelsVisible(int series, boolean visible);**  
Equivalent to `setSeriesItemLabelsVisible(series, Boolean.valueOf(visible), true)`—see the next constructor.
- **public void setSeriesItemLabelsVisible(int series, Boolean visible);**  
Equivalent to `setSeriesItemLabelsVisible(series, visible, true)`—see the next constructor.
- **public void setSeriesItemLabelsVisible(int series, Boolean visible, boolean notify);**  
Sets the flag that controls whether or not item labels are visible for a series and, if requested, sends a `RendererChangeEvent` to all registered listeners.

To control the default value:

- **public Boolean getBaseItemLabelsVisible();**  
Returns the default flag that controls whether or not item labels are visible.
- **public void setBaseItemLabelsVisible(boolean visible);**  
Sets the default flag that controls whether or not item labels are visible, and sends a `RendererChangeEvent` to all registered listeners.
- **public void setBaseItemLabelsVisible(Boolean visible);**  
Sets the default flag that controls whether or not item labels are visible, and sends a `RendererChangeEvent` to all registered listeners.
- **public void setBaseItemLabelsVisible(Boolean visible, boolean notify);**  
Sets the default flag that controls whether or not item labels are visible, and sends a `RendererChangeEvent` to all registered listeners.

### 37.23.11 Tool Tips

Tool tips that the renderer assigns to each data item are generated by a tool tip generator. For each data item, the renderer will obtain the tool tip generator by calling the following method:

```
➔ public XYToolTipGenerator getToolTipGenerator(int row, int column);
```

Returns the tool tip generator for the specified data item. If this method returns `null`, no tool tip will be assigned for this item.

Most renderers allow a tool tip generator to be specified on a per-series basis, with a default generator providing the fall-back. For convenience, the methods that are used to specify these generators are included in the interface:

```
➔ public XYToolTipGenerator getSeriesToolTipGenerator(int series);
```

Returns the tool tip generator (possibly `null`) for the specified series.

```
➔ public void setSeriesToolTipGenerator(int series, XYToolTipGenerator generator);
```

Sets the tool tip generator (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

The default generator, typically used when no per-series generator is assigned, is controlled via the following methods:

```
➔ public XYToolTipGenerator getBaseToolTipGenerator();
```

Returns the default tool tip generator, which may be `null`. This default is typically used when the per-series generator is `null`.

```
➔ public void setBaseToolTipGenerator(XYToolTipGenerator generator);
```

Sets the default tool tip generator (`null` is permitted) and sends a `RendererChangeEvent` to all registered listeners.

For more information about tool tips, refer to section 11.

### 37.23.12 URLs

A URL generator is used to generate URLs for each item drawn by the renderer. These URLs are only used in the creation of HTML image maps. A single URL generator is used for all data items, this attribute has not been defined on a per-series basis.

```
➔ public XYURLGenerator getURLGenerator();
```

Returns the URL generator (possibly `null`) used for all data items.

```
➔ public void setURLGenerator(XYURLGenerator urlGenerator);
```

Sets the URL generator (`null` is permitted) to be used for all data items.

For more information about HTML image map generation, refer to section 30.3.1.

### 37.23.13 Legend Items

This interface extends `LegendItemSource` which means that all renderers are a source of legend items.

```
➔ public LegendItem getLegendItem(int datasetIndex, int series);
```

Returns a legend item for the specified series. This method can return `null`, which means that no item will be displayed in the legend for the specified series.

A plug-in generator is responsible for creating the text label for each series in the legend:

```
➔ public XYSeriesLabelGenerator getLegendItemLabelGenerator();
```

Returns the legend item label generator. This method should not return null.

```
➔ public void setLegendItemLabelGenerator(XYSeriesLabelGenerator generator);
```

Sets the legend item label generator and sends a `RendererChangeEvent` to all registered listeners. If `generator` is `null`, this method should throw an `IllegalArgumentException`.

### 37.23.14 Series Visibility in Legend

The chart's legend (if it has one) will display the name of each series drawn by the renderer, unless a flag is set to hide the series from the legend.

```
→ public boolean isSeriesVisibleInLegend(int series);
```

Returns `true` if the specified series should be included in the legend, and `false` otherwise. This method is typically implemented to do a look-up on the per-series and default visibility flags—see the methods below.

For convenience, the methods that control the per-series and default visibility flags are included in this interface:

```
→ public Boolean getSeriesVisibleInLegend(int series);
```

Returns the flag that controls whether the specified series is visible in the legend. This method can return `null`, which means that no flag is set for the series.

```
→ public void setSeriesVisibleInLegend(int series, Boolean visible);
```

Equivalent to `setSeriesVisibleInLegend(series, visible, true)`—see the next method.

```
→ public void setSeriesVisibleInLegend(int series, Boolean visible, boolean notify);
```

Sets the flag that controls whether or not the specified series is included in the legend, and sends a `RendererChangeEvent` to all registered listeners.

```
→ public boolean getBaseSeriesVisibleInLegend();
```

Returns the default flag that controls whether or not a series is visible in the legend. The default value is `true`.

```
→ public void setBaseSeriesVisibleInLegend(boolean visible);
```

Equivalent to `setBaseSeriesVisibleInLegend(visible, true)`—see the next method.

```
→ public void setBaseSeriesVisibleInLegend(boolean visible, boolean notify);
```

Sets the default flag that controls whether or not a series is visible in the legend, and sends a `RendererChangeEvent` to all registered listeners.

### 37.23.15 Other Methods

A range of other methods in the interface are intended for use by the `XYPlot` class when interacting with a renderer.

```
→ public void drawDomainGridLine(Graphics2D g2, XYPlot plot, ValueAxis axis, Rectangle2D dataArea, double value);
```

Draws a gridline for the specified domain axis at the given `value`.

```
→ public void drawRangeLine(Graphics2D g2, XYPlot plot, ValueAxis axis, Rectangle2D dataArea, double value, Paint paint, Stroke stroke);
```

Draws a line perpendicular the the specified range axis corresponding to the given `value`.

The plot calls the following methods to draw markers against the domain and range axes:

```
→ public void drawDomainMarker(Graphics2D g2, XYPlot plot, ValueAxis axis, Marker marker, Rectangle2D dataArea);
```

Draws the specified marker that highlights a value or interval along the specified domain axis.

```
→ public void drawRangeMarker(Graphics2D g2, XYPlot plot, ValueAxis axis, Marker marker, Rectangle2D dataArea);
```

Draws the specified marker that highlights a value or interval along the specified range axis.

The following methods support the painting of grid bands:

```
→ public void fillDomainGridBand(Graphics2D g2, XYPlot plot, ValueAxis axis, Rectangle2D dataArea, double start, double end);
```

Fills a band representing the specified interval along the given domain axis.

```
→ public void fillRangeGridBand(Graphics2D g2, XYPlot plot, ValueAxis axis, Rectangle2D dataArea, double start, double end);
```

Fills a band representing the specified interval along the given range axis.

### 37.23.16 Change Listeners

All renderers must support a change notification mechanism that allows a listener to register with the renderer and receive a `RendererChangeEvent` whenever any attribute of the renderer changes. This mechanism is used by JFreeChart to provide automatic updating of charts displayed in a `ChartPanel`.

```
► public void addChangeListener(RendererChangeListener listener);
Registers a listener so that it will receive notification of changes to this renderer.

► public void removeChangeListener(RendererChangeListener listener);
De-registers a listener so that it will no longer receive notification of changes to this renderer.
```

The `XYPlot` class will automatically register itself with any renderer that is assigned to it.

### 37.23.17 Annotations

You can assign one or more `XYAnnotation` instances to a renderer. These annotations will be drawn relative to the axes that the renderer is mapped to. For example, see `AnnotationDemo2.java` in the JFreeChart demos.

```
► public void addAnnotation(XYAnnotation annotation);
Adds the annotation to the foreground layer for this renderer.

► public void addAnnotation(XYAnnotation annotation, Layer layer);
Adds the annotation to the specified layer for this renderer.

► public boolean removeAnnotation(XYAnnotation annotation);
Removes an annotation from the renderer.

► public void removeAnnotations();
Removes all annotations from the renderer.

► public void drawAnnotations(Graphics2D g2, Rectangle2D dataArea,
ValueAxis domainAxis, ValueAxis rangeAxis, Layer layer,
PlotRenderingInfo info);
Draws the annotations in the specified layer.
```

Note that you can also add annotations directly to an `XYPlot`, in which case they are drawn relative to the plot's primary axes.

### 37.23.18 Notes

Some renderers require a dataset that is a specific extension of `XYDataset`. For example, the `HighLowRenderer` requires an `OHLCDataset`.

#### See Also

[AbstractXYItemRenderer](#), [XYPlot](#).

## 37.24 XYItemRendererState

### 37.24.1 Overview

A state object that retains information between the successive calls to a renderer's `drawItem()` method. This class extends the `RendererState` class, and is used internally by JFreeChart (you won't normally need to use it directly, unless you are writing your own renderer class).

### 37.24.2 Constructor

To create a new instance:

```
► public XYItemRendererState(PlotRenderingInfo info);
Creates a new state instance.
```

### 37.24.3 Fields

This class defines a working `Line2D` instance that can be reused by a renderer to avoid creating numerous instances that require garbage collection:

```
➔ public Line2D workingLine;
A reusable line.
```

#### See Also

[RendererState](#)

## 37.25 XYLineAndShapeRenderer

### 37.25.1 Overview

A *renderer* that displays items from an [XYDataset](#) by drawing a line between each  $(x, y)$  point and overlaying a shape at each  $(x, y)$  point—for an example, see figure 37.18. One of the key features of this renderer is that it allows you to control on a *per series* basis whether:

- lines are drawn between the data points;
- shapes are drawn at each data point;
- shapes are filled or not filled;

This class implements the [XYItemRenderer](#) interface, so it can be used with the [XYPlot](#) class. It extends the [AbstractXYItemRenderer](#) base class. Subclasses include:

- [DeviationRenderer](#);
- [XYErrorRenderer](#);
- [XYSplineRenderer](#).

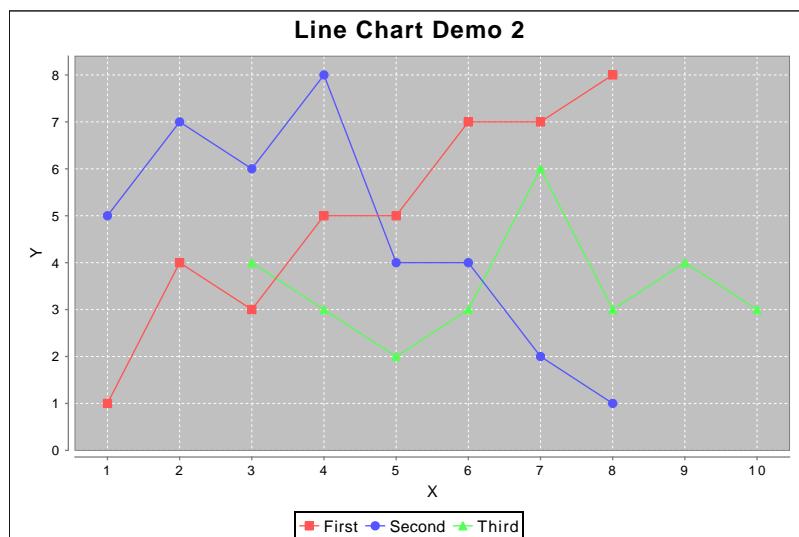


Figure 37.18: A sample chart (see `LineChartDemo2.java`)

### 37.25.2 Usage

This renderer is used for several types of chart created via the `ChartUtilities` class:

- `createXYLineChart()`—creates a standard line chart;
- `createTimeSeriesChart()`—creates a time series line chart;
- `createScatterPlot()`—creates a scatter plot;

Given an arbitrary `XYPlot`, you can install a new instance of this renderer using the following code (or a variation of it):

```
XYPlot plot = (XYPlot) chart.getPlot();
XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
renderer.setSeriesLinesVisible(0, true);
renderer.setSeriesShapesVisible(0, false);
renderer.setSeriesLinesVisible(1, false);
renderer.setSeriesShapesVisible(1, true);
plot.setRenderer(renderer);
```

Flags have been set so that items in the first series are connected with lines, while items in the second series are displayed as individual shapes.

### 37.25.3 Constructor

This class has two constructors:

- `public XYLineAndShapeRenderer();`  
Equivalent to `XYLineAndShapeRenderer(true, true)`—see the next constructor.
- `public XYLineAndShapeRenderer(boolean lines, boolean shapes);`  
Creates a new renderer, with connecting lines and shapes for each data item as requested.

### 37.25.4 Lines

This renderer has, as you'd expect, the facility to connect data items with lines. The color and style of the lines is controlled by methods that are inherited from the `AbstractRenderer` class (for example, `setSeriesPaint()` and `setSeriesStroke()`).

In addition, this renderer defines a set of flags provides the option to show or hide the lines on a per-series basis. To determine whether or not a line is drawn for an item (connecting the current item with the previous item), JFreeChart calls the following method:

- `public boolean getItemLineVisible(int series, int item);`  
Returns a flag that controls whether or not a line is drawn between the current and previous items.

#### Override Line Visibility

To control whether or not lines are drawn for the items in ALL series (this is an override setting that you should seldom if ever need to use):

- `public Boolean getLinesVisible(); [Deprecated 1.0.7]`  
Returns the flag that controls whether lines are drawn for the items in ALL series. This flag overrides all other settings, unless it is `null`. The default value is `null`.
- `public void setLinesVisible(Boolean visible); [Deprecated 1.0.7]`  
Sets the flag that controls whether or not lines are drawn for the items in ALL series, and sends a `RendererChangeEvent` to all registered listeners. You should leave this flag set to `null` if you prefer to use the “per series” flags (see the next section).
- `public void setLinesVisible(boolean visible); [Deprecated 1.0.7]`  
As above.

This override setting has been deprecated from version 1.0.7 onwards—you can simply rely on the per-series and base level settings.

## Per Series Line Visibility

To control whether or not lines are drawn for the items in one series (this assumes the (now deprecated) override flag in the previous section is set to `null`, which is the default):

```
➔ public Boolean getSeriesLinesVisible(int series);
Returns a flag that controls whether or not lines are drawn for the items in the specified series.

➔ public void setSeriesLinesVisible(int series, Boolean flag);
Sets a flag that controls whether or not lines are drawn for the items in the specified series,
and sends a RendererChangeEvent to all registered listeners. If this is set to null, then the base
setting (see the next section) will apply.

➔ public void setSeriesLinesVisible(int series, boolean visible);
As above.
```

The flags are stored as `Boolean` objects—if the flag is `null` for a series, then the base value is used (see the next section).

## Base Line Visibility

The base line visibility setting is used for any series where both the override and per series settings are not specified:

```
➔ public boolean getBaseLinesVisible();
Returns the default line visibility for all series. This setting is used when no per-series or
override setting is specified. The initial value is specified in the constructor.

➔ public void setBaseLinesVisible(boolean flag);
Sets the default flag that controls whether or not the renderer draws lines between the  $(x, y)$ 
items in a series, and sends a RendererChangeEvent to all registered listeners.
```

It is recommended that you set the default value as required first, and then override the setting on a per series basis. If you have set the flag for a series, but later want to restore the default value, note that there is a version of the `setSeriesLinesVisible()` method that accepts a `Boolean` flag which you can set to `null`.

### 37.25.5 Dashed Lines

It is common to use a dashed stroke to draw the connecting lines between items within a series. An issue arises when the items within a series are close together on the chart, because by default the renderer will draw the connecting line individually for each data item (connecting the current item to the previous item). The stroke pattern for the line is reset for each segment, which can result in the stroke pattern not being visible for the series. A workaround is available, in which the connecting lines for the entire series are drawn as a single line:

```
➔ public boolean getDrawSeriesLineAsPath();
Returns the flag that controls whether the connecting lines between data items are drawn as a
path, or individually. The default value is false (individual line segments).

➔ public void setDrawSeriesLineAsPath(boolean flag);
Sets the flag that controls whether the connecting lines between the data items are drawn as a
path, or individually. If the flag value is changed, a RendererChangeEvent is sent to all registered
listeners.
```

### 37.25.6 Item Shapes – Visibility

This renderer can draw shapes at each data point, and has flags that control:

- the visibility of the shapes;
- whether or not the shapes are filled;

To find out whether or not a shape for a data item is to be displayed:

► `public boolean getItemShapeVisible(int series, int item);`

Returns `true` if a shape should be displayed for the specified item. This performs a lookup on the override, per-series and base level visibility settings for shape visibility—see the following methods.

### Override Shape Visibility

To control whether or not shapes are drawn for the items in ALL series (this is an override setting that you should seldom, if ever, need to use):

► `public Boolean getShapesVisible(); [Deprecated 1.0.7]`

Returns the flag that controls whether shapes are drawn for the items in ALL series. This flag overrides all other settings, unless it is `null` (the default).

► `public void setShapesVisible(Boolean visible); [Deprecated 1.0.7]`

Sets the flag that controls whether or not shapes are drawn for the items in ALL series, and sends a `RendererChangeEvent` to all registered listeners. You should leave this flag set to `null` if you prefer to use the per-series flags (described in the next section).

► `public void setShapesVisible(boolean visible); [Deprecated 1.0.7]`

As above.

This override setting has been deprecated from version 1.0.7 onwards—you can simply rely on the per-series and base level settings.

### Per Series Shape Visibility Flags

To control whether or not shapes are drawn for items in a series (this assumes the (now deprecated) override flag in the previous section is set to `null`, which is the default):

► `public Boolean getSeriesShapesVisible(int series);`

Returns a flag that indicating whether or not shapes are displayed for the items in the specified series.

► `public void setSeriesShapesVisible(int series, Boolean flag);`

Sets a flag that controls whether or not shapes are drawn for the items in the specified series, then sends a `RendererChangeEvent` to all registered listeners. If this is set to `null`, then the base setting (described in the next section) will apply.

► `public void setSeriesShapesVisible(int series, boolean visible);`

As above.

### Base Shape Visibility

The base shape visibility setting is used for any series where both the override and per-series settings are not specified:

► `public boolean getBaseShapesVisible();`

Returns the default shape visibility for all series. This setting is used when no per-series or override setting is specified. The initial value is specified via the constructors.

► `public void setBaseShapesVisible(boolean flag);`

Sets the default flag that controls whether or not the renderer draws shapes for each ( $x, y$ ) item in a series, then sends a `RendererChangeEvent` to all registered listeners.

### 37.25.7 Item Shapes – Fill State

This renderer defines flags to control whether or not the item shapes (if they are visible) are filled or “hollow”:

► `public boolean getItemShapeFilled(int series, int item);`

Returns `true` if the specified item should be filled, and `false` otherwise. This method does a lookup based on the flags defined in the following sections.

## Override Shapes Filled

To control whether or not shapes are filled for the items in ALL series (this is an override setting that you should seldom need to use):

► `public void setShapesFilled(boolean filled); [Deprecated 1.0.7]`  
 Equivalent to `setShapesFilled(Boolean.valueOf(filled))`—see the next method.

► `public void setShapesFilled(Boolean filled); [Deprecated 1.0.7]`  
 Sets the flag that controls whether or not shapes are filled for the items in ALL series, and sends a `RendererChangeEvent` to all registered listeners. You should leave this flag set to `null` if you prefer to use the per-series flags (described in the next section).

This override setting has been deprecated from version 1.0.7 onwards—you can simply rely on the per-series and base-level settings.

## Series Shapes Filled

To control whether or not shapes are filled for items in a series (this assumes the (now deprecated) override flag in the previous section is set to `null`, which is the default):

► `public Boolean getSeriesShapesFilled(int series);`  
 Returns a flag that indicates whether or not the items for the specified series should be filled. This method can return `null`, which means the renderer will use the base default setting.

► `public void setSeriesShapesFilled(int series, boolean flag);`  
 Equivalent to `setSeriesShapesFilled(series, Boolean.valueOf(flag))`—see the next method.

► `public void setSeriesShapesFilled(int series, Boolean flag);`  
 Sets the flag that controls whether or not the renderer fills the shapes for the specified series, and sends a `RendererChangeEvent` to all registered listeners.

## Base Shapes Filled

This base shapes filled setting is used for any series where both the override and per-series settings are not specified:

► `public boolean getBaseShapesFilled();`  
 Returns the default shape filled flag for all series. This setting is used when no per-series or override setting is specified. The default values is `true`.

► `public void setBaseShapesFilled(boolean flag);`  
 Sets the default shapes filled flag that controls whether or not the renderer

### 37.25.8 Other Shape Flags

There are several additional flags that affect the shape rendering. First, shapes can be drawn with or without outlines:

► `public boolean getDrawOutlines();`  
 Returns `true` if the item shapes are drawn with outlines, and `false` otherwise. The default value is `true`.

► `public void setDrawOutlines(boolean flag);`  
 Sets the flag that controls whether or not outlines are drawn for the item shapes, and sends a `RendererChangeEvent` to all registered listeners.

When rendering shapes, there are three paint settings that can be used: (1) the series paint, (2) the series fill paint and (3) the series outline paint. By default, the renderer will simply use the first option, the regular series paint. However, you can modify two flags to have the renderer use the other paint settings—see figure 37.19 to see the effects.

The first flag controls whether or not the renderer uses the outline paint to draw shape outlines—if this is `false`, the regular series paint is used:

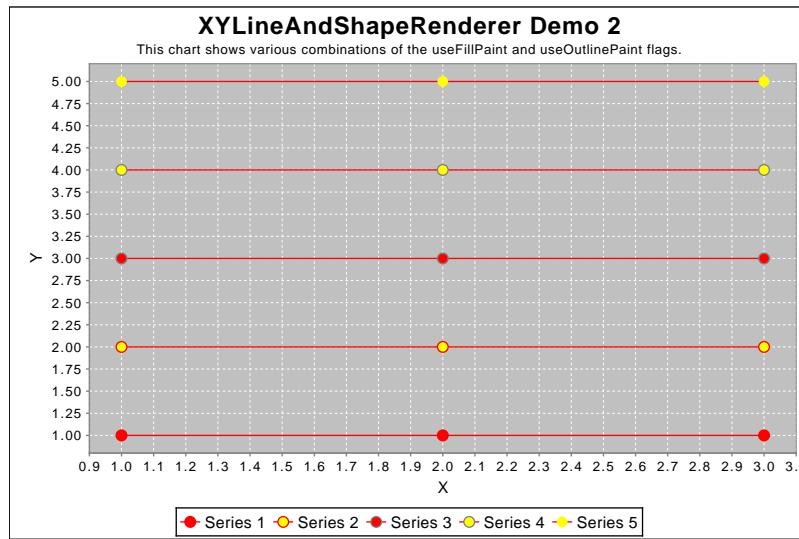


Figure 37.19: Fill and outline options

► `public boolean getUseOutlinePaint();`

Returns `true` if the shape outlines are drawn using the outline paint, and `false` if the shape outlines are drawn using the regular paint. The default value is `false`.

► `public void setUseOutlinePaint(boolean flag);`

Sets the flag that controls whether or not shape outlines are drawn using the outline paint (if not, the series paint is used instead), then sends a `RendererChangeEvent` to all registered listeners.

In a similar way, the other flag controls whether or not shapes are filled using the series fill paint:

► `public boolean getUseFillPaint();`

Returns `true` if the series fill paint is used to fill shapes, and `false` if the regular series paint is used instead. The default value is `false`.

► `public void setUseFillPaint(boolean flag);`

Sets the flag that controls whether the series fill paint is used by the renderer to fill shapes, and sends a `RendererChangeEvent` to all registered listeners.

### 37.25.9 Legend Customisation

This renderer allows a simple customisation of the legend display where you can specify the shape (typically a line, but any shape is permitted) that will represent each series in the legend:

► `public Shape getLegendLine();`

Returns the shape used to represent a series in the legend. The default value is `Line2D.Double(-7.0, 0.0, 7.0, 0.0)`. This method never returns `null`.

► `public void setLegendLine(Shape line);`

Sets the shape used to represent a series in the legend and sends a `RendererChangeEvent` to all registered listeners. This method throws an `IllegalArgumentException` if `line` is `null`. The supplied shape should be centered around `(0, 0)` as it will be translated into position by JFreeChart's drawing code.

For an example, see `TimeSeriesDemo7.java` in the JFreeChart demo collection.

### 37.25.10 Other Methods

The renderer makes two passes through the dataset, drawing the lines in the first pass, and then drawing the shapes in the second pass. The number of passes is returned by the following method:

```
► public int getPassCount();
Returns 2.
```

### 37.25.11 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 37.25.12 Notes

Some points to note:

- the renderer makes two passes through the data. In the first pass, the lines connecting the  $(x, y)$  data points are drawn. In the second pass, the shapes at each data point are drawn. In this way, the lines appear to be “under” the shapes, which makes for a better presentation;
- there is some overlap between this class and the `StandardXYItemRenderer` class—in general, you should try to use `XYLineAndShapeRenderer`;
- there are many demos for this renderer included in the JFreeChart demo collection (for example, `XYLineAndShapeRendererDemo1.java`).

## 37.26 XYSplinerenderer

### 37.26.1 Overview

An extension of the `XYLineAndShapeRenderer`, this class connects data points using spline curves—this results in a smooth line passing through all the data points. See figure 37.20 for an example. **This renderer was first introduced in JFreeChart version 1.0.7.**

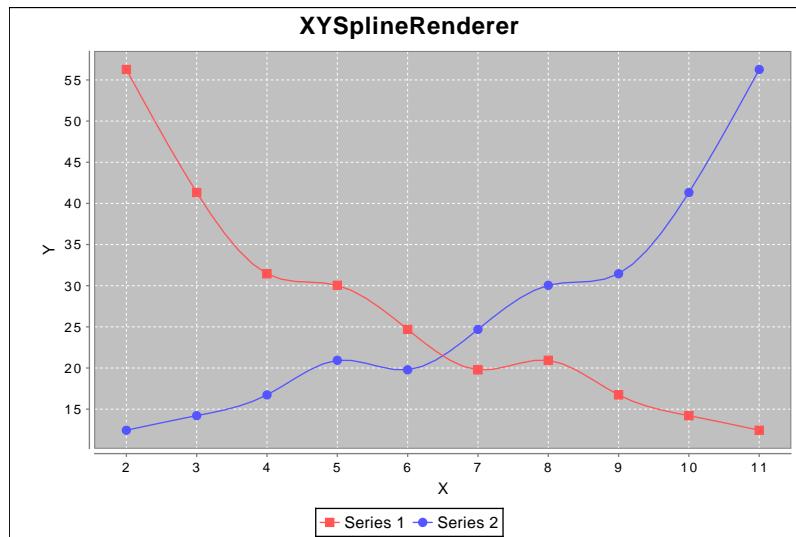


Figure 37.20: A sample chart (see `XYSplinerendererDemo1a.java`)

### 37.26.2 Constructors

This class defines two constructors:

► `public XYsplineRenderer(); [1.0.7]`

Equivalent to `XYsplineRenderer(5)`—see the next method.

► `public XYsplineRenderer(int precision); [1.0.7]`

Creates a new renderer, with the specified precision, that displays both lines and shapes.

### 37.26.3 General Attributes

This renderer inherits most of its attributes from `XYLineAndShapeRenderer`, but adds a precision attribute that controls the number of line segments used to approximate the curve between data points:

► `public int getPrecision(); [1.0.7]`

Returns the number of line segments used to approximate the curve between data points. The initial value is specified in the constructor (the default is 5).

► `public void setPrecision(int p); [1.0.7]`

Sets the number of line segments used to approximate the curve between data points, and sends a `RendererChangeEvent` to all registered listeners.

### 37.26.4 Other Methods

The following methods, overridden in this class, are typically called by JFreeChart—you shouldn't need to call these methods directly:

► `public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot, XYDataset data, PlotRenderingInfo info); [1.0.7]`

Initialises the renderer. The `drawSeriesLineAsPath` flag is set to `true`, and the `processVisibleItemsOnly` flag is set to `false`—the latter optimisation can't work in combination with the spline curve fitting.

► `protected void drawPrimaryLineAsPath(XYItemRendererState state, Graphics2D g2, XYPlot plot, XYDataset dataset, int pass, int series, int item, ValueAxis domainAxis, ValueAxis rangeAxis, Rectangle2D dataArea); [1.0.7]`

Draws the line representing all the points in one series. This method is overridden to apply the spline curve fitting.

### 37.26.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj); [1.0.7]`

Tests this renderer for equality with an arbitrary object (which may be `null`).

Instances of this class are cloneable and serializable.

### 37.26.6 Notes

Some points to note:

- there is no `ChartFactory` method to create a chart using this renderer, so you need to construct the chart in piece-wise fashion (refer to the demo applications for guidance);
- a demo (`XYsplineRendererDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[XYLineAndShapeRenderer](#)

## 37.27 XYStepRenderer

### 37.27.1 Overview

An *XY step renderer* draws items from an `XYDataset` using “stepped” lines to connect each  $(x, y)$  point. This renderer is designed for use with the `XYPlot` class.

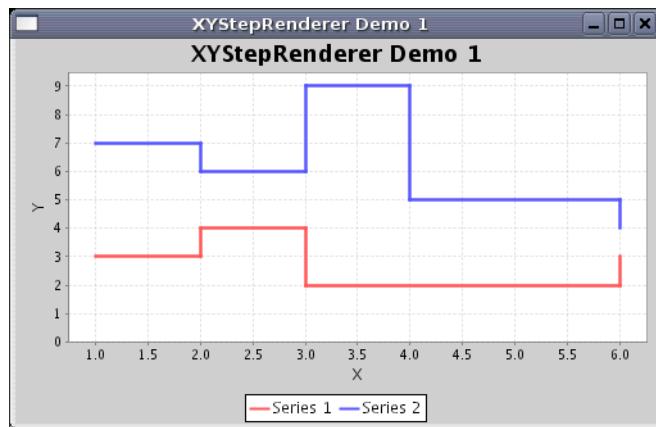


Figure 37.21: A sample chart using `XYStepRenderer`

### 37.27.2 Usage

A demo (`XYStepRendererDemo1.java`) is included in the JFreeChart demo distribution.

### 37.27.3 Constructors

To create a new renderer:

```
→ public XYStepRenderer();
Creates a new default renderer.

→ public XYStepRenderer(XYToolTipGenerator toolTipGenerator, XYURLGenerator urlGenerator);
Creates a new renderer with the specified tool tip generator and URL generator.
```

### 37.27.4 Equals, Cloning and Serialization

This renderer inherits an `equals()` method from its superclass. The renderer is both `Cloneable` and `Serializable`.

### 37.27.5 Notes

Some points to note:

- the “hotspot” for tooltips is a square centered on the data point (but not the corner of the “step”). You can use the `setDefaultEntityRadius()` method in the `AbstractXYItemRenderer` class to increase the size of the hotspot.

#### See Also

[CategoryStepRenderer](#).

## 37.28 XYStepAreaRenderer

### 37.28.1 Overview

A renderer that displays data from an `XYDataset` in a step format with the area under the steps filled—see figure 37.22 for an example. A demo (`XYStepAreaRendererDemo1.java`) is included in the JFreeChart demo collection.

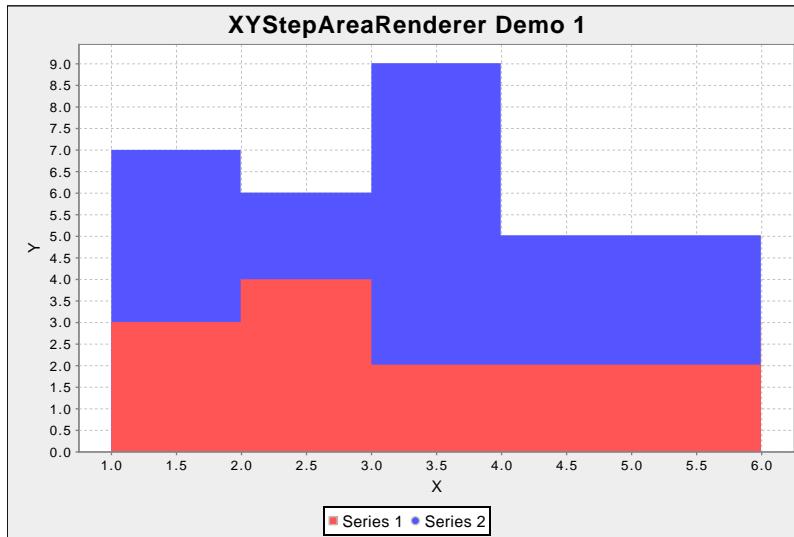


Figure 37.22: A sample chart (see `XYStepAreaRendererDemo1.java`)

### 37.28.2 Constructors

The following constructors are defined:

► `public XYStepAreaRenderer();`

Equivalent to `XYStepAreaRenderer(AREA)`—see the next constructor.

► `public XYStepAreaRenderer(int type);`

Equivalent to `XYStepAreaRenderer(type, null, null)`—see the next constructor.

► `public XYStepAreaRenderer(int type, XYToolTipGenerator toolTipGenerator, XYURLGenerator urlGenerator);`

Creates a new renderer with the specified `type`, `toolTipGenerator` and `urlGenerator`. The `type` should be one of the constants defined by this class:

- `AREA` – fills the area under the steps, but does not draw shapes;
- `SHAPES` – draws shapes at each data point, but does not fill the area beneath the data points;
- `AREA_AND_SHAPES` – fills the area under the steps, and draws shapes at each data point.

Both the `toolTipGenerator` and `urlGenerator` arguments can be `null`.

### 37.28.3 General Attributes

The following methods control the general attributes for this renderer:

► `public boolean isOutline();`

Returns the flag that controls whether or not the outline is shown. The default is `false`.

```

→ public void setOutline(boolean show);
Sets the flag that controls whether or not an outline is drawn around the step area, and sends a RendererChangeEvent to all registered listeners.

→ public boolean getShapesVisible();
Returns the flag that controls whether or not shapes are drawn at each data point.

→ public void setShapesVisible(boolean flag);
Sets the flag that controls whether or not shapes are drawn at each data point, and sends a RendererChangeEvent to all registered listeners.

→ public boolean isShapesFilled();
Returns the flag that controls whether or not shapes (if visible) are filled.

→ public void setShapesFilled(boolean filled);
Sets the flag that controls whether or not shapes are filled, and sends a RendererChangeEvent to all registered listeners.

→ public boolean getPlotArea();
Returns the flag that controls whether or not the area beneath the steps is filled.

→ public void setPlotArea(boolean flag);
Sets the flag that controls whether or not the area beneath the steps is filled, and sends a RendererChangeEvent to all registered listeners.

→ public double getRangeBase();
Returns the base value for the renderer. The default value is 0.0.

→ public void setRangeBase(double val);
Sets the base value for the renderer and sends a RendererChangeEvent to all registered listeners.

```

### 37.28.4 Other Methods

The other methods in this class are called by JFreeChart—you won’t normally call these directly:

```

→ public XYItemRendererState initialise(Graphics2D g2, Rectangle2D dataArea, XYPlot plot,
XYDataset data, PlotRenderingInfo info);
Initialises the renderer, returning a state object that will be passed to each call to drawItem().

→ public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea,
PlotRenderingInfo info, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
XYDataset dataset, int series, int item, CrosshairState crosshairState, int pass);
Draws one item on the chart. This method is called by the plot for each item in the dataset.

```

### 37.28.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```

→ public boolean equals(Object obj);
Tests this renderer for equality with an arbitrary object. This method returns true if:


- obj is not null;
- obj is an instance of XYStepAreaRenderer;
- obj has the same attributes as this renderer.

```

Instances of this class are `Cloneable` and `Serializable`.

## 37.29 YIntervalRenderer

### 37.29.1 Overview

An `XYItemRenderer` that draws lines indicating a y-interval corresponding to each x-value—see figure 37.23 for an example. This renderer requires an `IntervalXYDataset`, and is designed for use with the `XYPlot` class.

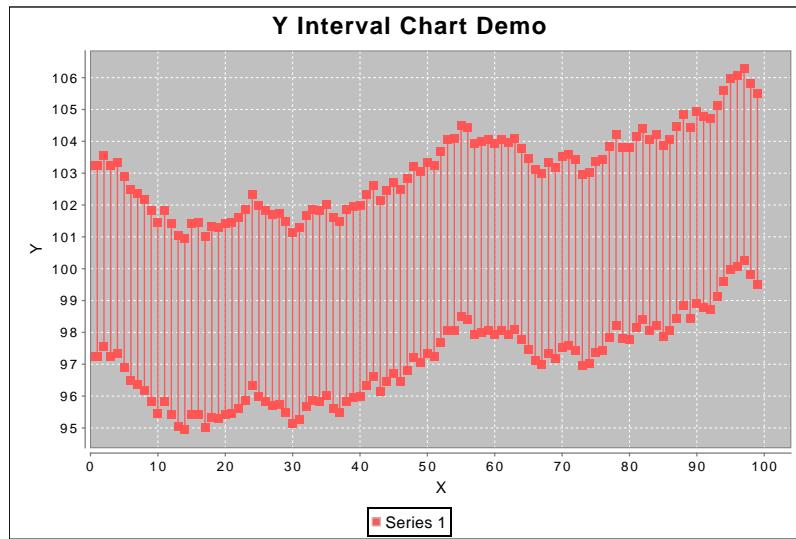


Figure 37.23: A sample chart (see `YIntervalChartDemo1.java`)

### 37.29.2 Constructors

To create a new renderer:

```
→ public YIntervalRenderer();
Creates a new renderer with default attributes.
```

### 37.29.3 Methods

The following method is called by the `XYPlot` class for each data item it needs to draw:

```
→ public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea,
PlotRenderingInfo info, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
XYDataset dataset, int series, int item, CrosshairState crosshairState, int pass)
Draws one item from the dataset. This method is called by the XYPlot class, you won't normally
call it yourself.
```

### 37.29.4 Notes

Some points to note:

- to customise the shapes drawn at the end points of the intervals, use the `setSeriesShape(int, Shape)` method inherited from `AbstractXYItemRenderer`;
- a demo application (`YIntervalChartDemo1.java`) is included in the JFreeChart demo distribution.

# Chapter 38

## Package: org.jfree.chart.servlet

### 38.1 Overview

This package contains servlet utility classes developed for JFreeChart by Richard Atkinson. An excellent demo for these classes can be found at:

[http://homepage.ntlworld.com/richard\\_c\\_atkinson/jfreechart](http://homepage.ntlworld.com/richard_c_atkinson/jfreechart)

### 38.2 ChartDeleter

#### 38.2.1 Overview

A utility class that maintains a list of temporary files (chart images created by the `ServletUtilities` class) and deletes them at the expiry of an `HttpSession`.

### 38.3 DisplayChart

#### 38.3.1 Overview

A servlet that displays a chart image from the temporary directory.

### 38.4 ServletUtilities

#### 38.4.1 Overview

A utility class for performing operations in a servlet environment. The methods in this class are all static.

#### 38.4.2 Saving Charts to Image Files

Several methods are provided to write charts to image files in the system's temporary directory, with automatic registration with the `ChartDeleter` class to remove the temporary files upon expiry of the session. If you don't want to use this temporary persistence mechanism, then you should use the `ChartUtilities` class directly.

To save a chart in a PNG file in the temporary directory (designated by the system property `java.io.tmpdir`):

```
➔ public static String saveChartAsPNG(JFreeChart chart, int width, int height,
    ChartRenderingInfo info, HttpSession session) throws IOException;
Saves a chart to a PNG image file in the temporary directory and returns the filename used.
```

The file is registered with a `ChartDeleter` instance that is linked to the specified session—this means the image file will be deleted when the session expires. The `info` parameter should be, if not `null`, a new instance of `ChartRenderingInfo`—it will be populated with information about the chart as it is drawn for the PNG file (this information could be used to create an HTML image map, for example). Note that the temporary file name prefix can be set using the `setTempFilePrefix()` method.

```
↳ public static String saveChartAsPNG(JFreeChart chart, int width, int height,
HttpSession session) throws IOException;
```

As for the previous method, with the `info` argument set as `null`.

Equivalent methods are provided to save charts in JPEG format, but you should note that:

- JPEG is a “lossy” format that is designed for photographic images—the results for most charts will be better if you use the PNG encoding;
- JPEG is supported by JFreeChart only when running on JRE 1.4.2 or later;

# Chapter 39

## Package: org.jfree.chart.title

### 39.1 Overview

This package contains classes that are used as chart titles and/or subtitles. The `JFreeChart` class maintains one chart title (an instance of `TextTitle`) plus a list of subtitles (which can be any subclass of `Title`).

When a chart is drawn, the title and/or subtitles will “grab” a rectangular section of the chart area in which to draw themselves. This reduces the amount of space for plotting data, so although there is no limit to the number of subtitles you can add to a chart, for practical reasons you need to keep the number reasonably low.

### 39.2 Events

When you add a `Title` to a `JFreeChart` instance, the chart registers itself as a `TitleChangeListener`. Any subsequent changes to the title will result in a `TitleChangeEvent` being sent to the chart. The chart then passes the event on to all its registered `ChartChangeListener`s. If the chart is displayed in a `ChartPanel`, the panel will receive a `ChartChangeEvent` and respond by repainting the chart.

### 39.3 CompositeTitle

#### 39.3.1 Overview

A chart title that contains other chart titles in some arrangement. This class provides some flexibility for displaying chart titles side-by-side or in other layouts.

#### 39.3.2 Usage

In `DualAxisDemo1.java`, the following code is used to add two legends, one on the left of the chart and the other on the right of the chart:

```
LegendTitle legend1 = new LegendTitle(plot.getRenderer(0));
legend1.setMargin(new RectangleInsets(2, 2, 2, 2));
legend1.setBorder(new BlockBorder());

LegendTitle legend2 = new LegendTitle(plot.getRenderer(1));
legend2.setMargin(new RectangleInsets(2, 2, 2, 2));
legend2.setBorder(new BlockBorder());

BlockContainer container = new BlockContainer(new BorderArrangement());
container.add(legend1, RectangleEdge.LEFT);
container.add(legend2, RectangleEdge.RIGHT);
container.add(new EmptyBlock(2000, 0));
CompositeTitle legends = new CompositeTitle(container);
legends.setPosition(RectangleEdge.BOTTOM);
chart.addSubtitle(legends);
```

### 39.3.3 Constructors

To create a new instance:

- `public CompositeTitle();`  
Creates a new (empty) title.
- `public CompositeTitle(BlockContainer container);`  
Creates a new title based on the specified container (which may be pre-populated with the titles contained by this instance).

### 39.3.4 Methods

The following methods allow you to access the container used to hold the titles for this composite title:

- `public BlockContainer getContainer();`  
Returns the container that holds the titles within this composite title. You can use this to add additional titles.
- `public void setTitleContainer(BlockContainer container);`  
Sets the container for this composite title, replacing any existing container.

### 39.3.5 Layout and Drawing Methods

The `JFreeChart` class will call the following methods to layout and draw the titles, you won't normally need to call these methods yourself:

- `public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);`  
Arranges the contents of the title within the given constraint, and returns the size of the title after the arrangement is done.
- `public void draw(Graphics2D g2, Rectangle2D area);`  
Draws the title within the given area.
- `public Object draw(Graphics2D g2, Rectangle2D area, Object params);`  
Draws the title within the given area. The parameters are ignored by this method, and the returned value is always `null` (this may change in the future).

### 39.3.6 Equality, Cloning and Serialization

This class overrides `equals()`:

- `public boolean equals(Object obj)`  
Tests this title for equality with an arbitrary object. This method returns true if and only if:
  - `obj` is an instance of `CompositeTitle`;
  - the container in `obj` is equal to the container for this composite title.

This class is cloneable and serializable.

## 39.4 DateTitle

### 39.4.1 Overview

A chart title that displays the current date (extends `TextTitle`). This class would normally be used to add the date to a chart as a subtitle.

### 39.4.2 Constructor

To create a new date title for the default locale:

- `public DateTitle(int style);`  
Creates a new date title with the specified style (defined by the `DateFormat` class). The title position is, by default, the lower right corner of the chart.

### 39.4.3 Methods

To set the date format:

```
➔ public void setDateFormat(int style, Locale locale);
Sets the date format to the given style and locale (the style is defined by constants in the
DateFormat class).
```

Other methods are inherited from the [TextTitle](#) class.

## 39.5 ImageTitle

### 39.5.1 Overview

A chart title that displays an image (extends [Title](#)).

### 39.5.2 Constructors

To create an image title:

```
➔ public ImageTitle(Image image);
Creates an image title. By default, the title is positioned at the top of the chart, and the image
is centered horizontally within the available space.
```

### 39.5.3 Methods

To change the image displayed by the image title:

```
➔ public void setImage(Image image);
Sets the image for the title and sends a TitleChangeEvent to all registered listeners.
```

Other methods are inherited from the [Title](#) class.

## 39.6 LegendGraphic

### 39.6.1 Overview

A graphical item, displayed as part of a legend item, that provides a visual link to a series in a chart. The [LegendTitle](#) class uses this class in the construction of a chart's legend.

### 39.6.2 Constructor

To create a new instance:

```
➔ public LegendGraphic(Shape shape, Paint fillPaint);
Creates a new graphic using the given shape and fillPaint.
```

### 39.6.3 Shape Attributes

To control whether or not the shape is visible:

```
➔ public boolean isShapeVisible();
Returns true if the shape is visible, and false otherwise.

➔ public void setShapeVisible(boolean visible);
Sets the visibility of the shape.
```

To access the shape itself:

```
➔ public Shape getShape();
Returns the shape for the legend graphic.
```

► public void setShape(Shape shape);  
 Sets the shape for the legend graphic.

To control whether or not the shape is filled:

► public boolean isShapeFilled();  
 Returns true if the shape is filled, and false otherwise.  
 ► public void setShapeFilled(boolean filled);  
 Sets the flag that controls whether or not the shape is filled.  
 ► public Paint getFillPaint();  
 Returns the paint used to fill the shape.  
 ► public void setFillPaint(Paint paint);  
 Sets the paint used to fill the shape.

As of version 1.0.4, this class supports the use of a GradientPaint for the fill paint, by recording a transformer for the gradient coordinates:<sup>1</sup>

► public GradientPaintTransformer getFillPaintTransformer(); [1.0.4]  
 Returns the gradient paint transformer for the fill paint.  
 ► public void setFillPaintTransformer(GradientPaintTransformer transformer); [1.0.4]  
 Sets the gradient paint transformer for the fill paint.

To control whether or not the shape outline is drawn:

► public boolean isShapeOutlineVisible();  
 Returns true if the shape outline is displayed, and false otherwise.  
 ► public void setShapeOutlineVisible(boolean visible);  
 Sets the flag that controls whether or not the shape outline is drawn.  
 ► public Paint getOutlinePaint();  
 Returns the paint used to draw the shape outline.  
 ► public void setOutlinePaint(Paint paint);  
 Sets the paint used to draw the shape outline.  
 ► public Stroke getOutlineStroke();  
 Returns the stroke used to draw the shape outline.  
 ► public void setOutlineStroke(Stroke stroke);  
 Sets the stroke used to draw the shape outline.  
 ► public RectangleAnchor getShapeAnchor(RectangleAnchor anchor);  
 Returns the anchor point for the shape.  
 ► public void setShapeAnchor(RectangleAnchor anchor);  
 Sets the anchor point for the shape.  
 ► public RectangleAnchor getShapeLocation();  
 Returns the shape location.  
 ► public void setShapeLocation(RectangleAnchor location);  
 Sets the shape location.

### 39.6.4 Line Attributes

To control whether or not a line is drawn for the legend graphic:

► public boolean isLineVisible();  
 Returns true if a line is drawn for this legend graphic, and false otherwise.  
 ► public void setLineVisible(boolean visible);  
 Sets the flag that controls whether or not a line is drawn for this legend graphic.

---

<sup>1</sup>Only some renderers support this.

To control the shape of the line:

► `public Shape getLine();`

Returns the shape used for the line. Usually, this is a `Line2D`, but it is possible to use another shape, such as a `GeneralPath` to draw the line.

► `public void setLine(Shape line);`

Sets the shape used for the line. Typically this will be a `Line2D`, but you can use other `Shape` instances (for example, a `GeneralPath`). Note that, for alignment purposes, the (0, 0) coordinate should lie approximately at the center of the line.

► `public Paint getLinePaint();`

Returns the `Paint` used to display the line.

► `public void setLinePaint(Paint paint);`

Sets the `Paint` used to display the line.

► `public Stroke getLineStroke();`

Returns the `Stroke` used to display the line.

► `public void setLineStroke(Stroke stroke);`

Sets the `Stroke` used to display the line.

### 39.6.5 Other Methods

The following methods are used by JFreeChart for layout and rendering:

► `public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);`  
Arranges the graphics and returns its size.

► `public void draw(Graphics2D g2, Rectangle2D area);`  
Draws the graphic within the specified rectangle.

► `public Object draw(Graphics2D g2, Rectangle2D area, Object params);`  
Draws the graphic within the specified rectangle.

### 39.6.6 Equals, Cloning and Serialization

To check this legend for equality with another object:

► `public boolean equals(Object obj);`  
Tests this title for equality with an arbitrary object.

This class is `Cloneable` and `Serializable`.

#### See Also

[LegendItemBlockContainer](#).

## 39.7 LegendItemBlockContainer

### 39.7.1 Overview

A container used internally by JFreeChart to represent one item in a legend. This is a subclass of [BlockContainer](#).

### 39.7.2 Constructors

To create a:

► `public LegendItemBlockContainer(Arrangement arrangement, int dataset, int series);`

Creates a new container. The dataset and series indices are used to identify the source for this legend item.

### 39.7.3 Methods

► `public int getDatasetIndex();`

Returns the index of the dataset that this legend item represents. This is copied over to the entity that is (optionally) created when this item is drawn.

► `public int getSeriesIndex();`

Returns the index of the series that this legend item represents. This is copied over to the entity that is (optionally) created when this item is drawn.

► `public Object draw(Graphics2D g2, Rectangle2D area, Object params);`

Draws the container (which represents a legend item). This method is called by JFreeChart, you won't normally need to call it directly.

### 39.7.4 Notes

This class is used internally by the `LegendTitle` class—you won't normally need to interact with this class directly.

#### See Also

[LegendGraphic](#), [LegendTitle](#).

## 39.8 LegendTitle

### 39.8.1 Overview

A legend displays labels for the series in a chart, usually along with a small graphic item that identifies the series (by color and/or style). For example, figure 39.1 shows a chart with two legends, one on the left showing the colours for series “S1”, “S2” and “S3”, and the other on the right showing the colour for series “S4”.

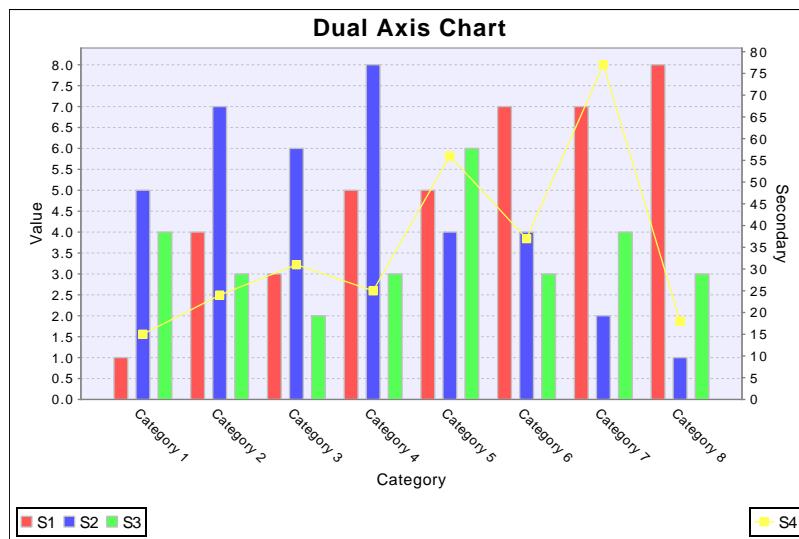


Figure 39.1: A chart with two legends (see `DualAxisDemo1.java`)

A legend is added to a chart using the `addLegend()` method in the `JFreeChart` class. This does the same thing as calling `addSubtitle()`, since the legend is treated in the same way as any subtitle (see `Title`). It is even possible to add more than one legend to a chart, and configure each to display different subsets of the series in the chart (for example, see `DualAxisDemo1.java` in the JFreeChart demo collection).

### 39.8.2 Usage

#### Adding a Legend

A chart typically has just one legend (although it is possible to add more legends to a chart, if necessary). If you create your chart by calling one of the methods in the `ChartFactory` class, you can get a default legend simply by setting the `legend` argument to `true`.

#### Controlling the Legend Position

The legend position can be specified by calling the `setPosition()` method defined in the `Title` class. Assuming that your chart has a single legend (the most common case), you can change the location of the legend as follows:

```
LegendTitle legend = chart.getLegend();
legend.setPosition(RectangleEdge.BOTTOM);
```

### 39.8.3 Constructors

To create a new legend:

- `public LegendTitle(LegendItemSource source);`  
Creates a new legend that uses the specified `source` for legend items. Typically, the `source` is a plot instance, in which case the legend will display all the series for the plot. It is possible to display a subset of the series in the plot, by specifying a single renderer as the `source`.
- `public LegendTitle(LegendItemSource source, Arrangement hLayout, Arrangement vLayout);`  
Creates a new legend that uses the specified `source` for legend items. The `hLayout` is used for layout when the legend is at the top or bottom of the chart, and the `vLayout` is used when the legend is at the left or right of the chart.

### 39.8.4 Legend Item Sources

The legend uses one or more sources for its legend items. A source is any class that implements the `LegendItemSource` interface—this includes all plots and renderers in JFreeChart. The legend items are fetched each time the chart is drawn (or redrawn), which allows for the fact that a dataset change may alter the items that should be displayed in the legend.

- `public LegendItemSource[] getSources();`  
Returns an array of the sources for the legend. The array may be empty, but is never `null`.
- `public void setSources(LegendItemSource[] sources);`  
Sets the sources for the legend. A `null` argument will cause an exception.

By default, the legend will use the plot for the source, which results in all series being displayed in the legend. You'll only need to use the `setSources()` method if you want to display a legend (or several legends) containing only a subset of the series in the chart.

### 39.8.5 Legend Appearance and Layout

The legend background is controlled with the following methods:

- `public Paint getBackgroundPaint();`  
Returns the background paint for the legend. The default value is `null`.
- `public void setBackgroundPaint(Paint paint);`  
Sets the background paint for the legend, and sends a `TitleChangeEvent` to all registered listeners.  
If this is `null`, the legend will be transparent.

### 39.8.6 Legend Item Appearance and Layout

The legend will typically contain one legend item for each series displayed in a chart. The item has a small graphic that identifies the series in the chart, and a label that corresponds to the series name. A number of methods are provided to customise the appearance of these legend items.

To modify the font used to display the legend item labels:

► public Font getItemFont();

Returns the font (never null) for the legend item labels. The default font is `SansSerif PLAIN 10`.

► public void setItemFont(Font font);

Sets the font for the legend item labels and sends a `TitleChangeEvent` to all registered listeners. A null argument will cause an exception.

For example:

```
LegendTitle legend = chart.getLegend();
if (legend != null) {
    legend.setItemFont(new Font("Dialog", Font.PLAIN, 18));
}
```

To set the padding around the item labels:

► public RectangleInsets getItemLabelPadding();

Returns the padding (never null) for the item labels. The default is (2.0, 2.0, 2.0, 2.0).

► public void setItemLabelPadding(RectangleInsets padding);

Sets the padding for the item labels and sends a `TitleChangeEvent` to all registered listeners. A null argument will cause an exception.

To control the location of the item graphic relative to its text:

► public RectangleEdge getLegendItemGraphicEdge();

Returns the location (never null) of the item graphic relative to its text.

► public void setLegendItemGraphicEdge(RectangleEdge edge);

Sets the location of the item graphic relative to its text and sends a `TitleChangeEvent` to all registered listeners. A null argument will cause an exception.

The location of the item graphic within its rectangle is determined by two attributes, the anchor point and the location. The anchor point is a point on the item graphic that can be aligned with the location point.

► public RectangleAnchor getLegendItemGraphicAnchor();

Returns the anchor (never null), which determines a point relative to the bounding box of the legend item graphic that is used for alignment.

► public void setLegendItemGraphicAnchor(RectangleAnchor anchor);

Sets the anchor, which determines a point relative to the bounding box of the legend item graphic that is used for alignment.

► public RectangleAnchor getLegendItemGraphicLocation();

Returns a location, relative to the bounding box of the legend item. The legend graphic will be aligned relative to this point.

► public void setLegendItemGraphicLocation(RectangleAnchor anchor);

Sets the location, which defines a point relative to the bounding box of the legend item.

The padding around the legend item graphic is controlled with the following methods:

► public RectangleInsets getLegendItemGraphicPadding();

Returns the padding around the legend item graphic.

► public void setLegendItemGraphicPadding(RectangleInsets padding);

Sets the padding around the legend item graphic.

### 39.8.7 The Legend Wrapper

A legend wrapper provides a mechanism to add one or more items (such as a title and subtitle) to the legend, while still allowing for the automatic layout of the legend items.

► `public void setWrapper(BlockContainer wrapper);`

Sets the wrapper for the legend title. One of the blocks contained by `wrapper` should be the item container which you can obtain from the `getItemContainer()` method.

► `public BlockContainer getItemContainer();`

Returns the container that is populated with legend items each time the legend is drawn.

A demo application (`LegendWrapperDemo1.java`) that shows how to use the wrapper facility is included in the JFreeChart demo distribution

### 39.8.8 Other Methods

The remaining methods in the `LegendTitle` class are mostly used internally:

► `protected void fetchLegendItems();`

Fetches the legend items from the sources defined for the legend. This will be done every time the legend is drawn.

► `protected Block createLegendItemBlock(LegendItem item);`

Creates a block representing the specified `item`. This code is contained in a separate method to allow the possibility of overriding it to change the appearance of individual legend items.

► `public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);`

Arranges the contents of the legend subject to the specified constraint and returns the size of the legend.

► `public void draw(Graphics2D g2, Rectangle2D area);`

Draws the legend within the specified area.

► `public Object draw(Graphics2D g2, Rectangle2D area, Object params);`

Draws the legend within the specified area.

### 39.8.9 Equals, Cloning and Serialization

To check this legend for equality with another object:

► `public boolean equals(Object obj);`

Tests this title for equality with an arbitrary object.

This class is `Cloneable` and `Serializable`.

#### See Also

[LegendItemSource](#).

## 39.9 PaintScaleLegend

### 39.9.1 Overview

A chart title that displays a `PaintScale` (extends `Title`). This is used to illustrate the color scale used by a renderer like the `XYBlockRenderer`.

*This class was first introduced in JFreeChart version 1.0.4.*

### 39.9.2 Constructors

To create a new instance:

► `public PaintScaleLegend(PaintScale scale, ValueAxis axis); [1.0.4]`

Creates a new legend for the given scale. The supplied axis is used to show the numerical range for the scale.

### 39.9.3 Methods

The following methods are defined:

- **public PaintScale getScale(); [1.0.4]**  
Returns the paint scale displayed in this legend.
- **public void setScale(PaintScale scale); [1.0.4]**  
Sets the paint scale to be displayed in the legend and sends a [TitleChangeEvent](#) to all registered listeners.
- **public ValueAxis getAxis(); [1.0.4]**  
Returns the axis that shows the numerical range of the paint scale.
- **public void setAxis(ValueAxis axis); [1.0.4]**  
Sets the axis that shows the numerical range of the paint scale and sends a [TitleChangeEvent](#) to all registered listeners.
- **public AxisLocation getAxisLocation(); [1.0.4]**  
Returns the location of the axis relative to the legend.
- **public void setAxisLocation(AxisLocation location); [1.0.4]**  
Sets the location of the axis relative to the legend and sends a [TitleChangeEvent](#) to all registered listeners.
- **public double getAxisOffset(); [1.0.4]**  
Returns the offset (in Java2D units) between the color strip and the axis. The default value is 0.0.
- **public void setAxisOffset(double offset); [1.0.4]**  
Sets the offset (in Java2D units) between the color strip and the axis, then sends a [TitleChangeEvent](#) to all registered listeners.
- **public double getStripWidth(); [1.0.4]**  
Returns the width of the color strip in Java2D units. The default value is 15.0.
- **public void setStripWidth(double width); [1.0.4]**  
Sets the width of the color strip in Java2D units and sends a [TitleChangeEvent](#) to all registered listeners.
- **public boolean isStripOutlineVisible(); [1.0.4]**  
Returns a flag that controls whether or not an outline is drawn for the color strip. The default value is `false`.
- **public void setStripOutlineVisible(boolean visible); [1.0.4]**  
Sets a flag that controls whether or not an outline is drawn for the color strip and sends a [TitleChangeEvent](#) to all registered listeners.
- **public Paint getStripOutlinePaint(); [1.0.4]**  
Returns the paint used to draw the outline for the color strip (if the outline is visible).
- **public void setStripOutlinePaint(Paint paint); [1.0.4]**  
Sets the paint used to draw the outline for the color strip (if the outline is visible) and sends a [TitleChangeEvent](#) to all registered listeners.
- **public Stroke getStripOutlineStroke(); [1.0.4]**  
Returns the stroke used to draw the outline for the color strip (if the outline is visible).
- **public void setStripOutlineStroke(Stroke stroke); [1.0.4]**  
Sets the stroke used to draw the outline for the color strip (if the outline is visible) and sends a [TitleChangeEvent](#) to all registered listeners.
- **public Paint getBackgroundPaint(); [1.0.4]**  
Returns the background paint for the legend. The default value is `null`, which means the legend background is transparent.
- **public void setBackgroundPaint(Paint paint); [1.0.4]**  
Sets the background paint for the legend (`null` is permitted) and sends a [TitleChangeEvent](#) to all registered listeners.

Other methods are inherited from the [Title](#) class.

### 39.9.4 Other Methods

The following methods are used by JFreeChart when drawing the legend:

- `public Size2D arrange(Graphics2D g2, RectangleConstraint constraint); [1.0.4]`  
Performs a layout on the legend, subject to the supplied `constraint`.
- `public void draw(Graphics2D g2, Rectangle2D area); [1.0.4]`  
Draws the legend within the specified `area`.
- `public Object draw(Graphics2D g2, Rectangle2D area, Object params); [1.0.4]`  
Draws the legend within the specified `area`.

### 39.9.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj); [1.0.4]`  
Tests this legend for equality with an arbitrary object.

Instances of this class are `Cloneable` (also implements `PublicCloneable`) and `Serializable`.

### 39.9.6 Notes

Some points to note:

- several demos (`XYBlockChartDemo1-3.java`) are included in the JFreeChart demo collection.

#### See Also

[PaintScale](#), [XYBlockRenderer](#).

## 39.10 TextTitle

### 39.10.1 Overview

A chart title that displays a text string (extends `Title`). Long titles automatically wrap to the next line, and you can also specify line breaks by inserting a newline character in the title's text.

### 39.10.2 Constructors

To create a text title for a chart:

- `public TextTitle();`  
Equivalent to `TextTitle("")`—see the next constructor.
- `public TextTitle(String text);`  
Creates a chart title using the specified text. By default, the title will be positioned at the top of the chart, centered horizontally. The font defaults to `SansSerif`, 12pt bold and the color defaults to black.

There are other constructors that provide more control over the attributes of the `TextTitle`.

- `public TextTitle(String text, Font font);`  
Creates a new title with the specified text and font, and default attributes.
- `public TextTitle(String text, Font font, Paint paint, RectangleEdge position, HorizontalAlignment horizontalAlignment, VerticalAlignment verticalAlignment, RectangleInsets padding);`  
Creates a new text title with the specified attributes.

### 39.10.3 General Attributes

This class inherits attributes from the [Title](#) class, and adds a number of its own.

To control the title text:

```
► public String getText();
Returns the text displayed in the title.

► public void setText(String text);
Sets the text for the title and sends a TitleChangeEvent to all registered listeners. The text string may contain explicit  
n characters, which force a line break in the title. If text is null, this method throws an  
IllegalArgumentException.
```

To set the font for the title:

```
► public Font getFont();
Returns the font used to display the title. The default value is Font("SansSerif", Font.BOLD, 12). This method never returns null.

► public void setFont(Font font);
Sets the font for the title and sends a TitleChangeEvent to all registered listeners. If font is null, this method throws an IllegalArgumentException.
```

To set the color of the title:

```
► public Paint getPaint();
Returns the paint used to display the title text. The default value is Color.black. This method never returns null.

► public void setPaint(Paint paint);
Sets the paint used to display the title text and sends a TitleChangeEvent to all registered listeners. If paint is null, this method throws an IllegalArgumentException.
```

To control the text alignment:

```
► public HorizontalAlignment getTextAlignment();
Returns the horizontal text alignment. The default value is HorizontalAlignment.CENTER. Note that this indicates the text alignment within the title's bounds.

► public void setTextAlignment(HorizontalAlignment alignment);
Sets the alignment of the text within the title bounds, and sends a TitleChangeEvent to all registered listeners. The alignment argument must be one of:


- HorizontalAlignment.LEFT;
- HorizontalAlignment.CENTER;
- HorizontalAlignment.RIGHT.

```

To control the background paint for the title:

```
► public Paint getBackgroundPaint();
Returns the paint used to fill the background area within the title's bounds. This method can return null, in which case the title's background is transparent.

► public void setBackgroundPaint(Paint paint);
Sets the paint used to fill the background area, and sends a TitleChangeEvent to all registered listeners.
```

You can specify the (optional) tool tip text for the title:

```
► public String getToolTipText();
Returns the text that will be displayed as the tool tip for this title. The default value is null (no tool tip).

► public void setToolTipText(String text);
Sets the text that will be displayed as the tool tip for this title, and sends a TitleChangeEvent to all registered listeners. The text argument can be null, which means that no tool tip will be displayed for the title.
```

Similarly, you can specify a URL for the title (typically used in HTML image maps only):

► `public String getURLText();`

Returns the URL text for the title. The default value is `null`.

► `public void setURLText(String text);`

Sets the URL text for the title and sends a `TitleChangeEvent` to all registered listeners. The `text` argument can be `null`.

There is a flag that controls whether the title bounds fit the title text or expand to fit any remaining space:

► `public boolean getExpandToFitSpace();`

Returns the flag that controls whether or not the title expands to fit the available space. The default value is `false`.

► `public void setExpandToFitSpace(boolean expand);`

Sets the flag that controls whether or not the title bounds expand to match the available space, and sends a `TitleChangeEvent` to all registered listeners.

### 39.10.4 Other Methods

The following method is called by the `JFreeChart` class to draw the chart title:

► `public void draw(Graphics2D g2, Rectangle2D area);`

Draws the title onto a graphics device, to occupy the specified area.

There are additional methods inherited from the `Title` class.

### 39.10.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`

Tests this title for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 39.10.6 Notes

Some points to note:

- the title string can contain any characters from the Unicode character set. However, you need to ensure that the `Font` that you use to display the title actually supports the characters you want to display. Most fonts do not support the full range of Unicode characters, but this website has some information about fonts that you might be able to use:

<http://www.css.de/slovo/unifonts.htm>

- to set a border around the title, use the inherited `setFrame()` method.

### See Also

[Title](#).

## 39.11 Title

### 39.11.1 Overview

The base class for all chart titles. Several concrete sub-classes have been implemented, including: `TextTitle`, `DateTitle`, `LegendTitle` and `ImageTitle`. All titles inherit margin, border and padding attributes from the `AbstractBlock` class.

### 39.11.2 Constructors

This is an abstract class. The following constructors are available for subclasses to use:

- `protected Title();`  
Creates a title with default attributes.
- `protected Title(RectangleEdge position, HorizontalAlignment horizontalAlignment, VerticalAlignment verticalAlignment);`  
Creates a title at the specified position using the given alignments.
- `protected Title(RectangleEdge position, HorizontalAlignment horizontalAlignment, VerticalAlignment verticalAlignment, RectangleInsets padding);`  
Creates a new Title with the specified position, alignment and padding. All arguments must be non-null.

### 39.11.3 Methods

To control the position of the title:

- `public RectangleEdge getPosition();`  
Returns the position of the title (never null).
- `public void setPosition(RectangleEdge position);`  
Sets the position for the title (null not permitted). Following the change, a `TitleChangeEvent` is sent to all registered listeners (including, by default, the `JFreeChart` object that the title belongs to).

Within the rectangular area allocated for the title, you can specify the horizontal alignment:

- `public void setHorizontalAlignment(HorizontalAlignment alignment);`  
Sets the horizontal alignment for the title (null not permitted). Following the change, a `TitleChangeEvent` is sent to all registered listeners.

Similarly, you can specify the vertical alignment:

- `public void setVerticalAlignment(VerticalAlignment alignment);`  
Sets the vertical alignment for the title (null not permitted). Following the change, a `TitleChangeEvent` is sent to all registered listeners.

### 39.11.4 Drawing Titles

Subclasses should implement the following method to draw themselves within the specified `area`:

- `public abstract void draw(Graphics2D g2, Rectangle2D area);`  
Draws the title. Subclasses must implement this method.

### 39.11.5 Event Notification

Most changes to a title will generate a `TitleChangeEvent` which will be sent to all registered listeners. By default, the chart that a title belongs to will be set up to receive these change events and typically you won't need to register any other listeners. However, this can be done with the following methods:

- `public void addChangeListener(TitleChangeListener listener);`  
Registers a listener to receive change events generated by the title.
- `public void removeChangeListener(TitleChangeListener listener);`  
Deregisters a listener so that it no longer receives change events generated by the title.

Subclasses change send a change event to all registered listeners using the following method:

- `protected void notifyListeners(TitleChangeEvent event);`  
Sends the `method` to all registered listeners.

There is a flag that can be used to temporarily disable change events generated by the title:

► `public boolean getNotify();`

Returns the flag that indicates whether or not listeners should be notified when any title attribute is changed.

► `public void setNotify(boolean flag);`

Sets the flag that indicates whether or not listeners should be notified when any title attribute is changed. When this flag is set to `true`, a change event is generated immediately.

### 39.11.6 Equals, Cloning and Serialization

To test a title for equality with an arbitrary object:

► `public boolean equals(Object obj);`

Returns `true` if this title is equal to the specified object.

All titles should be `Cloneable` and `Serializable`, otherwise charts using titles will fail to clone and serialize.

► `public Object clone() throws CloneNotSupportedException;`

Returns a clone of the title.

### 39.11.7 Notes

Some points to note:

- the original version of this class was written by David Berry. I've since made a few changes to the original version, but the idea for allowing a chart to have multiple titles came from David.
- the `JFreeChart` class implements the `TitleChangeListener` interface, and receives notification whenever a chart title is changed (this, in turn, triggers a `ChartChangeEvent` which usually results in the chart being redrawn).
- this class implements `Cloneable`, which is useful when editing title properties because you can edit a copy of the original, and then either apply the changes or cancel the changes.

# Chapter 40

## Package: org.jfree.chart.urls

### 40.1 Overview

This package contains support for URL generation for HTML image maps. URLs are generated (if they are required) at the point that a renderer draws the visual representation of a data item. The renderer queries a *URL generator* via one of the following interfaces:

- `CategoryURLGenerator`;
- `PieURLGenerator`;
- `XYURLGenerator`;
- `XYZURLGenerator`;

JFreeChart provides standard implementations for each of these interfaces. In addition, you can easily write your own implementation and take full control of the URLs that are generated within your image map.

### 40.2 CategoryURLGenerator

#### 40.2.1 Overview

A *category URL generator* is used to generate a URL for each data item in a `CategoryPlot`. The generator is associated with the plot's renderer (an instance of `CategoryItemRenderer`) and the URLs are used when you create an HTML image map for a chart image.

#### 40.2.2 Methods

This method returns a URL for a specific data item:

```
→ public String generateURL(CategoryDataset dataset, int series, int category);
```

Returns a URL for the specified data item. The `series` is the row index, and the `category` is the column index for the `dataset`. A typical implementation of this interface will use these arguments to construct an appropriate URL for the data item, but of course some implementations may choose to ignore these arguments. Note that JFreeChart calls this method during chart rendering, you won't normally call it directly yourself.

#### 40.2.3 Notes

Some points to note:

- the `StandardCategoryURLGenerator` class is the only implementation of this interface provided in the JFreeChart class library, but you can add your own implementation(s);

- the `ChartUtilities` class contains code for writing HTML image maps.

## 40.3 CustomPieURLGenerator

### 40.3.1 Overview

A URL generator where the URLs for each pie section are manually specified in advance. This class implements the `PieURLGenerator` interface.

### 40.3.2 Constructors

To create a new generator:

```
↳ public CustomPieURLGenerator();
Creates a new generator, initially with no URL entries assigned.
```

### 40.3.3 Methods

To get a URL for a pie section:

```
↳ public String generateURL(PieDataset dataset, Comparable key, int pieIndex);
Returns a URL from the generator's lookup table.

↳ public int getListCount();
Returns the number of items (Map instances) in the URL list (see the addURLs() method). Each map contains URLs for the sections in one pie chart (for regular pie charts, only one map is required, but for use with the MultiplePiePlot class multiple maps are supported).

↳ public int getURLCount(int list);
Returns the number of URLs in the specified map.

↳ public String getURL(Comparable key, int pieItem);
Returns the URL for a section in the specified pie chart. JFreeChart will call this method during chart rendering—you won't normally need to call this method yourself.

↳ public void addURLs(Map urlMap);
Adds a collection of URLs for one pie chart.1
```

### 40.3.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
↳ public boolean equals(Object obj);
Tests this generator for equality with an arbitrary object.

↳ public Object clone() throws CloneNotSupportedException;
Returns a clone of the generator.
```

### 40.3.5 Notes

Some points to note:

- the API for this class could be filled out a little, particularly in the area of added and removing maps;

---

<sup>1</sup>This method has several limitations. Once you have added a map, you can't retrieve it. In addition, you can't specify the index for the map (that is, which pie chart it applies to if the generator is being used with a `MultiplePiePlot`).

## 40.4 CustomXYURLGenerator

### 40.4.1 Overview

A URL generator that uses custom strings as the URL for each item in an `XYDataset`. This class implements the `XYURLGenerator` interface.

## 40.5 PieURLGenerator

### 40.5.1 Overview

An interface defining the API that a caller (typically the `PiePlot` class) can use to obtain a URL for a pie section. The resulting URL is used in the creation of HTML image maps.

There are two implementations of this interface provided in JFreeChart:

- `StandardPieURLGenerator`;
- `CustomPieURLGenerator`.

You are, of course, free to write your own implementation.

### 40.5.2 Usage

In the `PiePlot` class, a URL generator can be assigned to the plot using the `setURLGenerator(PieURLGenerator)` method.

### 40.5.3 Methods

This method returns a URL for a specific data item:

```
➔ public String generateURL(PieDataset dataset, Comparable key, int pieIndex);
```

Returns a URL for the specified data item. The `key` is the key for the current section within the dataset, and the `pieIndex` is used when multiple pie plots are included within one chart (see the `MultiplePiePlot` class).

### 40.5.4 Notes

Some points to note:

- by convention, all classes that implement this interface should be either:
  - immutable; or
  - implement the `PublicCloneable` interface.

This provides a mechanism for a referring class to determine whether or not it needs to clone the generator, and access to the `clone()` method in the case that the generator is cloneable.

- the `ImageMapUtilities` class contains methods to help with writing HTML image maps.

## 40.6 StandardCategoryURLGenerator

### 40.6.1 Overview

A class that generates a URL for a data item in a `CategoryPlot`. By default, this generator will create URLs in the format:

```
index.html?series=<serieskey>&category=<categorykey>
```

...where `<serieskey>` and `<categorykey>` are replaced with values from the dataset. This class implements the `CategoryURLGenerator` interface.<sup>2</sup>

#### 40.6.2 Usage

If you create a chart using the `ChartFactory` class, you can ask for a default URL generator to be installed in the renderer just by setting the `urls` flag (a parameter for most chart creation methods) to `true`.

Alternatively, you can create a new generator and register it with the renderer (replacing the existing generator, if there is one) as follows:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
CategoryItemRenderer renderer = plot.getRenderer();
CategoryURLGenerator generator = new StandardCategoryURLGenerator(
    "index.html", "series", "category");
renderer.setItemURLGenerator(generator);
```

Set the URL generator to `null` if you do not require URLs to be generated.

#### 40.6.3 Constructors

To create a new generator:

```
► public StandardCategoryURLGenerator();
Creates a new generator with default values:


- prefix: index.html;
- seriesParameterName: series;
- categoryParameterName: category.


► public StandardCategoryURLGenerator(String prefix);
Creates a new generator with the given prefix and default values for the other attributes:


- seriesParameterName: series;
- categoryParameterName: category.


► public StandardCategoryURLGenerator(String prefix,
String seriesParameterName, String categoryParameterName);
Creates a new generator with the specified attributes.
```

#### 40.6.4 Methods

The following method is called by the renderer to generate the URL for a single data item in a chart:

```
► public String generateURL(CategoryDataset dataset, int series, int category)
Returns a string that will be used as the URL for the specified data item.
```

#### 40.6.5 Notes

Some points to note:

- this class is the only implementation of the `CategoryURLGenerator` interface that is provided by JFreeChart, but you can easily write your own implementation.

---

<sup>2</sup>Note that the use of `&` for the parameter separator is to ensure compliance with XHTML 1.0.

## 40.7 StandardPieURLGenerator

### 40.7.1 Overview

A default URL generator for use when creating HTML image maps for pie charts. An instance of this class can generate a URL for each section of a pie chart, in the form:

```
index.html?category=<sectionkey>&pieIndex=<pieIndex>
```

...where `<sectionkey>` and `<pieIndex>` are replaced with appropriate values. This class implements the [PieURLGenerator](#) interface, and instances of this class are immutable.

### 40.7.2 Usage

If you create a new pie chart using the [ChartFactory](#) class, you can request that a default URL generator be installed simply by passing `true` as the value of the `urls` flag. Alternatively, you can create a new instance and assign it at any time as follows:

```
PiePlot plot = (PiePlot) chart.getPlot();
PieURLGenerator generator = new StandardPieURLGenerator();
plot.setURLGenerator(generator);
```

Set the URL generator to `null` if you do not require URLs to be generated.

### 40.7.3 Constructor

To create a new generator:

- ▶ `public StandardPieURLGenerator();`  
Equivalent to `this("index.html")`—see the next constructor.
- ▶ `public StandardPieURLGenerator(String prefix);`  
Equivalent to `this(prefix, "category")`—see the next constructor.
- ▶ `public StandardPieURLGenerator(String prefix, String categoryParameterName);`  
Equivalent to `this(prefix, categoryParameterName, "pieIndex")`—see the next constructor.
- ▶ `public StandardPieURLGenerator(String prefix, String categoryParameterName,
String indexParameterName);`  
Creates a new generator with the given attributes.

The default values for unspecified attributes are:

- `prefix: index.html;`
- `categoryParameterName: category;`
- `indexParameterName: pieIndex.`

### 40.7.4 Methods

To generate a URL for a pie section:

```
▶ public String generateURL(PieDataset dataset, Comparable key, int pieIndex);  
Returns a string that will be used as the URL for the specified pie section. This method is called by JFreeChart—you won't normally need to call it directly.
```

### 40.7.5 Equals, Cloning and Serialization

This class overrides the equals method:

```
➔ public boolean equals(Object obj);
```

Tests this generator for equality with an arbitrary object. This method returns true if and only if:

- obj is not null;
- obj is an instance of StandardPieURLGenerator;
- this generator and obj have equal field values.

Instances of this class are `Serializable` but not `Cloneable` (cloning is not necessary because instances of this class are immutable).

### 40.7.6 Notes

Some points to note:

- the pie index in the URL comes from the `getIndex()` method in the `PiePlot` class. Typically this has a non-zero value only when using the `MultiplePiePlot` class.

## 40.8 StandardXYURLGenerator

### 40.8.1 Overview

A standard URL generator for use with an `XYItemRenderer`. The generator is used when creating HTML image maps, and defines a URL for each data item that the renderer draws. This class implements the `XYURLGenerator` interface.

### 40.8.2 Constructors

This class defines three constructors:

```
➔ public StandardXYURLGenerator();
```

Equivalent to `StandardXYURLGenerator("index.html")`—see the constructor below.

```
➔ public StandardXYURLGenerator(String prefix);
```

Equivalent to `StandardXYURLGenerator()`—see the constructor below.

```
➔ public StandardXYURLGenerator(String prefix, String seriesParameterName,
String itemParameterName);
```

Creates a new URL generator.

### 40.8.3 Method

The following method is called by the renderer (to generate a URL for an item in the dataset), you won't normally call this method directly:

```
➔ public String generateURL(XYDataset dataset, int series, int item);
```

Returns a URL for an item in the specified dataset.

### 40.8.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
```

Tests this generator for equality with an arbitrary object. This method returns true if and only if:

- obj is not null;

- `obj` is an instance of `XYURLGenerator`;
- `obj` has the same attributes as this generator.

Instances of this class are `Serializable` but not `Cloneable` (cloning isn't necessary because instances of this class are immutable).

## 40.9 StandardXYZURLGenerator

### 40.9.1 Overview

A URL generator that creates URLs for the items in an `XYZDataset`.

## 40.10 TimeSeriesURLGenerator

### 40.10.1 Overview

A URL generator that creates URLs for the items in an `XYDataset`. The x-values from the dataset are evaluated as "milliseconds since midnight 1-Jan-1970" (as for `java.util.Date`) and converted to date format.

## 40.11 URLUtilities

### 40.11.1 Overview

A utility class for working with URLs. At present, this class provides a single method for encoding URLs that delegates to different methods in the Java runtime depending on the current runtime version.

### 40.11.2 Method

```
➔ public static String encode(String s, String encoding);
Delegates to java.net.URLEncoder.encode(s, encoding) if the underlying JRE is version 1.4 or
later, otherwise reverts to the deprecated java.net.URLEncoder.encode(s) method.
```

### 40.11.3 Notes

In the future, when JFreeChart support for JDK 1.3.1 is dropped,<sup>3</sup> this class will no longer be required.

## 40.12 XYURLGenerator

### 40.12.1 Overview

An *XY URL generator* is used by a `XYItemRenderer` to generate URLs for use in HTML image maps.

### 40.12.2 Methods

This method returns a URL for a specific data item:

```
➔ public String generateURL(XYDataset data, int series, int item);
Returns a URL for the specified data item.
```

---

<sup>3</sup>There is no date for this yet.

### 40.12.3 Notes

Some points to note:

- the `StandardXYZURLGenerator` class is the only implementation of this interface provided in the JFreeChart class library.
- the `ChartUtilities` class contains methods for writing HTML image maps.

## 40.13 XYZURLGenerator

### 40.13.1 Overview

An *XYZ URL generator* is used by a `XYItemRenderer` to generate URLs for use in HTML image maps.

### 40.13.2 Methods

This method returns a URL for a specific data item:

```
→ public String generateURL(XYDataset data, int series, int item);  
Returns a URL for the specified data item.
```

### 40.13.3 Notes

Some points to note:

- the `StandardXYZURLGenerator` class is the only implementation of this interface provided in the JFreeChart class library.
- the `ChartUtilities` class contains methods for writing HTML image maps.

# Chapter 41

## Package: org.jfree.chart.util

### 41.1 Overview

This package contains utility classes used by JFreeChart. These classes might be candidates for a future release of JCommon, but to ensure that our dependency on JCommon is for version 1.0.0 or later, we keep this code here for now.

### 41.2 RelativeDateFormat

#### 41.2.1 Overview

A custom date formatter that shows elapsed time in hours, minutes and seconds (relative to some predefined point in time). You can use this formatter to format the labels on an axis that shows elapsed time—see, for example, `RelativeDateFormatDemo1.java`. **This class was first introduced in JFreeChart 1.0.3.**

#### 41.2.2 Usage

The following sample code illustrates the use of this date formatter:

```
public static void main(String[] args) {
    GregorianCalendar c0 = new GregorianCalendar(2006, 10, 1, 0, 0, 0);
    GregorianCalendar c1 = new GregorianCalendar(2006, 10, 1, 11, 37, 43);
    c1.set(Calendar.MILLISECOND, 123);

    System.out.println("Default: ");
    RelativeDateFormat rdf = new RelativeDateFormat(c0.getTimeInMillis());
    System.out.println(rdf.format(c1.getTime()));
    System.out.println();

    System.out.println("Hide milliseconds: ");
    rdf.setSecondFormatter(new DecimalFormat("0"));
    System.out.println(rdf.format(c1.getTime()));
    System.out.println();

    System.out.println("Show zero day output: ");
    rdf.setShowZeroDays(true);
    System.out.println(rdf.format(c1.getTime()));
    System.out.println();

    System.out.println("Alternative suffixes: ");
    rdf.setShowZeroDays(false);
    rdf.setDaySuffix(":");
    rdf.setHourSuffix(":");
    rdf.setMinuteSuffix(":");
    rdf.setSecondSuffix("");
    System.out.println(rdf.format(c1.getTime()));
    System.out.println();
}
```

The output from this code is:

```

Default:
11h37m43.123s

Hide milliseconds:
11h37m43s

Show zero day output:
0d11h37m43s

Alternative suffixes:
11:37:43

```

### 41.2.3 Constructors

The following constructors are defined:

- `public RelativeDateFormat(); [1.0.3]`  
Creates a new instance with `baseMillis` set to zero.
- `public RelativeDateFormat(Date time); [1.0.3]`  
Creates a new instance with the reference point (or `baseMillis`) taken from the supplied `Date` instance.
- `public RelativeDateFormat(long baseMillis); [1.0.3]`  
Creates a new instance with the specified reference point (`baseMillis` is specified in milliseconds since 1-Jan-1970, the same encoding used by `java.util.Date`).

### 41.2.4 General Attributes

The `baseMillis` attribute defines the reference point for the elapsed time calculation:

- `public long getBaseMillis(); [1.0.3]`  
Returns the reference point for the elapsed time calculation, expressed in milliseconds since midnight, 1-Jan-1970. This is typically defined in the constructor.
- `public void setBaseMillis(long baseMillis); [1.0.3]`  
Sets the reference point for the elapsed time calculation.

A flag determines whether or not the formatter displays the number of days if the elapsed time is less than one day:

- `public boolean setShowZeroDays(); [1.0.3]`  
Returns `true` if the formatter should display the day field if the elapsed time is less than one day, and `false` otherwise. The default is `false`.
- `public void setShowZeroDays(boolean show); [1.0.3]`  
Sets the flag that determines whether or not the formatter should display the day field if the elapsed time is less than one day.

To control the text displayed following the day count:

- `public String getDaySuffix(); [1.0.3]`  
Returns the string that is appended to the number of days in the formatted output. The default value is "d".
- `public void setDaySuffix(String suffix); [1.0.3]`  
Sets the string that is appended to the number of days in the formatted output. This method throws an `IllegalArgumentException` if `suffix` is `null` (use an empty string for no suffix).

To control the text displayed following the number of hours in the elapsed time:

- `public String getHourSuffix(); [1.0.3]`  
Returns the string that is appended to the number of hours in the formatted output. The default value is "h".
- `public void setHourSuffix(String suffix); [1.0.3]`  
Sets the string that is appended to the number of hours in the formatted output. This method throws an `IllegalArgumentException` if `suffix` is `null` (use an empty string for no suffix).

To control the text displayed following the number of minutes in the elapsed time:

► `public String getMinuteSuffix(); [1.0.3]`

Returns the string that is appended to the number of minutes in the formatted output. The default value is "m".

► `public void setMinuteSuffix(String suffix); [1.0.3]`

Sets the string that is appended to the number of days in the formatted output. This method throws an `IllegalArgumentException` if `suffix` is `null` (use an empty string for no suffix).

To control the text displayed following the number of seconds in the elapsed time:

► `public String getSecondSuffix(); [1.0.3]`

Returns the string that is appended to the number of seconds in the formatted output. The default value is "s".

► `public void setSecondSuffix(String suffix); [1.0.3]`

Sets the string that is appended to the number of seconds in the formatted output. This method throws an `IllegalArgumentException` if `suffix` is `null` (use an empty string for no suffix).

Since the elapsed seconds can be computed with millisecond precision, it is possible to specify a `NumberFormat` instance to control how this value is displayed:

► `public void setSecondFormatter(NumberFormat formatter); [1.0.3]`

Specifies the number formatter used for the seconds field (remember that date/time values can be specified with millisecond precision). If `formatter` is `null`, this method throws an `IllegalArgumentException`.

#### 41.2.5 Formatting and Parsing

The formatting is implemented by the following method:

► `public StringBuffer format(Date date, StringBuffer toAppendTo, FieldPosition fieldPosition); [1.0.3]`

Appends the elapsed time between `baseMillis` and the millisecond represented by `Date` to the specified `StringBuffer`.

Parsing is not supported by this class:

► `public Date parse(String source, ParsePosition pos); [1.0.3]`

This method always returns `null`, because parsing is not supported by this formatter.

#### 41.2.6 Equals, Cloning and Serialization

This class overrides the equals method:

► `public boolean equals(Object obj); [1.0.3]`

Tests this formatter for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### 41.2.7 Notes

Some points to note:

- this class cannot be used for parsing dates;
- this class was first introduced in JFreeChart 1.0.3;
- a demo (`RelativeDateFormatDemo1.java`) is included in the JFreeChart demo collection.

# Chapter 42

## Package: org.jfree.data

### 42.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

A design principle in JFreeChart is that there should be a clear separation between the *data* (as represented by the classes in this package) and its *presentation* (controlled by the plot and renderer classes defined elsewhere). For this reason, you will not find methods or attributes that relate to presentation (for example, series colors or line styles) in the dataset classes.

### 42.2 ComparableObjectItem

#### 42.2.1 Overview

A base class that associates an `Object` with a `Comparable` instance. This base class is designed to be subclassed—the following subclasses are included in JFreeChart:

- `OHLCItem`;
- `XIntervalDataItem`;
- `YIntervalDataItem`;
- `XYIntervalDataItem`;

Typically, you won't use any of these classes directly—they will be created by the corresponding `ComparableObjectSeries` subclass as required.

*This class was first introduced in JFreeChart version 1.0.3.*

#### 42.2.2 Constructor

A single constructor is defined:

► `public ComparableObjectItem(Comparable x, Object y);`

Creates a new instance. If `x` is `null`, this constructor throws an `IllegalArgumentException`. Note that `x` should be both immutable and `Serializable`.

#### 42.2.3 Methods

The following methods are defined:

► `protected Comparable getComparable();`

Returns the `x`-value that was passed to the constructor (never `null`).

```
► protected Object getObject();
Returns the y-value that was passed to the constructor (possibly null).

► protected void setObject(Object y);
Sets the y-value (null is permitted).

► public int compareTo(Object o1);
Compares this instance to an arbitrary object. This default implementation checks to see if o1 is an instance of ComparableObjectItem, and if so returns the comparison of the Comparable for this instance versus the Comparable for o1.
```

#### 42.2.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this instance for equality with an arbitrary object.
```

Instances of this class are `cloneable` and `Serializable` (provided that the x and y values passed to the constructor are `Serializable`).

#### See Also

[ComparableObjectSeries](#).

### 42.3 ComparableObjectSeries

#### 42.3.1 Overview

A series containing zero, one or many items of the form (`Comparable`, `Object`). This is a very general base class that provides a useful base for subclasses to implement special kinds of series and datasets. Known subclasses include:

- [XYIntervalSeries](#).

*This class was first introduced in JFreeChart version 1.0.3.*

#### 42.3.2 Constructor

To create a new series:

```
► public ComparableObjectSeries(Comparable key); [1.0.3]
Equivalent to ComparableObjectSeries(key, true, true)—see the next constructor.

► public ComparableObjectSeries(Comparable key, boolean autoSort,
boolean allowDuplicateXValues); [1.0.3]
Creates a new series. The key is used to uniquely identify the series, and should be both immutable and Serializable. The autoSort flag controls whether or not the items in the series are sorted by ascending order of the Comparable x-value. The allowDuplicateXValues method determines whether two (or more) items in the series can have the same Comparable x-value.
```

#### 42.3.3 Methods

The following methods are defined:

```
► public boolean getAutoSort(); [1.0.3]
Returns true if the items in the series are automatically sorted into ascending order by x-value, and false otherwise.

► public boolean getAllowDuplicateXValues(); [1.0.3]
Returns true if two (or more) items in the series can have the same x-value, and false otherwise.
```

► `public int getItemCount(); [1.0.3]`  
 Returns the number of items in the series.

► `public int getMaximumItemCount(); [1.0.3]`  
 Returns the maximum number of items that can be added to the series.

► `public void setMaximumItemCount(int maximum); [1.0.3]`  
 Sets the maximum number of items that can be added to the series.

► `protected void add(Comparable x, Object y); [1.0.3]`  
 Equivalent to `add(x, y, true)`—see the next method.

► `protected void add(Comparable x, Object y, boolean notify); [1.0.3]`  
 Adds a new (`Comparable`, `Object`) data item to the series and, if `notify` is `true`, sends a `SeriesChangeEvent` to all registered listeners.

► `protected void add(ComparableObjectItem item, boolean notify); [1.0.3]`  
 Adds a new items to the series and, if `notify` is `true`, sends a `SeriesChangeEvent` to all registered listeners.

► `public int indexOf(Comparable x); [1.0.3]`  
 Returns the index of an item in the series that has the specified x-value, or -1.

► `protected void update(Comparable x, Object y); [1.0.3]`  
 Replaces the y-value for the item with the specified x-value.

► `protected void updateByIndex(int index, Object y); [1.0.3]`  
 Replaces the y-value for the item at the specified `index`.

► `protected ComparableObjectItem getDataItem(int index); [1.0.3]`  
 Returns the data item at the specified position in the series.

► `protected void delete(int start, int end); [1.0.3]`  
 Deletes all the items in the series from `start` to `end` inclusive, and sends a `SeriesChangeEvent` to all registered listeners.

► `protected void clear(); [1.0.3]`  
 Clears all items from the series and sends a `SeriesChangeEvent` to all registered listeners.

► `protected ComparableObjectItem remove(int index); [1.0.3]`  
 Removes the item at the specified index, sends a `SeriesChangeEvent` to all registered listeners, and returns a reference to the removed item.

► `public ComparableObjectItem remove(Comparable x); [1.0.3]`  
 Removes the item with the specified x-value, sends a `SeriesChangeEvent` to all registered listeners, and returns a reference to the removed item.

#### 42.3.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`  
 Tests this series for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 42.4 DataUtilities

#### 42.4.1 Overview

This class contains utility methods that relate to general data classes (but not datasets, for which there is the `DatasetUtilities` class).

### 42.4.2 Methods

To create an array of `Number` objects from an array of `double` primitives:

► `public static Number[] createNumberArray(double[] data);`

Returns an array of `Double` objects created from the values in the `data` array (`null` not permitted).

► `public static Number[][] createNumberArray2D(double[][] data);`

Returns an array of arrays of `Double` objects created from the values in the `data` array. Note that this structure may be “jagged” (each array within the structure may have a different length).

To calculate the cumulative percentage values from a collection of data values:

► `public static KeyedValues getCumulativePercentages(KeyedValues data);`

Returns a new collection of data values containing the cumulative percentage values from the specified data.

#### See Also

[DatasetUtilities](#).

## 42.5 DefaultKeyValue

### 42.5.1 Overview

A *(key, value)* data item, where the key is an instance of `Comparable` and the value is an instance of `Number`. For the value, you can use `null` to represent a missing or unknown value. This class provides a default implementation of the `KeyValue` interface.

### 42.5.2 Usage

This class is typically used to represent individual data items in a larger collection, such as [DefaultKeyedValues](#).

### 42.5.3 Constructor

To create a new instance:

► `public DefaultKeyValue(Comparable key, Number value);`

Creates a new data item that associates a value with a key. The key should be an immutable object such as `String`. The value can be any `Number` instance, or `null` to represent a missing or unknown value.

### 42.5.4 Methods

There are methods to access the key and value attributes:

► `public Comparable getKey();`

Returns the key.

► `public Number getValue();`

Returns the value (possibly `null`).

Once a `DefaultKeyValue` instance is created, the key can never be changed, but you can update the value:

► `public synchronized void setValue(Number value);`

Sets the value for this data item.

### 42.5.5 Notes

Some points to note:

- cloning is supported, but no deep cloning is performed because it is assumed that both the key and value are immutable (we know this is true for the value, and assume it to be true for the key).
- this class is serializable provided that the key is serializable.

## 42.6 DefaultKeyedValues

### 42.6.1 Overview

An ordered list of *(key, value)* data items, where the key is an instance of `Comparable`<sup>1</sup> and the value is an instance of `Number`. This class implements the `KeyedValues` interface.

This class is typically used as internal storage for other classes in JFreeChart—for example, the `DefaultPieDataset` class.

### 42.6.2 Constructor

To create a new instance:

```
→ public DefaultKeyedValues();
Creates a new instance that is initially empty.
```

### 42.6.3 Reading Values

To find the number of items in the data structure:

```
→ public int getItemCount();
Returns the number of items in the data structure (possibly 0).
```

To obtain a list of the keys in the data structure:

```
→ public List getKeys();
Returns a new list containing references to the keys in this data structure. The order of the keys in the list is significant, because it is possible to fetch keys and values from this data structure using an integer index (see getKey(int) and getValue(int)). Modifying the content of the returned list has no effect on the DefaultKeyedValues instance.
```

To get the key for an item in the data structure:

```
→ public Comparable getKey(int index);
Returns the key (never null) at the specified index. If index is not in the range 0 to getItemCount() - 1, this method throws an IndexOutOfBoundsException.
```

To get the index for a key in the data structure:

```
→ public int getIndex(Comparable key);
Returns the index of the specified key, or -1 if the key is not found. If key is null, this method throws an IllegalArgumentException.
```

To get the value for an item in the data structure:

```
→ public Number getValue(int index);
Returns the value (possibly null) with the specified index. If index is not in the range 0 to getItemCount() - 1, this method throws an IndexOutOfBoundsException.
→ public Number getValue(Comparable key);
Returns the value (possibly null) associated with the specified key. If there is no item with the specified key, this method throws an UnknownKeyException. If key is null, this method throws an IllegalArgumentException.
```

---

<sup>1</sup>You can use `String` instances as keys, since `String` implements `Comparable`.

#### 42.6.4 Adding and Updating Values

To add or update a value in the container:

- `public void addValue(Comparable key, double value);`  
Equivalent to `addValue(key, new Double(value))`—see below.
- `public void addValue(Comparable key, Number value);`  
Equivalent to `setValue(key, value)`—see below.
- `public void setValue(Comparable key, double value);`  
Equivalent to `setValue(key, new Double(value))`—see below.
- `public void setValue(Comparable key, Number value);`  
Adds a new (key, value) data item to the end of the list or, if there is already an item with the specified key, updates an existing item. If key is null, this method throws an `IllegalArgumentException`. A null value is permitted, and indicates an unknown or missing data value.

In the preceding methods, the key can be any instance of `Comparable`—for most purposes, a `String` is sufficient. It is recommended that you use immutable objects for keys, although this is not enforced.

#### 42.6.5 Removing Values

To remove an item from the list:

- `public void removeValue(int index);`  
Removes the value with the specified index (which should be in the range 0 to `getItemCount() - 1`). If index is outside this range, this method throws an `IndexOutOfBoundsException`.
- `public void removeValue(Comparable key);`  
Removes the value with the specified key (if the key is not recognised, this method throws an `UnknownKeyException`). If key is null, this method throws an `IllegalArgumentException`.

To clear all values from the container:

- `public void clear(); [1.0.2]`  
Clears any values from the container, making it empty.

#### 42.6.6 Sorting Items

A couple of utility methods are provided to sort the items in the list by key or by value:

- `public void sortByKeys(SortOrder order);`  
Sorts the items in the list, by key, into the specified order (`SortOrder.ASCENDING` or `SortOrder.DESCENDING`). If order is null, this method throws an `IllegalArgumentException`.
- `public void sortByValues(SortOrder order);`  
Sorts the items in the list, by value, into the specified order (`SortOrder.ASCENDING` or `SortOrder.DESCENDING`). If order is null, this method throws an `IllegalArgumentException`.

#### 42.6.7 Equals, Cloning and Serialization

This class overrides the `equals()` and `hashCode()` methods:

- `public boolean equals(Object obj);`  
Tests this instance for equality with an arbitrary (possibly null) object. Returns true if and only if:
  - obj is an instance of `KeyedValues`;
  - obj contains the same items in the same order as this list.
- `public int hashCode();`  
Returns a hash code for this instance.

To make a clone:

- `public Object clone() throws CloneNotSupportedException;`  
Returns an independent copy of this instance. A `CloneNotSupportedException` may be thrown if you use keys that are not `Cloneable`.

Instances of this class are serializable.

### 42.6.8 Notes

Some points to note:

- if you are relying on this data structure being `Cloneable` and `Serializable`, you should ensure that the keys for the data items you add to the structure support cloning and serialization;
- this class provides a default implementation of the `KeyedValues` interface;
- this class is used in the implementation of the `DefaultPieDataset` class.

## 42.7 DefaultKeyedValues2D

### 42.7.1 Overview

A storage structure for a table of values that are associated with (non-`null`) keys. This class provides a default implementation of the `KeyedValues2D` interface. For general JFreeChart work, you probably won't need to use this class directly.

### 42.7.2 Constructors

This class defines two constructors:

- ➔ `public DefaultKeyedValues2D();`  
Equivalent to `DefaultKeyedValues2D(false)`—see the next constructor.
- ➔ `public DefaultKeyedValues2D(boolean sortRowKeys);`  
Creates a new data structure, initially empty. The `sortRowKeys` controls whether or not the rows in the table are reordered by ascending order of row keys.

### 42.7.3 Data Access By Index

To access the data values from the table by index, the following method can be used:

- ➔ `public Number getValue(int row, int column);`  
Returns the value (possibly `null`) at a given cell in the table. If `row` is not in the range 0 to `getRowCount() - 1` this method throws an `IndexOutOfBoundsException`. A similar check is performed for the `column` argument.

The row and column counts can be found with the following methods:

- ➔ `public int getRowCount();`  
Returns the number of rows in the table.
- ➔ `public int getColumnCount();`  
Returns the number of columns in the table.

### 42.7.4 Data Access By Keys

Sometimes it is more convenient to access the data values using a pair of keys, one to identify the row and the other to identify the column:

- ➔ `public Number getValue(Comparable rowKey, Comparable columnKey);`  
Returns the value for the specified cell in the table. If there is no cell with the specified keys, this method throws an `UnknownKeyException`. If `rowKey` or `columnKey` is `null`, this method throws an `IllegalArgumentException`.

To find information about the row keys, and to convert them to and from indices:

- ➔ `public List getRowKeys();`  
Returns the list of row keys, in unmodifiable form.

► `public Comparable getRowKey(int row);`  
 Returns the key for the specified row—the key is never `null`. If `row` is not in the range 0 to `getRowCount() - 1`, this method throws an `IndexOutOfBoundsException`.

► `public int getRowIndex(Comparable key);`  
 Returns the index of the row with the specified key, or `-1` if no row uses the specified key. If `key` is `null`, this method throws an `IllegalArgumentException`.

Similarly, the following methods are provided for the column keys:

► `public List getColumnKeys();`  
 Returns the list of column keys, in unmodifiable form.

► `public Comparable getColumnKey(int column);`  
 Returns the key for the specified column—the key is never `null`. If `column` is not in the range 0 to `getColumnCount() - 1`, this method throws an `IndexOutOfBoundsException`.

► `public int getColumnIndex(Comparable key);`  
 Returns the index of the row with the specified key, or `-1` if no row uses the specified key. If `key` is `null`, this method throws an `IllegalArgumentException`.

#### 42.7.5 Adding and Removing Data

To add a value to the table, or update an existing value:

► `public void addValue(Number value, Comparable rowKey, Comparable columnKey);`  
 Equivalent to `setValue(value, rowKey, columnKey)`—see the next method.

► `public void setValue(Number value, Comparable rowKey, Comparable columnKey);`  
 Adds a value to the table, or overwrites an existing value. If `rowKey` or `columnKey` is `null`, this method throws an `IllegalArgumentException`.

To remove an item from the table:

► `public void removeValue(Comparable rowKey, Comparable columnKey);`  
 Removes an item from the table (if that results in either the row or column holding only `null` values, that row or column is removed). If `rowKey` or `columnKey` is `null`, this method throws an `IllegalArgumentException`.

To remove a row from the dataset:

► `public void removeRow(int row);`  
 Removes the specified row from the dataset. If `row` is not in the range 0 to `getRowCount() - 1`, this method throws an `IndexOutOfBoundsException`.

► `public void removeRow(Comparable rowKey);`  
 Removes the specified row from the dataset. If the table does not contain a row with the specified key, this method throws an `UnknownKeyException`. If `rowKey` is `null`, this method throws an `IllegalArgumentException`.

To remove a column from the dataset:

► `public void removeColumn(int column);`  
 Removes the specified column from the dataset. If `column` is not in the range 0 to `getColumnCount() - 1`, this method throws an `IndexOutOfBoundsException`.

► `public void removeColumn(Comparable columnKey);`  
 Removes the specified column from the dataset. If the table does not contain a column with the specified key, this method throws an `UnknownKeyException`. If `columnKey` is `null`, this method throws an `IllegalArgumentException`.

To clear the table completely:

► `public void clear();`  
 Clears the table of all data values.

### 42.7.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);  
Tests this data structure for equality with an arbitrary object.
```

Instances of this class are `Cloneable` and `Serializable`.

### 42.7.7 Notes

The `DefaultCategoryDataset` class uses an instance of this class to store its data.

## 42.8 DomainInfo

### 42.8.1 Overview

An interface that provides information about the bounds for a dataset's domain (x-values). A dataset should implement this interface if it can provide this information in an efficient way—otherwise, methods in the `DatasetUtilities` class will iterate over all values in the dataset to determine the bounds.

### 42.8.2 Methods

To get the minimum value in the dataset's domain:

```
► public double getDomainLowerBound(boolean includeInterval);  
Returns the lower bound in the dataset's domain (x-values).
```

To get the maximum value in the dataset's domain:

```
► public double getDomainUpperBound(boolean includeInterval);  
Returns the upper bound in the dataset's domain (x-values).
```

To get the range of values in the dataset's domain:

```
► public Range getDomainBounds(boolean includeInterval);  
Returns the bounds of the dataset's domain (x-values).
```

For all of the above methods, the `includeInterval` argument is intended for “extended” datasets that define domain values as intervals (for example, instances of `IntervalXYDataset`). For these datasets, the caller may be interested in the bounds with or without including the interval. Regular datasets can ignore this argument.

### 42.8.3 Notes

It is not mandatory for a dataset to implement this interface.

#### See Also

[RangeInfo](#), [DatasetUtilities](#).

## 42.9 DomainOrder

### 42.9.1 Overview

An enumeration of the order of the domain values in a dataset—see table 42.1 for a list of the defined values.

This enumeration is used by the `getDomainOrder()` method in the `XYDataset` interface.

ID:	Description:
DomainOrder.ASCENDING	Ascending order.
DomainOrder.DESCENDING	Descending order.
DomainOrder.NONE	No order.

Table 42.1: Constants defined by DomainOrder

## 42.10 KeyedObject

### 42.10.1 Overview

A simple class that associates an arbitrary object with a key value.

### 42.10.2 Constructor

To create a new instance:

► public KeyedObject(Comparable key, Object object);  
Creates a new (key, value) pair.

### 42.10.3 Methods

To access the key:

► public Comparable getKey();  
Returns the key, as defined via the constructor.

To access the object:

► public Object getObject();  
Returns the object (possibly null).  
► public void setObject(Object object);  
Sets the object (null is permitted).

### 42.10.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);  
Tests this instance for equality with an arbitrary object.

Instances of this class are cloneable:

► public Object clone() throws CloneNotSupportedException;  
Returns a clone of this instance. The key is assumed to be immutable, so the clone references the same key as this instance. The object is deep-cloned if it implemented the `PublicCloneable` interface.

Instances of this class are serializable, provided that both the key and its associated object are serializable.

#### See Also

[KeyedObjects](#).

## 42.11 KeyedObjects

### 42.11.1 Overview

An ordered list of (`Comparable`, `Object`) data items. This class is used by the [KeyedObjects2D](#) class.

### 42.11.2 Constructor

To create a new instance:

► `public KeyedObjects();`  
Creates a new collection, initially empty.

### 42.11.3 General Methods

To find the number of items in the list:

► `public int getItemCount();`  
Returns the number of (*key, value*) items in the collection.

To get the key for the item at a specified index in the list:

► `public Comparable getKey(int index);`  
Returns the key at the given position in the list. If *index* is not in the range 0 to `getItemCount()` - 1, this method throws an `IndexOutOfBoundsException`.

To get the index for the item in the list with the specified key:

► `public int getIndex(Comparable key);`  
Returns the position in the list for the given *key*, or -1 if the specified key is not present. If *key* is `null`, this method throws an `IllegalArgumentException`.

To get a list containing the keys for this list:

► `public List getKeys();`  
Returns a list containing all the keys from this list.

To get the object at the specified position in the list:

► `public Object getObject(int item);`  
Returns the object (possibly `null`) at the given position in the list. If *item* is not in the range 0 to `getItemCount()` - 1, this method throws an `IndexOutOfBoundsException`.

To get the object associated with a key:

► `public Object getObject(Comparable key);`  
Returns the object associated with the specified *key*. If the specified key is not found, this method throws an `UnknownKeyException`.

### 42.11.4 Adding and Removing

To add an item to the (end of the) list:

► `public void addObject(Comparable key, Object object);`  
Equivalent to `setObject(key, object)`—see the next method.  
► `public void setObject(Comparable key, Object object);`  
Adds a key and its associated object to the list (the *object* may be `null`). If the list already contains an item with the specified key, the object for that key is updated. It is highly recommended that the key should be an immutable object instance. If *key* is `null`, this method throws an `IllegalArgumentException`.

To insert an item at a specific position within the list:

► `public void insertValue(int position, Comparable key, Object value); [1.0.7]`  
Inserts an item at the specified position in the list.

To remove an item:

► `public void removeValue(int index);`  
Removes the item at the specified position in the list.  
► `public void removeValue(Comparable key);`  
Removes the item with the specified key. If the specified key is not found, this method throws an `UnknownKeyException`.

To clear all items from the list:

► `public void clear(); [1.0.7]`  
Clears all items from the list.

#### 42.11.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object o);
Tests this list for equality with an arbitrary object.

► public Object clone() throws CloneNotSupportedException;
Returns a clone of the list.
```

#### See Also

[KeyedObject](#), [KeyedObjects2D](#).

### 42.12 KeyedObjects2D

#### 42.12.1 Overview

A tabular data structure that stores arbitrary objects that are accessible via row and column keys. This class is used internally by the following classes:

- [DefaultBoxAndWhiskerCategoryDataset](#);
- [DefaultStatisticalCategoryDataset](#).

#### 42.12.2 Constructor

This class defines only the default constructor:

```
► public KeyedObjects2D();
Creates a new (empty) table of objects.
```

#### 42.12.3 Data Access By Index

To access the data values from the table by index, the following method can be used:

```
► public Object getObject(int row, int column);
Returns the object (possibly null) at a given cell in the table. If row is not in the range 0 to getRowCount() - 1, this method throws an IndexOutOfBoundsException. Likewise for the column argument.
```

The row and column counts can be found with the following methods:

```
► public int getRowCount();
Returns the number of rows in the table.

► public int getColumnCount();
Returns the number of columns in the table.
```

#### 42.12.4 Data Access By Keys

To access a data item by key values:

```
► public Object getObject(Comparable rowKey, Comparable columnKey);
Returns the object (possibly null) at a cell in the table. If rowKey or columnKey is null, this method throws an IllegalArgumentException. If rowKey or columnKey is not recognised, this method throws an UnknownKeyException.
```

The row keys:

```
► public List getRowKeys();
Returns a list of the row keys for this table.
```

► public Comparable getRowKey(int row);  
 Returns the key for the specified row in the table.

► public int getRowIndex(Comparable key);  
 Returns the row index for the specified key.

The column keys:

► public List getColumnKeys();  
 Returns a list of the column keys for this table.

► public Comparable getColumnKey(int column);  
 Returns the key for the specified column in the table.

► public int getColumnIndex(Comparable key);  
 Returns the column index for the specified key.

#### 42.12.5 Adding and Removing Items

To add/remove items from the table:

► public void addObject(Object object, Comparable rowKey, Comparable columnKey);  
 Equivalent to `setObject(object, rowKey, columnKey)`—see the next method.

► public void setObject(Object object, Comparable rowKey, Comparable columnKey);  
 Adds the specified object to the table with the give row and column keys. If `rowKey` or `columnKey` is `null`, this method throws an `IllegalArgumentException`.

► public void removeObject(Comparable rowKey, Comparable columnKey);  
 Removes the object at the specified cell in the table.

To remove an entire row or column from the table:

► public void removeRow(int rowIndex);  
 Removes an entire row from the table.

► public void removeRow(Comparable rowKey);  
 Removes an entire row from the table. If `rowKey` is `null`, this method throws an `IllegalArgumentException`. If `rowKey` is not recognised, this method throws an `UnknownKeyException`.

► public void removeColumn(int columnIndex);  
 Removes an entire column from the table.

► public void removeColumn(Comparable columnKey);  
 Removes an entire column from the table. If `columnKey` is `null`, this method throws an `IllegalArgumentException`. If `columnKey` is not recognised, this method throws an `UnknownKeyException`.

To clear all the data from this structure:

► public void clear(); [1.0.7]  
 Clears all data.

#### 42.12.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);  
 Tests this table for equality with an arbitrary object.

► public Object clone() throws CloneNotSupportedException;  
 Returns a clone of this table.

#### See Also

[KeyedObject](#), [KeyedObjects](#).

## 42.13 KeyedValue

### 42.13.1 Overview

A *keyed value* is a value (`Number`) that is associated with a key (`Comparable`).

### 42.13.2 Methods

This interface extends the `Value` interface.

To access the key associated with the value:

```
→ public Comparable getKey();
    Returns the key associated with the value.
```

### 42.13.3 Notes

The `DefaultKeyedValue` class provides one implementation of this interface.

## 42.14 KeyedValueComparator

### 42.14.1 Overview

This class is used to compare two `KeyedValue` objects, either by key or by value.

### 42.14.2 Constructor

To create a new instance:

```
→ public KeyedValueComparator(KeyedValueComparatorType type, SortOrder order);
    Creates a new comparator. The type specifies whether to sort by keys or by values, and the
    order specifies ascending or descending order.
```

### 42.14.3 Methods

```
→ public KeyedValueComparatorType getType();
    Returns the type of comparator (by key or by value).

→ public SortOrder getOrder();
    Returns the sort order (SortOrder.ASCENDING or SortOrder.DESCENDING).

→ public int compare(Object o1, Object o2);
    Compares two objects, which are assumed to be instances of KeyedValue, and returns -1, 0 or 1
    to indicate the relative ordering of the two objects.
```

### 42.14.4 Notes

This comparator is used by the `sortByKeys()` and `sortByValues()` methods in the `DefaultKeyedValues` class.

## 42.15 KeyedValueComparatorType

### 42.15.1 Overview

Used to represent the two comparison types:

- `BY_KEY`—sorts items by key;
- `BY_VALUE`—sorts items by value.

These constants are used by the `KeyedValueComparator` class.

## 42.16 KeyedValues

### 42.16.1 Overview

A collection of *(key, value)* data items, where the key is an instance of `Comparable` and the value is an instance of `Number`. This interface extends the `Values` interface. The `DefaultKeyedValues` class provides a default implementation.

### 42.16.2 Methods

To access the key associated with a value:

```
➔ public Comparable getKey(int index);
```

Returns the key (never `null`) at the specified index. This method should throw an `IndexOutOfBoundsException` if `index` is not in the range 0 to `getItemCount()` - 1.

To convert a key into an item index:

```
➔ public int getIndex(Comparable key);
```

Returns the item index for the specified key, or -1 if no such key is found. This method should throw an `IllegalArgumentException` if `key` is `null`.

To get a list of all keys in the collection:

```
➔ public List getKeys();
```

Returns an unmodifiable list of the keys in the collection.

To get the value associated with a key:

```
➔ public Number getValue(Comparable key);
```

Returns the value associated with a key.

### 42.16.3 Notes

Some points to note:

- the *(key, value)* pairs in the collection have a specific order, since each key is associated with a zero-based index;
- the `DefaultKeyedValues` class provides one implementation of this interface.

## 42.17 KeyedValues2D

### 42.17.1 Overview

A table of values that can be accessed using a *row key* and a *column key*. This interface extends the `Values2D` interface.

### 42.17.2 Methods

To get the key for a row:

```
➔ public Comparable getRowKey(int row);
```

Returns the key associated with a row.

To convert a row key into an index:

```
➔ public int getRowIndex(Comparable key);
```

Returns the row index for the given key.

To get a list of the row keys:

```
➔ public List getRowKeys();
```

Returns a list of the row keys.

To get the key for a column:

```
► public Comparable getColumnKey(int column);
```

Returns the key associated with a column.

To convert a column key into an index:

```
► public int getColumnIndex(Comparable key);
```

Returns the column index for a given key.

To return a list of column keys:

```
► public List getColumnKeys();
```

Returns a list of the column keys.

To get the value associated with a pair of keys:

```
► public Number getValue(Comparable rowKey, Comparable columnKey);
```

Returns the value associated with the keys (possibly null).

### 42.17.3 Notes

The `DefaultKeyedValues2D` class provides one implementation of this interface.

## 42.18 KeyToGroupMap

### 42.18.1 Overview

A utility class that provides a mapping between a set of keys (instances of `Comparable`) and a set of groups (also instances of `Comparable`). A default group is always specified, and any key that is not explicitly mapped to a group is assumed to be mapped to the default group.

This class is `Serializable` and implements the `Cloneable` and `PublicCloneable` interfaces.

### 42.18.2 Constructors

To create a new map:

```
► public KeyToGroupMap(Comparable defaultGroup);
```

Creates a map with the specified default group (`null` not permitted). Apart from the default group, the new map is empty. You can add groups and mappings using the methods documented below.

There is also a default constructor:

```
► public KeyToGroupMap();
```

Creates a map with a default group named “Default Group”.

### 42.18.3 Methods

To find the group that a key is mapped to:

```
► public Comparable getGroup(Comparable key);
```

Returns the group that a key is mapped to. This method never returns `null`—if the `key` has not been explicitly mapped, the default group is returned.

To map a key to a group:

```
► public void mapKeyToGroup(Comparable key, Comparable group);
```

Adds a mapping between the specified `key` and `group` (`null` is not permitted for the key, `null` for the group clears any existing mapping for the specified `key`). If the `key` is already mapped to a group, the mapping is changed. If the group is not defined within the map, it is added automatically.

To find out how many groups are represented within the map:

► `public int getGroupCount();`

Returns the number of groups in the map (this is always at least 1, since there is always a default group).

To obtain a list of the groups in the map:

► `public List getGroups();`

Returns a list of the groups in the map. This list always contains at least one group (the default group). The list itself is independent of the map, so you can alter it without affecting the state of the map. The default group will always appear first in the list, the remaining groups are in the order that they were originally added to the map.

All groups in the map are assigned a unique index (the index of the default group is always 0). To get the index for a group:

► `public int getGroupIndex(Comparable group);`

Returns the group index (which corresponds to the position within the list returned by the `getGroups()` method).

#### 42.18.4 Notes

Some points to note:

- an instance of this class is used by the [GroupedStackedBarRenderer](#) class.

### 42.19 Range

#### 42.19.1 Overview

A class that represents a range of values by recording the lower and upper bounds of the range. This can be used, for example, to specify the bounds for an axis on a chart.

#### 42.19.2 Constructor

To create a new instance:

► `public Range(double lower, double upper);`

Creates a new instance with the specified bounds. Note that `lower` must be less than or equals to `upper`. Once created, an instance is immutable—you cannot change the bounds on that instance.

#### 42.19.3 Methods

This class provides methods to access the bounds, but not to change them. To get the lower bound, upper bound, or central value for the range:

► `public double getLowerBound();`

Returns the lower bound for the range.

► `public double getUpperBound();`

Returns the upper bound for the range.

► `public double getCentralValue();`

Returns the central value for the range.

#### 42.19.4 Other Methods

To test whether or not a value falls within the range:

```
► public boolean contains(double value);  
Returns true if lowerbound <= value <= upperbound, and false otherwise.
```

To test whether this range intersects with another range:

```
► public boolean intersects(double b0, double b1);  
Returns true if this range intersects with the specified range, and false otherwise.  
  
► public boolean intersects(Range range); [1.0.9]  
Returns true if this range intersects the specified range, and false otherwise.
```

To “force” a value to fit within a range:

```
► public double constrain(double value);  
Returns the value within the range that is closest to value. This will either be value or one of  
the range bounds.
```

#### 42.19.5 Combining, Shifting and Expanding Ranges

To combine two ranges:

```
► public static Range combine(Range range1, Range range2);  
Returns a new range which encompasses both of the specified ranges.
```

To create a new range that is based on an existing range but expanded by a certain percentage:

```
► public static Range expand(Range range, double lowerMargin, double upperMargin);  
Creates and returns a new range that is an expanded version of the supplied range. The  
specified margins (percentages of the range length) are added to the existing range boundaries  
to create the new range.
```

To shift a range:

```
► public static Range shift(Range base, double delta);  
Creates a new range by adding delta to the lower and upper bounds of this range.  
  
► public static Range shift(Range base, double delta, boolean allowZeroCrossing);  
Creates a new range by adding delta to the lower and upper bounds of this range. The  
allowZeroCrossing argument controls whether or not the bounds are allowed to cross zero. For  
example, you might have a positive range that you want to shift downwards, but without  
allowing the bounds to become negative.  
  
► public static Range scale(Range base, double factor); [1.0.9]  
Scales the range by multiplying the range bounds by the specified factor. If factor is less than  
or equal to zero, this method throws an IllegalArgumentException.
```

#### 42.19.6 Equals and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);  
Returns true if obj:  


- is not null;
- is an instance of Range;
- has upper and lower bounds that are the same as those of this range.

```

Otherwise returns `false`.

This class is `Serializable` but not `Cloneable` (not required since instances are immutable).

### 42.19.7 Notes

Some points to note:

- the `DateRange` class extends this class to support a date range.

## 42.20 RangeInfo

### 42.20.1 Overview

An interface that provides information about the bounds for a dataset's range (y-values). A dataset should implement this interface if it can provide this information in an efficient way—otherwise, methods in the `DatasetUtilities` class will iterate over all values in the dataset to determine the bounds.

### 42.20.2 Methods

To get the minimum value in the dataset's range:

```
➔ public double getRangeLowerBound(boolean includeInterval);
```

Returns the lower bound for the dataset's range (in other words, the smallest y-value in the dataset). If the dataset contains no values, this method should return `Double.NaN`.

To get the maximum value in the dataset's range:

```
➔ public double getRangeUpperBound(boolean includeInterval);
```

Returns the upper bound for the dataset's range (in other words, the largest y-value in the dataset). If the dataset contains no values, this method should return `Double.NaN`.

To get the range of values in the dataset's range:

```
➔ public Range getRangeBounds(boolean includeInterval);
```

Returns the bounds for the dataset's range. If the dataset contains no values, this method should return `null`.

For all of the above methods, the `includeInterval` argument is intended for “extended” datasets that define range values as intervals (for example, instances of `IntervalXYDataset`). For these datasets, the caller may be interested in the bounds with or without including the interval. Regular datasets can ignore this argument.

### 42.20.3 Notes

It is not mandatory for a dataset to implement this interface.

#### See Also

[DomainInfo](#).

## 42.21 RangeType

### 42.21.1 Overview

This class provides an enumeration of range types for a `NumberAxis` class. The available types are:

- `RangeType.FULL` (the default for a `NumberAxis`);
- `RangeType.NEGATIVE`;
- `RangeType.POSITIVE`;

This is used to restrict the range of values displayed on a `NumberAxis` to positive values only, or negative values only.

## 42.22 UnknownKeyException

### 42.22.1 Overview

An exception that indicates that a key that has been used to access a data value is not recognised. For example, methods in the following classes can throw this exception:

- `DefaultPieDataset`;
- `DefaultCategoryDataset`;
- `DefaultKeyedValues`;
- `DefaultKeyedValues2D`;

## 42.23 Value

### 42.23.1 Overview

An interface for accessing a single value (`Number` object). By way of an example, the `ValueDataset` interface extends this interface, and is used by the `ThermometerPlot` class.

### 42.23.2 Methods

The interface defines a single method for accessing the value:

```
➔ public Number getValue();
Returns the value (possibly null).
```

### 42.23.3 Notes

Some notes:

- the `KeyedValue` interface extends this interface.
- the `DefaultKeyedValue` class provides one implementation of this interface.

## 42.24 Values

### 42.24.1 Overview

An interface for accessing an ordered list of values.

### 42.24.2 Methods

To get the number of items in the collection:

```
➔ public int getItemCount();
Returns the number of items in the ordered list of values. This may be zero.
```

To get a value from the list:

```
➔ public Number getValue(int index);
Returns the value with the specified index (possibly null). Classes that implement this method
should throw an IndexOutOfBoundsException if index is not in the range 0 to getItemCount() -
1.
```

### 42.24.3 Notes

Some notes:

- the `KeyedValues` interface extends this interface.
- the `DefaultKeyedValues` class provides one implementation of this interface.

## 42.25 Values2D

### 42.25.1 Overview

An interface for accessing a table of values by row and column index.

### 42.25.2 Methods

To get the number of rows in the table:

► `public int getRowCount();`  
Returns the row count.

To get the number of columns in the table:

► `public int getColumnCount();`  
Returns the column count.

To get a value from one cell in the table:

► `public Number getValue(int row, int column);`  
Returns a value (possibly `null`) from a cell in the table.

### 42.25.3 Notes

Some points to note:

- the `KeyedValues2D` interface extends this interface.
- the `DefaultKeyedValues2D` class provides one implementation of this interface.

# Chapter 43

## Package: org.jfree.data.category

### 43.1 Introduction

This package contains interfaces and classes for the datasets used (primarily) by JFreeChart's [CategoryPlot](#) class.

### 43.2 CategoryDataset

#### 43.2.1 Overview

A *category dataset* is a table of values that can be accessed using row and column keys. This type of dataset is most commonly used to create bar charts.

This interface extends the [KeyedValues2D](#) and [Dataset](#) interfaces.

#### 43.2.2 Methods

This interface adds no additional methods to those defined in the [KeyedValues2D](#) and [Dataset](#) interfaces.

#### 43.2.3 Notes

Some points to note:

- this interface provides the methods required for *reading* the dataset, not for updating it. Classes that implement this interface may be “read-only”, or they may provide “write” access.
- the [DefaultCategoryDataset](#) class provides a useful implementation of this interface.
- the [CategoryToPieDataset](#) class converts one row or column of the dataset into a [PieDataset](#).
- you can read a [CategoryDataset](#) from a file (in a prespecified XML format) using the [DatasetReader](#) class.

#### See Also

[CategoryPlot](#).

### 43.3 CategoryToPieDataset

#### 43.3.1 Overview

A utility class that presents one row or column of data from a [CategoryDataset](#) via the [PieDataset](#) interface.

### 43.3.2 Constructor

To create a new instance:

```
→ public CategoryToPieDataset(CategoryDataset source, TableOrder extract, int index);
```

Creates a new pie dataset based on the `source`. The `extract` argument specifies whether the dataset uses a row or column from the source dataset (use `TableOrder.BY_ROW` or `TableOrder.BY_COLUMN`), and the `index` controls which row or column is selected.

### 43.3.3 Methods

The following methods provide details of the configuration of this dataset (as passed to the constructor):

```
→ public CategoryDataset getUnderlyingDataset(); [1.0.2]
```

Returns a reference to the underlying dataset, which may be `null`.

```
→ public TableOrder getExtractType(); [1.0.2]
```

Returns `TableOrder.BY_COLUMN` or `TableOrder.BY_ROW` to indicate how data is extracted from the underlying dataset (the actual row or column used is determined by the value returned by `getExtractIndex()`).

```
→ public int getExtractIndex(); [1.0.2]
```

Returns the row or column index from which to extract data from the underlying dataset.

The following methods are required to implement the `PieDataset` interface:

```
→ public int getItemCount();
```

Returns the number of items in the dataset (which may be zero).

```
→ public Number getValue(int index);
```

Returns the value at the specified index (the value may be `null`). This method throws an `IndexOutOfBoundsException` if `index` is not in the range from 0 to `getItemCount() - 1`.

```
→ public Comparable getKey(int index);
```

Returns the key with the specified index. This method throws an `IndexOutOfBoundsException` if `index` is not in the range from 0 to `getItemCount() - 1`.

```
→ public int getIndex(Comparable key);
```

Returns the index of the specified key, or -1 if there is no such key. If `key` is `null`, this method throws an `IllegalArgumentException`.

```
→ public List getKeys();
```

Returns an unmodifiable list of keys for the dataset. The list may be empty, but is never `null`.

```
→ public Number getValue(Comparable key);
```

Returns the value associated with the specified key, or `null` if the key is not recognised.<sup>1</sup>

```
→ public void datasetChanged (DatasetChangeEvent event);
```

This method receives events from the underlying dataset and responds by firing a new event with this dataset as the source. You shouldn't need to call this method directly.

### 43.3.4 Equals, Cloning and Serialization

This class overrides the `equals` method. Like other datasets, equality is determined only by the data values (and keys) and not other characteristics of the dataset implementation:

```
→ public boolean equals(Object obj);
```

Tests this dataset for equality with an arbitrary object. Returns `true` if and only if:

- `obj` is an instance of `PieDataset`;
- `obj` has the same keys and values (in the same order) as this dataset.

Instances of this class are `Cloneable` and `Serializable`.

---

<sup>1</sup>Note that `null` is a possible value for valid keys also—in that case, you can call the `getIndex(Comparable)` method to determine whether or not the key exists for the dataset.

### 43.3.5 Notes

This class registers itself with the underlying `CategoryDataset` to receive change events. Whenever the underlying dataset is changed, a new `DatasetChangeEvent` is triggered and sent to all registered listeners.

## 43.4 DefaultCategoryDataset

### 43.4.1 Overview

A dataset that implements the `CategoryDataset` interface, forming a two-dimensional table of values, with keys for the row and column headings. The column headings are the “categories” for the dataset, while the row headings are the series keys. This dataset is commonly used to create bar charts, but can also be used for other chart types.

### 43.4.2 Constructors

To create a new dataset:

```
➔ public DefaultCategoryDataset();
Creates a new dataset that is initially empty.
```

### 43.4.3 Simple Data Access

This dataset is a two-dimensional table, with each cell in the table containing a `Number` value (or `null`). To find out how many rows and columns the dataset/table has:

```
➔ public int getRowCount();
Returns the number of rows in the dataset (possibly zero).

➔ public int getColumnCount();
Returns the number of columns in the dataset (possibly zero).
```

To retrieve a value from the dataset:

```
➔ public Number getValue(int row, int column);
Returns the value from a cell in the dataset—the value may be null. If row is not in the range 0 to getRowCount() - 1, this method throws an IndexOutOfBoundsException (likewise for the column argument).
```

### 43.4.4 Row And Column Keys

Each row and column in the dataset is associated with a key, which is an instance of `Comparable` (typically a `String`). A pair of keys (one row key and one column key) uniquely identifies a cell in the table. In charts, when a label is required for a row (series) or column (category), the `toString()` method for the appropriate key is used.

To fetch a value from the dataset, given the row and column key:

```
➔ public Number getValue(Comparable rowKey, Comparable columnKey);
Returns the value from a cell in the dataset—this may be null. If either of the specified keys is not defined for this dataset, the method throws an UnknownKeyException. If rowKey or columnKey is null, this method throws an IllegalArgumentException.
```

The following methods are used to manage the row keys:

```
➔ public List getRowKeys();
Returns an ordered and unmodifiable list of the row keys for this dataset.

➔ public Comparable getRowKey(int row);
Returns the key for the specified row. If row is not in the range 0 to getRowCount() - 1, this method throws an IndexOutOfBoundsException.
```

► `public int getRowIndex(Comparable key);`

Returns the row index for the specified key, or -1 if there is no row with the specified key. If `key` is null, this method throws an `IllegalArgumentException`.

Similar methods are provided to manage the column keys:

► `public List getColumnKeys();`

Returns an ordered and unmodifiable list of the column keys for this dataset.

► `public Comparable getColumnKey(int column);`

Returns the key for the specified column. If `column` is not in the range 0 to `getColumnCount() - 1`, this method throws an `IndexOutOfBoundsException`.

► `public int getColumnIndex(Comparable key);`

Returns the column index for the specified key, or -1 if there is no column with the specified key. If `key` is null, this method throws an `IllegalArgumentException`.

#### 43.4.5 Adding and Updating Data

To add a value to the dataset:

► `public addValue(Number value, Comparable rowKey, Comparable columnKey)`

Adds a value to the dataset then sends a `DatasetChangeEvent` to all registered listeners. The value can be null (to indicate missing data). If there is already a value for the given keys, it is overwritten. If `rowKey` or `columnKey` is null, this method throws an `IllegalArgumentException`.

► `public void addValue(double value, Comparable rowKey, Comparable columnKey);`

Equivalent to `addValue(new Double(value), rowKey, columnKey)`—see the previous method.

► `public void setValue(Number value, Comparable rowKey, Comparable columnKey);`

Equivalent to `addValue(Number, Comparable, Comparable)`—see above.

► `public void setValue(double value, Comparable rowKey, Comparable columnKey);`

Equivalent to `addValue(double, Comparable, Comparable)`—see above.

The following method increments an existing value by the specified amount:

► `public void incrementValue(double value, Comparable rowKey, Comparable columnKey);`

Adds `value` to an existing item in the dataset. If the existing value is null, it is treated as though it were 0.0. If `rowKey` or `columnKey` is not defined for this dataset, this method throws an `UnknownKeyException`.

#### 43.4.6 Removing Data

To remove a value from the dataset:

► `public void removeValue(Comparable rowKey, Comparable columnKey);`

Removes the specified value from the dataset and sends a `DatasetChangeEvent` to all registered listeners. If `rowKey` or `columnKey` is null, this method throws an `IllegalArgumentException`.

To remove an entire row from the dataset:

► `public void removeRow(int rowIndex);`

Removes the specified row from the dataset and sends a `DatasetChangeEvent` to all registered listeners.

► `public void removeRow(Comparable rowKey);`

As above.

To remove an entire column from the dataset:

► `public void removeColumn(int columnIndex);`

Removes the specified column from the dataset and sends a `DatasetChangeEvent` to all registered listeners.

► `public void removeColumn(Comparable columnKey);`

As above.

To remove all data from the dataset:

► `public void clear();`

Clears all data from the dataset and sends a `DatasetChangeEvent` to all registered listeners.

### 43.4.7 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
    Tests this dataset for equality with an arbitrary object.
```

Instances of this class are `Cloneable`<sup>2</sup> and `Serializable`.

### 43.4.8 Notes

Some points to note:

- the `DatasetUtilities` class has static methods for creating instances of this class using array data;
- internally, this class uses an instance of `DefaultKeyedValues2D` to store its data;
- the `DatasetReader` class provides a facility for reading a dataset from an XML file.

#### See Also

[CategoryDataset](#), [CategoryPlot](#).

## 43.5 DefaultIntervalCategoryDataset

### 43.5.1 Overview

A default implementation of the `IntervalCategoryDataset` interface. You can use this dataset with a `CategoryPlot` and an `IntervalBarRenderer`.

### 43.5.2 Constructors

The following constructors are defined:

```
► public DefaultIntervalCategoryDataset(double[][] starts, double[][] ends);
    Creates a new dataset with the supplied start and end values.

► public DefaultIntervalCategoryDataset(Number[][] starts, Number[][] ends);
    Creates a new dataset with the supplied start and end values.

► public DefaultIntervalCategoryDataset(String[] seriesNames, Number[][] starts,
    Number[][] ends);
    Creates a new dataset with the specified series names, start values and end values. The category
    names are automatically generated.

► public DefaultIntervalCategoryDataset(Comparable[] seriesKeys, Comparable[] categoryKeys,
    Number[][] starts, Number[][] ends);
    Creates a new dataset with the specified series names, category names, start values and end
    values.
```

In all of the constructors, the data values are indexed by series then by category—for example, to create a dataset with two series and three categories:

```
double[] starts_S1 = new double[] {0.1, 0.2, 0.3};
double[] starts_S2 = new double[] {0.3, 0.4, 0.5};
double[][] starts = new double[][] {starts_S1, starts_S2};

double[] ends_S1 = new double[] {0.5, 0.6, 0.7};
double[] ends_S2 = new double[] {0.7, 0.8, 0.9};
double[][] ends = new double[][] {ends_S1, ends_S2};

DefaultIntervalCategoryDataset d = new DefaultIntervalCategoryDataset(starts, ends);
```

---

<sup>2</sup>Note that in versions prior to 1.0.5, cloning doesn't work correctly.

### 43.5.3 Series and Categories

To find the number of series:

► public int getSeriesCount();  
Equivalent to `getRowCount()`—see the next method.

► public int getRowCount();  
Returns the number of rows (series) in the dataset.

To convert between series keys and series indices:

► public Comparable getSeriesKey(int series);  
Equivalent to `getRowKey(series)`—see the next method.

► public Comparable getRowKey(int row);  
Returns the key for the specified series. If `row` is not in the range 0 to `getRowCount()-1`, this method throws an `IllegalArgumentException`.

► public int getSeriesIndex(Comparable seriesKey);  
Equivalent to `getRowIndex(seriesKey)`—see the next method.

► public int getRowIndex(Comparable rowKey);  
Returns the index of the specified `rowKey`, or -1 if the key does not belong to the dataset. If `rowKey` is null, this method throws an `IllegalArgumentException`.

To get an ordered list of the series keys for the dataset:

► public List getRowKeys();  
Returns an ordered (and unmodifiable) list of the row/series keys for the dataset.

To modify all the series keys:

► public void setSeriesKeys(Comparable[] seriesKeys);  
Replaces the existing series keys with the supplied keys. If `seriesKeys` is null, or `seriesKeys.length` does not match the number of series in the dataset, this method throws an `IllegalArgumentException`.

To find the number of categories (or columns) in the dataset, you can use any of the following methods:

► public int getCategoryCount();  
Returns the number of categories in the dataset. Every series has the same categories.

► public int getColumnCount();  
Returns the number of columns in the dataset.

To get the key for a category:

► public Comparable getColumnKey(int column);  
Returns the key for the specified column (category).

► public int getColumnIndex(Comparable columnKey);  
Returns the index of the specified `columnKey`, or -1 if the key does not belong to the dataset. If `columnKey` is null, this method returns an `IllegalArgumentException`.

To get a list of category keys:

► public List getColumnKeys();  
Returns an ordered and unmodifiable list of the column keys.

To modify all the category keys:

► public void setCategoryKeys(Comparable[] categoryKeys);  
Replaces the existing category keys with the supplied keys. If `categoryKeys` is null, or `categoryKeys.length` does not match the number of categories in the dataset, this method throws an `IllegalArgumentException`.

#### 43.5.4 Accessing Data Values

The following methods provide access to the data values:

- `public Number getStartValue(Comparable series, Comparable category);`  
Returns the start value for the specified series and category.
- `public Number getStartValue(int series, int category);`  
Returns the start value for the specified series and category.
- `public Number getEndValue(Comparable series, Comparable category);`  
Returns the end value for the specified series and category.
- `public Number getEndValue(int series, int category);`  
Returns the end value for the specified series and category.

In order to support the [CategoryDataset](#) interface, the following methods are implemented:

- `public Number getValue(Comparable series, Comparable category);`  
Returns the value for the specified series and category—in this case, the end value is returned.
- `public Number getValue(int series, int category);`  
Returns the value for the specified series and category—in this case, the end value is returned.

#### 43.5.5 Modifying Data Values

To modify existing values in the dataset:

- `public void setStartValue(int series, Comparable category, Number value);`  
Sets the start value for an existing data item and sends a [DatasetChangeEvent](#) to all registered listeners. This method doesn't allow the addition of new items.
- `public void setEndValue(int series, Comparable category, Number value);`  
Sets the end value for an existing data item and sends a [DatasetChangeEvent](#) to all registered listeners. This method doesn't allow the addition of new items.

#### 43.5.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this dataset for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### 43.5.7 Notes

Some points to note:

- the `equals()` and `clone()` implementations were broken prior to version 1.0.5;
- a demo (`IntervalBarChartDemo1.java`) is included in the JFreeChart demo collection.

#### See Also

[IntervalCategoryDataset](#).

## 43.6 IntervalCategoryDataset

### 43.6.1 Overview

An extension of the [CategoryDataset](#) interface that adds methods for returning a *start value* and an *end value* for each item in the dataset.

Like a [CategoryDataset](#), this dataset is conceptually a table of data items where the “categories” represent columns and the “series” represent rows. The cells within the table contain three items: the start value, the end value and the value (the final item may be the same as one of the previous values or it may be different).

The [DefaultIntervalCategoryDataset](#) class provides an implementation of this interface.

### 43.6.2 Interface

In addition to the methods inherited from [CategoryDataset](#), this interface defines several methods for accessing the starting and ending x-values for each data item.

To get the start value for a data item:

```
→ public Number getStartValue(int series, int category);  
    Returns the start value for the specified data item.  
  
→ public Number getStartValue(Comparable series, Comparable category);  
    Returns the start value for the specified data item
```

To get the end value for a data item:

```
→ public Number getEndValue(int series, int category);  
    Returns the end value for the specified data item.  
  
→ public Number getEndValue(Comparable series, Comparable category);  
    Returns the end value for the specified data item.
```

Note that all of the above methods can return `null` to represent a missing or unknown value.

### 43.6.3 Notes

Some points to note:

- the [IntervalBarRenderer](#) class expects to receive data from a dataset that implements this interface;
- the [DefaultIntervalCategoryDataset](#) class provides one implementation of this interface;

#### See Also

[CategoryDataset](#).

# Chapter 44

## Package: org.jfree.data.contour

### 44.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart's [ContourPlot](#) class.  
*All this code is deprecated as of version 1.0.4.*

### 44.2 ContourDataset

#### 44.2.1 Overview

The dataset used by the [ContourPlot](#) class. *This interface is deprecated as of version 1.0.4.*

#### 44.2.2 Methods

This interface defines the following methods in addition to those inherited from the [XYZDataset](#) interface:

- `public double getMinZValue();`  
Returns the minimum z-value.
- `public double getMaxZValue();`  
Returns the maximum z-value.
- `public Number[] getXValues();`  
Returns an array containing all the x-values.
- `public Number[] getYValues();`  
Returns an array containing all the y-values.
- `public Number[] getZValues();`  
Returns an array containing all the z-values.
- `public int[] indexX();`  
Returns the index values.
- `public int[] getXIndices();`  
Returns an int array contain the index into the x values.
- `public Range getZValueRange(Range x, Range y);`  
Returns the maximum z-value for the specified visible region of the plot.
- `public boolean isDateAxis(int axisNumber);`  
Returns true if the values for the specified axis are dates (where `axisNumber` is defined as 0-x, 1-y, and 2-z).

#### See Also

[DefaultContourDataset](#).

## 44.3 DefaultContourDataset

### 44.3.1 Overview

A default implementation of the [ContourDataset](#) interface. *This class is deprecated as of version 1.0.4.*

#### See Also

[ContourPlot](#)

## 44.4 NonGridContourDataset

### 44.4.1 Overview

A dataset for use with the [ContourPlot](#) class. *This class is deprecated as of version 1.0.4.*

# Chapter 45

## Package: org.jfree.data.function

### 45.1 Introduction

This package contains a simple function representation and some classes to represent common function types. JFreeChart cannot plot functions directly, but you can use the `sampleFunction2D()` method in the `DatasetUtilities` class to create a dataset containing sample values from any `Function2D`.

### 45.2 Function2D

#### 45.2.1 Overview

A simple interface for a 2D function. Implementations of this interface include:

- `LineFunction2D`;
- `NormalDistributionFunction2D`;
- `PowerFunction2D`.

It is a simple matter to implement your own functions.

#### 45.2.2 Methods

The interface defines a single method for obtaining the value of the function for a given input:

```
► public double getValue(double x);  
    Returns the value of the function for a given input.
```

#### 45.2.3 Notes

The `DatasetUtilities` class provides a method for creating an `XYDataset` by sampling the values of a function.

#### See Also

[LineFunction2D](#), [PowerFunction2D](#).

### 45.3 LineFunction2D

#### 45.3.1 Overview

A simple function of the form  $y = a + bx$ .

### 45.3.2 Constructor

To construct a new line function:

```
➔ public LineFunction2D(double a, double b);
Creates a new line function with the given coefficients.
```

### 45.3.3 Methods

```
➔ public double getValue(double x);
Returns the value of the function for a given input.
```

### 45.3.4 Notes

Some points to note:

- this class implements the [Function2D](#) interface.
- the [RegressionDemo1](#) application provides an example of this class being used.

#### See Also

[PowerFunction2D](#).

## 45.4 NormalDistributionFunction2D

### 45.4.1 Overview

A function that returns values for a normal distribution—see figure 45.1.

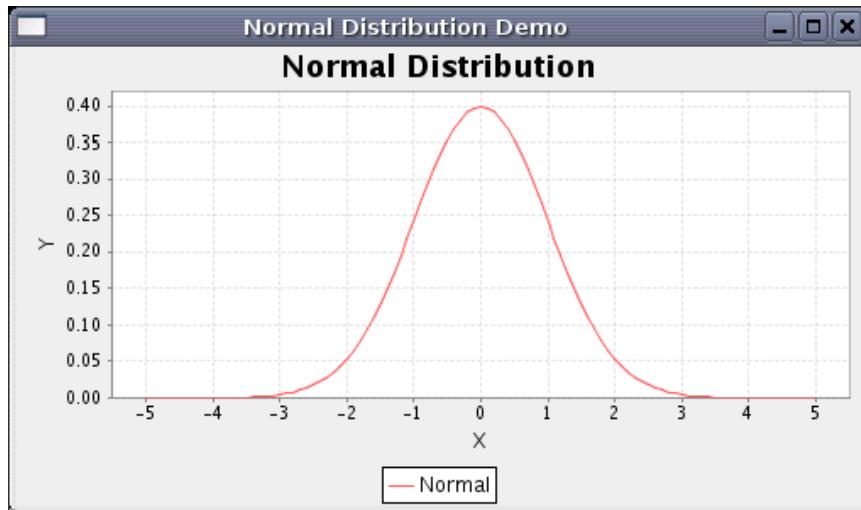


Figure 45.1: A chart using samples from a `NormalDistributionFunction2D`

A demo (`NormalDistributionDemo.java`) is included in the JFreeChart demo collection.

### 45.4.2 Constructor

To create a new instance:

```
➔ public NormalDistributionFunction2D(double mean, double std);
Creates a new normal distribution function with the given mean and standard deviation.
```

### 45.4.3 Methods

To get the mean and standard deviation:

► public double getMean();

Returns the mean value for the function (this is set in the constructor and cannot be modified).

► public double getStandardDeviation();

Returns the standard deviation value for the function (this is set in the constructor and cannot be modified).

To get the function value for a given  $x$  value:

► public double getValue(double x);

Returns the value of the normal distribution function for the given  $x$ .

## 45.5 PowerFunction2D

### 45.5.1 Overview

A function of the form  $y = ax^b$ .

### 45.5.2 Constructor

To construct a new power function:

► public PowerFunction2D(double a, double b);

Creates a new power function with the given coefficients.

### 45.5.3 Methods

► public double getValue(double x);

Returns the value of the function for a given input.

### 45.5.4 Notes

Some points to note:

- this class implements the [Function2D](#) interface.

- the [RegressionDemo1](#) application provides an example of this class being used.

### See Also

[LineFunction2D](#).

# Chapter 46

## Package: org.jfree.data.gantt

### 46.1 Introduction

This package contains classes used to represent the dataset for a simple Gantt chart.

### 46.2 GanttCategoryDataset

#### 46.2.1 Overview

An extension of the [IntervalCategoryDataset](#) interface that is intended for creating simple Gantt charts.

#### 46.2.2 Methods

This interface adds a range of methods in addition to those it inherits from the [IntervalCategoryDataset](#) interface. These are aimed at supporting subtasks within tasks, and providing information about the “percentage complete” for individual tasks.

To get the number of subtasks for a given task:

- ➔ `public int getSubIntervalCount(int row, int column);`  
Returns the number of subtasks defined for the specified item (possibly 0).
- ➔ `public int getSubIntervalCount(Comparable rowKey, Comparable columnKey);`  
Returns the number of subtasks defined for the specified item (possibly 0).

To get the start value (time in milliseconds) for a specific subtask:

- ➔ `public Number getStartValue(int row, int column, int subinterval);`  
Returns the start value for a subtask.
- ➔ `public Number getStartValue(Comparable rowKey, Comparable columnKey, int subinterval);`  
Returns the start value for a subtask.

To get the end value (time in milliseconds) for a specific subtask:

- ➔ `public Number getEndValue(int row, int column, int subinterval);`  
Returns the end value for a subtask.
- ➔ `public Number getEndValue(Comparable rowKey, Comparable columnKey, int subinterval);`  
Returns the end value for a subtask.

To get the percentage complete for a given task:

- ➔ `public Number getPercentComplete(int row, int column);`  
Returns the percentage complete for the specified task. This method can return `null` if the value is unknown.

► public Number getPercentComplete(Comparable rowKey, Comparable columnKey);  
 Returns the percentage complete for the specified task. This method can return `null` if the value is unknown.

To get the percentage complete for a subtask:

► public Number getPercentComplete(int row, int column, int subinterval);  
 Returns the percentage complete for the specified subtask. This method can return `null` if the value is unknown.

► public Number getPercentComplete(Comparable rowKey, Comparable columnKey, int subinterval);  
 Returns the percentage complete for the specified subtask. This method can return `null` if the value is unknown.

### 46.2.3 Notes

Some points to note:

- the `GanttRenderer` class expects to find a dataset of this type;
- this interface is implemented by the `TaskSeriesCollection` class;
- demo applications (`GanttDemo1-3.java`) are included in the JFreeChart demo distribution.

## 46.3 Task

### 46.3.1 Overview

A class that represents a *task*, consisting of:

- a task description;
- a duration (estimated or actual);
- a list of sub-tasks;

In JFreeChart, tasks are used in the construction of *Gantt charts*. One or more related tasks can be added to a `TaskSeries`. In turn, one or more `TaskSeries` can be added to a `TaskSeriesCollection`.

### 46.3.2 Constructors

To create a new task:

► public Task(String description, `TimePeriod` duration);  
 Creates a new task with the specified (estimated) duration.

► public Task(String description, Date start, Date end);  
 Creates a new task with the specified start and end dates.

### 46.3.3 Methods

To access the task description:

► public String getDescription();  
 Returns the task description (never `null`).

► public void setDescription(String description);  
 Sets the task description (`null` not permitted).

To access the task duration (actual or expected):

► public `TimePeriod` getDuration();  
 Returns the task duration (possibly `null`).

► public void setDuration(TimePeriod duration);  
 Sets the task duration (`null` permitted).

To access the “percentage complete” for the task:

► public Double getPercentComplete();  
 Returns the percentage complete (possibly `null`).

► public void setPercentComplete(Double percent);  
 Sets the percentage complete for the task (`null` permitted). The value should be between 0.0 and 1.0. For example, 0.75 is seventy-five percent.

► public void setPercentComplete(double percent);  
 Sets the percentage complete for the task.

#### 46.3.4 Subtasks

A task can define a number of subtasks. To add a subtask:

► public void addSubtask(Task subtask);  
 Adds a subtask (`null` not permitted).

To remove a subtask:

► public void removeSubtask(Task subtask);  
 Removes a subtask.

To find out how many subtasks are defined (if any):

► public int getSubtaskCount();  
 Returns the subtask count.

To access a particular subtask:

► public Task getSubtask(int index);  
 Returns a subtask from the list.

#### 46.3.5 Notes

Some points to note:

- this class is `Cloneable` and `Serializable`;
- tasks can be added to a [TaskSeries](#).

### 46.4 TaskSeries

#### 46.4.1 Overview

A *task series* is a collection of related tasks. You can add one or more `TaskSeries` objects to a [TaskSeriesCollection](#) to create a dataset that can be used to produce *Gantt charts*.

#### 46.4.2 Constructor

To create a new task series:

► public TaskSeries(String name);  
 Creates a new series with the specified name (`null` not permitted). The series is initially empty (contains no tasks).

### 46.4.3 Methods

To add and remove tasks:

- `public void add(Task task);`  
Adds a task to the series and sends a `SeriesChangeEvent` to all registered listeners.
- `public void remove(Task task);`  
Removes a task from the series and sends a `SeriesChangeEvent` to all registered listeners.
- `public void removeAll();`  
Removes all tasks from the series and sends a `SeriesChangeEvent` to all registered listeners.

To find the number of tasks in the series:

- `public int getItemCount();`  
Returns the number of items (tasks) in the series.

To access a particular task:

- `public Task get(int index);`  
Returns a task from the series.

You can obtain a list of the tasks in a series:

- `public List getTasks();`  
Returns an unmodifiable list of the tasks in a series.

### 46.4.4 Notes

Some points to note:

- the `TaskSeriesCollection` class is used to create collections of one or more task series.

## 46.5 TaskSeriesCollection

### 46.5.1 Overview

A *task series collection* contains one or more `TaskSeries` objects, and provides access to the task information via the `GanttCategoryDataset` interface. You can use this class as the dataset for a *Gantt chart*.

### 46.5.2 Constructor

To create a new collection:

- `public TaskSeriesCollection();`  
Creates a new collection, initially empty.

### 46.5.3 Adding and Removing Series

To add a new series:

- `public void add(TaskSeries series);`  
Adds a series to the collection (null not permitted) and sends a `DatasetChangeEvent` to all registered listeners.

To remove a series:

- `public void remove(TaskSeries series);`  
Removes a series from the collection and sends a `DatasetChangeEvent` to all registered listeners.
- `public void remove(int series);`  
Removes a series from the collection and sends a `DatasetChangeEvent` to all registered listeners.

To remove all series from the collection:

```
➔ public void removeAll();
```

Removes all the series from the collection.

To access a series in the collection:

```
➔ public TaskSeries getSeries(Comparable key);
```

Returns the series with the specified key, or `null` if there is no such series. This method first appeared in version 1.0.1.

```
➔ public TaskSeries getSeries(int series);
```

Returns the series with the specified index. This method first appeared in version 1.0.1.

#### 46.5.4 Retrieving Values

To support the use of this class as a dataset, the following methods are used to retrieve values:

```
➔ public Number getValue(Comparable rowKey, Comparable columnKey);
```

Returns the value for the given row (series) and column (task description).

```
➔ public Number getValue(int row, int column);
```

Returns the value for the given row (series) and column (task).

```
➔ public Number getStartValue(Comparable rowKey, Comparable columnKey);
```

Returns the start value for the given row (series) and column (task).

```
➔ public Number getStartValue (int row, int column);
```

Returns the start value for the given row (series) and column (task).

```
➔ public Number getEndValue (Comparable rowKey, Comparable columnKey);
```

Returns the end value for the given row (series) and column (task).

```
➔ public Number getEndValue(int row, int column);
```

Returns the end value for the given row (series) and column (task).

To get the percentage complete:

```
➔ public Number getPercentComplete(int row, int column);
```

Returns the percentage complete for the given row (series) and column (task).

```
➔ public Number getPercentComplete(Comparable rowKey, Comparable columnKey);
```

Returns the percentage complete for the given row (series) and column (task).

#### 46.5.5 Sub-Intervals

To find the number of sub-intervals for a task within a series:

```
➔ public int getSubIntervalCount(int row, int column);
```

Returns the number of sub-intervals (if any) for a task within a series.

```
➔ public int getSubIntervalCount(Comparable rowKey, Comparable columnKey);
```

Returns the number of sub-intervals (if any) for a task within a series.

```
➔ public Number getStartValue(int row, int column, int subinterval);
```

Returns the start value for a particular sub-interval within a task.

```
➔ public Number getStartValue(Comparable rowKey, Comparable columnKey, int subinterval);
```

Returns the start value for a particular sub-interval within a task.

```
➔ public Number getEndValue(int row, int column, int subinterval);
```

Returns the end value for a particular sub-interval within a task.

```
➔ public Number getEndValue(Comparable rowKey, Comparable columnKey, int subinterval);
```

Returns the end value for a particular sub-interval within a task.

To get the percentage complete for a sub-interval:

```
➔ public Number getPercentComplete(int row, int column, int subinterval);
```

Returns the percentage complete for a sub-interval.

```
➔ public Number getPercentComplete(Comparable rowKey, Comparable columnKey, int subinterval);
```

Returns the percentage complete for a sub-interval.

#### 46.5.6 Methods

To get the name of a series in the collection:

► `public String getSeriesName(int series);`  
Returns the name of a series in the collection.

To get the number of series in the collection:

► `public int getSeriesCount();`  
Returns the number of series in the collection.

► `public int getRowCount();`  
Returns the number of series in the collection.

► `public List getRowKeys();`  
Returns a list of the row keys (each series name is used as a row key).

► `public int getColumnCount();`  
The number of “columns” in the collection. This is equal to the number of unique keys (task descriptions) in all the task series in the collection.

► `public List getColumnKeys();`  
Returns a list of the column keys (an aggregation of all the task descriptions in all the series within the collection).

► `public Comparable getColumnKey(int index);`  
Returns the column key that corresponds to the given index.

► `public int getColumnIndex(Comparable columnKey);`  
Returns the index that corresponds to the given column key.

► `public int getRowIndex(Comparable rowKey);`  
Returns the index that corresponds to the given row key.

► `public Comparable getRowKey(int index);`  
Returns the row key that corresponds to the given index.

# Chapter 47

## Package: org.jfree.data.general

### 47.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

### 47.2 AbstractDataset

#### 47.2.1 Overview

A useful base class for implementing the `Dataset` interface (or extensions). This class provides a default implementation of the *change listener* mechanism, which allows the dataset to send a `DatasetChangeEvent` to registered listeners every time the dataset is updated.

#### 47.2.2 Constructors

The default constructor:

```
► protected AbstractDataset();  
Allocates storage for the registered change listeners.
```

#### 47.2.3 Dataset Groups

Datasets can be allocated to a group, but in the current version of JFreeChart the group is not used. Still, the methods remain:

```
► public DatasetGroup getGroup();  
Returns the group that the dataset belongs to (never null).  
  
► public void setGroup(DatasetGroup group);  
Sets the group for the dataset (null not permitted).
```

#### 47.2.4 Change Listeners

To register a change listener:

```
► public void addChangeListener(DatasetChangeListener listener);  
Registers a change listener with the dataset. The listener will be notified whenever the dataset  
changes, via a call to the datasetChanged() method.
```

To deregister a change listener:

```
► public void removeChangeListener(DatasetChangeListener listener);  
Deregisters a change listener. The listener will be no longer be notified whenever the dataset  
changes.
```

### 47.2.5 Other Methods

The following utility method can be used to send a change event to all registered listeners:

```
► protected void fireDatasetChanged();
Sends a DatasetChangeEvent to all registered listeners.
```

### 47.2.6 Notes

Some points to note:

- in most cases, JFreeChart will automatically register listeners for you, and update charts whenever the data changes.
- you can implement a dataset without subclassing `AbstractDataset`. This class is provided simply for convenience to save you having to implement your own change listener mechanism.
- if you write your own class that extends `AbstractDataset`, you need to remember to call `fireDatasetChanged()` whenever the data in your class is modified.

#### See Also

[Dataset](#), [DatasetChangeListener](#), [AbstractSeriesDataset](#).

## 47.3 AbstractSeriesDataset

### 47.3.1 Overview

A useful base class for implementing the `SeriesDataset` interface (or extensions). This class extends `AbstractDataset`.

### 47.3.2 Constructors

This class is never instantiated directly, so the constructor is protected:

```
► protected AbstractSeriesDataset();
Simply calls the constructor of the superclass.
```

### 47.3.3 Methods

Two abstract methods are declared:

```
► public abstract int getSeriesCount();
Returns the number of series in the dataset—to be implemented by subclasses.

► public abstract Comparable getSeriesKey(int series);
Returns the key for a series in the dataset. When a series label is required for display in a chart (typically in the chart's legend) the toString() method is called on the series key. If series is not in the range 0 to getSeriesCount() - 1, implementing methods should throw an IndexOutOfBoundsException (preferred) or an IllegalArgumentException (historical).
```

To get the index of a series from its key:

```
► public int indexOf(Comparable seriesKey);
Returns the index of the series with the specified key, or -1 if there is no such series in the dataset. If seriesKey is null, this method returns -1 (a series cannot have a null key).
```

This method receives series change notifications:

```
► public void seriesChanged(SeriesChangeEvent event);
The default behaviour provided by this method is to raise a DatasetChangeEvent every time this method is called.
```

#### 47.3.4 Notes

This class is provided simply for convenience, you are not required to use it when developing your own dataset classes. [AbstractXYDataset](#) is a subclass.

##### See Also

[Dataset](#), [AbstractXYDataset](#).

### 47.4 CombinationDataset

#### 47.4.1 Overview

An interface that defines the methods that should be implemented by a *combination dataset*.

#### 47.4.2 Notes

This interface is implemented by the [CombinedDataset](#) class.

### 47.5 CombinedDataset

#### 47.5.1 Overview

A dataset that can combine other datasets.

#### 47.5.2 Notes

The combined charts feature, originally developed by Bill Kelemen, has been restructured so that it is no longer necessary to use this class. However, you can still use this class if you need to construct a dataset that is the union of existing datasets.

##### See Also

[CombinationDataset](#).

### 47.6 Dataset

#### 47.6.1 Overview

The base interface for datasets. Not useful in its own right, this interface is further extended by [PieDataset](#), [CategoryDataset](#) and [SeriesDataset](#).

#### 47.6.2 Methods

This base interface defines two methods for registering change listeners:

```
► public void addChangeListener(DatasetChangeListener listener);  
Registers a change listener with the dataset. The listener will be notified whenever the dataset  
changes.  
  
► public void removeChangeListener(DatasetChangeListener listener);  
Deregisters a change listener.
```

#### 47.6.3 Notes

This interface is not intended to be used directly, you should use an extension of this interface such as [PieDataset](#), [CategoryDataset](#) or [XYDataset](#).

## 47.7 DatasetChangeEvent

### 47.7.1 Overview

An event that is used to provide information about changes to datasets. In general, any change to a dataset will trigger a `DatasetChangeEvent`, which then allows listeners to react to that change. By default, when a dataset is added to a plot, the plot registers itself as a listener with the dataset, and will receive notification whenever the dataset is changed.

### 47.7.2 Constructors

The standard constructor:

```
➔ public DatasetChangeEvent(Object source, Dataset dataset);  
Creates a new event. Usually the source is the dataset, but this is not guaranteed.
```

### 47.7.3 Methods

To get a reference to the `Dataset` that generated the event:

```
➔ public Dataset getDataset();  
Returns the dataset which generated the event.
```

### 47.7.4 Notes

The current implementation simply indicates that some change has been made to the dataset. In the future, this class may carry more information about the change.

#### See Also

[DatasetChangeListener](#).

## 47.8 DatasetChangeListener

### 47.8.1 Overview

An interface through which dataset change event notifications are posted. If a class needs to receive notification of changes to a dataset, then it should implement this interface and register itself with the dataset.

### 47.8.2 Methods

The interface defines a single method:

```
➔ public void datasetChanged(DatasetChangeEvent event);  
Receives notification of a change to a dataset.
```

### 47.8.3 Notes

The `Plot` class implements this interface in order to receive notification of changes to its dataset(s).

#### See Also

[DatasetChangeEvent](#).

## 47.9 DatasetGroup

### 47.9.1 Overview

A *dataset group* provides a mechanism for grouping related datasets. At present, this is not used.

### 47.9.2 Constructor

This class has a single constructor:

```
↳ public DatasetGroup();
Creates a new group.
```

### 47.9.3 Methods

The only method in this class creates a clone of the group:

```
↳ public Object clone() throws CloneNotSupportedException;
Returns a clone of the group.
```

### 47.9.4 Notes

As mentioned in the overview, this class currently serves no real purpose.

## 47.10 DatasetUtilities

### 47.10.1 Overview

A collection of utility methods for working with datasets. Additional methods are provided by the [DataUtilities](#) class.

### 47.10.2 Creating Datasets

In general, you should create and populate datasets by using the dataset class directly (that is, create a new instance and use its methods to populate it with data). However, for some special situations, utility methods have been written to create and populate datasets in specialised ways. These methods are documented here.

#### PieDatasets

A [PieDataset](#) is equivalent to a [CategoryDataset](#) that has only one row or only one column. Some methods are available to make it easy to create a new [PieDataset](#) from one row or column of a [CategoryDataset](#):

```
↳ public static PieDataset createPieDatasetForRow(CategoryDataset dataset,
Comparable rowKey);
Returns a pie dataset created from the values in the specified row of the given dataset.

↳ public static PieDataset createPieDatasetForRow(CategoryDataset dataset,
int row);
Returns a pie dataset created from the values in the specified row of the given dataset.

↳ public static PieDataset createPieDatasetForColumn(CategoryDataset dataset,
Comparable columnKey);
Returns a pie dataset created from the values in the specified column of the given dataset.

↳ public static PieDataset createPieDatasetForColumn(CategoryDataset dataset, int column);
Returns a pie dataset created from the values in the specified column of the given dataset.
```

#### CategoryDatasets

Many developers have requested the ability to create charts from data stored in arrays. To make this easier, the following methods will create a [CategoryDataset](#) from array-based data:

```
➔ public static CategoryDataset createCategoryDataset(String rowKeyPrefix, String columnKeyPrefix,
double[][] data);
```

Creates a category dataset by copying the values in the `data` array. Row and column keys are auto-generated using the supplied prefixes, by appending 1, 2, 3, etc. If `data` is a “jagged” array, the resulting dataset will contain `null` values for some items.

```
➔ public static CategoryDataset createCategoryDataset(String rowKeyPrefix, String columnKeyPrefix,
Number[][] data);
```

As for the preceding method, except that `data` is an array of `Number` objects.

```
➔ public static CategoryDataset createCategoryDataset(Comparable[] rowKeys, Comparable[] columnKeys,
double[][] data);
```

As for the preceding methods, except that row and column keys are explicitly provided rather than auto-generated.

```
➔ public static CategoryDataset createCategoryDataset(Comparable rowKey, KeyedValues rowData);
```

Creates a new dataset containing a single row of data.

## XYDatasets

To create an `XYDataset` by sampling values from a `Function2D`:

```
➔ public static XYDataset sampleFunction2D(Function2D f,
double start, double end, int samples, Comparable seriesName);
```

Creates a new `XYDataset` by sampling values in a specified range for the `Function2D`. If `f` or `seriesName` is `null`, this method throws an `IllegalArgumentException`. The returned dataset is currently an instance of `XYSeriesCollection` containing a single series.

For an example, see `Function2DDemo1.java` in the JFreeChart demo collection.

### 47.10.3 PieDataset Methods

To determine if a `PieDataset` has any data for display:

```
➔ public static boolean isEmptyOrNull(PieDataset dataset);
```

Returns `true` if the dataset is empty or `null`, and `false` otherwise. Empty in this context means the dataset contains no *positive* values.

To calculate the total of the values in a `PieDataset`:

```
➔ public static double calculatePieDatasetTotal(PieDataset dataset);
```

Returns the total of all the *positive* values in the dataset (negative and `null` values are ignored).

To reduce the number of items in a `PieDataset` by consolidating some of the smaller value items:

```
➔ public static PieDataset createConsolidatedPieDataset(PieDataset source, Comparable key,
double minimumPercent);
```

Creates a new pie dataset, based on `source`, by consolidating all the low value items (that is, those that represent less than `minimumPercent` of the total) into a single item with the specified `key`. Note that the consolidation only happens if there are at least 2 low value items to aggregate.

```
➔ public static PieDataset createConsolidatedPieDataset(PieDataset source, Comparable key,
double minimumPercent, int minItems);
```

Creates a new pie dataset, based on `source`, by consolidating all the low value items (that is, those that represent less than `minimumPercent` of the total) into a single item with the specified `key`. Note that the consolidation only happens if there are at least `minItems` low value items to aggregate.

#### 47.10.4 CategoryDataset Bounds

A [CategoryDataset](#) has numerical range values, and this class contains methods for determining the upper and lower bounds for these values. To get the minimum range value in a dataset:

```
➔ public static Number findMinimumRangeValue(CategoryDataset dataset);
```

Returns the minimum range value for the dataset. If the dataset implements the [RangeInfo](#) interface, then this will be used to obtain the minimum range value. Otherwise, this method iterates through all of the data.

To get the maximum range value in a dataset:

```
➔ public static Number findMaximumRangeValue(CategoryDataset dataset);
```

Returns the maximum range value for the dataset. If the dataset implements the [RangeInfo](#) interface, then this will be used to obtain the maximum range value. Otherwise, this method iterates through all of the data.

```
➔ public static Range findRangeBounds(CategoryDataset dataset);
```

Returns the bounds of the range (or Y-) values in the dataset.

```
➔ public static Range findRangeBounds(CategoryDataset dataset, boolean includeInterval);
```

Returns the bounds of the range (or Y-) values in the dataset. If `dataset` is an instance of [IntervalCategoryDataset](#), then the `includeInterval` flag determines whether or not the y-interval is taken into account for the bounds.

```
➔ public static Range iterateCategoryRangeBounds(CategoryDataset dataset, boolean includeInterval);
```

As for the preceding method, but calculated by iteration.

In some cases, the data from a [CategoryDataset](#) is presented in a “stacked” format (for example, in a stacked bar chart). In these cases, it is necessary to calculate the minimum and maximum of the category *totals* (positive and negative values totalled separately). To get the minimum “stacked” range value in a [CategoryDataset](#):

```
➔ public static Number findMinimumStackedRangeValue(CategoryDataset dataset);
```

Returns the minimum stacked range value in a dataset.

To get the maximum “stacked” range value in a [CategoryDataset](#):

```
➔ public static Number findMaximumStackedRangeValue(CategoryDataset dataset);
```

Returns the maximum stacked range value in a dataset.

```
➔ public static Range findStackedRangeBounds(CategoryDataset dataset);
```

Returns the bounds for the stacked range values.

```
➔ public static Range findStackedRangeBounds(CategoryDataset dataset, KeyToGroupMap map);
```

Returns the bounds for the stacked range values, taking into account the grouping specified by `map`.

```
➔ public static Range findCumulativeRangeBounds(CategoryDataset dataset);
```

Returns the cumulative bounds for the specified dataset, or `null` if `dataset` contains only `null` values. If `dataset` is `null`, this method throws an [IllegalArgumentException](#). This method is currently used by the [WaterfallBarRenderer](#) class.

#### 47.10.5 XYDataset Bounds

To get the minimum domain value in a dataset:

```
➔ public static Number findMinimumDomainValue(XYDataset dataset);
```

Returns the minimum domain value for the dataset. If the dataset implements the [DomainInfo](#) interface, then this will be used to obtain the minimum domain value. Otherwise, this method iterates through all of the data.

To get the maximum domain value in a dataset:

```
➔ public static Number findMaximumDomainValue(XYDataset dataset);
```

Returns the maximum domain value for the dataset. If the dataset implements the [DomainInfo](#) interface, then this will be used to obtain the maximum domain value. Otherwise, this method iterates through all of the data.

```

→ public static Number findMinimumRangeValue(XYDataset dataset);
Returns the minimum range value for the dataset.

→ public static Number findMaximumRangeValue(XYDataset dataset);
Returns the maximum range value for the dataset.

→ public static Range findDomainBounds(XYDataset dataset);
Returns the bounds for the domain (or X-) values in the dataset.

→ public static Range findDomainBounds(XYDataset dataset, boolean includeInterval);
Returns the bounds for the domain (or X-) values in the dataset. The includeInterval flag determines whether or not the x-interval is taken into account when determining the bounds (note that an x-interval is only defined by datasets that implement the extended interface IntervalXYDataset).

→ public static Range iterateDomainBounds(XYDataset dataset);
Returns the bounds for the domain (or X-) values in the dataset, determined by iterating over all the values in the dataset.

→ public static Range iterateDomainBounds(XYDataset dataset, boolean includeInterval);
Returns the bounds for the domain (or X-) values in the dataset, determined by iterating over all the values in the dataset. The includeInterval flag determines whether or not the x-interval is taken into account when determining the bounds (note that an x-interval is only defined by datasets that implement the extended interface IntervalXYDataset).

→ public static Range findRangeBounds(XYDataset dataset);
Returns the bounds of the range (Y-) values in the dataset.

→ public static Range findRangeBounds(XYDataset dataset, boolean includeInterval);
Returns the bounds of the range (Y-) values in the dataset.

→ public static Range iterateXYRangeBounds(XYDataset dataset);
Finds the bounds of the range (Y-) values in the dataset, by iterating through the entire dataset. It is usually better to call findRangeBounds() since it will check if the range can be calculated more efficiently via the RangeInfo interface—if not, it calls this method anyway.

→ public static Range findStackedRangeBounds(TableXYDataset dataset);
Returns the bounds of the stacked range values in the dataset, assuming a base value (for stacking) of 0.0.

→ public static Range findStackedRangeBounds(TableXYDataset dataset, double base);
Returns the bounds of the stacked range values in the dataset, with the given base value for stacking.

```

#### 47.10.6 Other Methods

To calculate the total of all y-values for a given x-value:

```

→ public static double calculateStackTotal(TableXYDataset dataset, int item); [1.0.5]
Returns the sum of all y-values for the data items with the specified index. In a TableXYDataset, all these items have the same x-value. This total is used by some renderers to convert data values to percentages.

```

To check if a dataset contains any non-null values:

```

→ public static boolean isEmptyOrNull(CategoryDataset dataset);
Returns true if the dataset is empty or null, and false otherwise. This requires iterating through (possibly all of) the values in the dataset.

→ public static boolean isEmptyOrNull(XYDataset dataset);
Returns true if the dataset is empty or null, and false otherwise. This requires iterating through (possibly all of) the values in the dataset.

```

#### See Also

[DomainInfo](#), [RangeInfo](#), [DataUtilities](#).

## 47.11 DefaultKeyValueDataset

### 47.11.1 Overview

A dataset that contains a single (*key, value*) data item. This class implements the [KeyedValueDataset](#) interface.

### 47.11.2 Usage

This class does not get used by JFreeChart.

## 47.12 DefaultKeyedValuesDataset

### 47.12.1 Overview

A dataset that implements the [KeyedValuesDataset](#) interface.

### 47.12.2 Notes

This dataset extends the [DefaultPieDataset](#) class without modification—it exists for completeness sake, to follow the naming pattern established for related classes and interfaces.

## 47.13 DefaultKeyedValues2DDataset

### 47.13.1 Overview

A default implementation of the [KeyedValues2DDataset](#) interface.

## 47.14 DefaultPieDataset

### 47.14.1 Overview

A dataset that records zero, one or many values, each with an associated key. This class provides a default implementation of the [PieDataset](#) interface and can, of course, be used in the creation of pie charts (refer to the [PiePlot](#) class).

### 47.14.2 Constructors

To create a new pie dataset:

- `public DefaultPieDataset();`  
Creates a new dataset, initially empty.
- `public DefaultPieDataset(KeyedValues data);`  
Creates a new dataset by copying the values (and associated keys) from `data`. If `data` is `null`, this constructor throws an `IllegalArgumentException`.

### 47.14.3 Reading the Dataset

The following methods support reading data values from the dataset:

- `public int getItemCount();`  
Returns the number of items (key-value pairs) in the dataset.
- `public List getKeys();`  
Returns an unmodifiable list of the keys in the dataset. If there are no items in the dataset, an empty list is returned (rather than `null`).

```
→ public Comparable getKey(int item);
Returns the key for the given item index. If item is less than zero, this method throws an
IndexOutOfBoundsException, but if item is greater than getItemCount() - 1, this method returns
null.

→ public int getIndex(Comparable key);
Returns the index for the given key, or -1 if the key is not recognised. If key is null, this
method throws an IllegalArgumentException.
```

To retrieve a value from the dataset:

```
→ public Number getValue(int item);
Returns the value (possibly null) for the given item.

→ public Number getValue(Comparable key);
Returns the value associated with a key (possibly null). This method throws an UnknownKeyException
if key is not defined in the dataset, and an IllegalArgumentException if key is null.
```

#### 47.14.4 Updating the Dataset

To set the value associated with a key:

```
→ public void setValue(Comparable key, Number value);
Sets the value associated with a key (the value can be null). If the key already exists within
the dataset, its value is updated. If the key doesn't already exist, a new item is added to the
dataset. After the dataset is updated, a DatasetChangeEvent is sent to all registered listeners.

→ public void setValue(Comparable key, double value);
As for the preceding method. This is a convenience method that creates a Number instance using
value then calls the other setValue() method.
```

To remove an item from the dataset:

```
→ public void remove(Comparable key);
Removes the item with the specified key from the dataset and sends a DatasetChangeEvent to
all registered listeners. If there is no item with the specified key, this method does nothing. If
key is null, this method throws an IllegalArgumentException.
```

To remove all items from the dataset:

```
→ public void clear();
Removes all items from the dataset and sends a DatasetChangeEvent to all registered listeners.
If the dataset is already empty, this method does nothing.
```

#### 47.14.5 Equals, Cloning and Serialization

To test this dataset for equality with an arbitrary object:

```
→ public boolean equals(Object obj);
Returns true if obj:


- is not null;
- is an instance of PieDataset;
- contains the same keys and values in the same order as this dataset;


...otherwise this method returns false.
```

This class implements `Cloneable` (and `PublicCloneable`), but note that the registered listeners are not copied across to the clone.

This class is `Serializable`.

#### 47.14.6 Notes

Some points to note:

- the dataset is permitted to contain `null` and/or negative values.

**See Also**

[PieDataset](#), [PiePlot](#).

## 47.15 DefaultValueDataset

### 47.15.1 Overview

A dataset that contains a single (possibly `null`) value. This class provides a default implementation of the `ValueDataset` interface and is used in JFreeChart by the `MeterPlot` and `ThermometerPlot` classes.

### 47.15.2 Constructors

To create a new instance, use one of the following constructors:

- ▶ `public DefaultValueDataset();`  
Creates a new instance containing a `null` value.
- ▶ `public DefaultValueDataset(double value);`  
Creates a new instance containing the specified `value`.
- ▶ `public DefaultValueDataset(Number value);`  
Creates a new instance containing the specified `value` (which may be `null`).

### 47.15.3 Methods

To access the single value maintained by the dataset:

- ▶ `public Number getValue();`  
Returns the dataset's value, which may be `null`.
- ▶ `public void setValue(Number value);`  
Sets the dataset's value (`null` is permitted) and sends a `DatasetChangeEvent` to all registered listeners.

### 47.15.4 Equals, Cloning and Serialization

To test this dataset for equality with an arbitrary object:

- ▶ `public boolean equals(Object obj);`  
Returns `true` if `obj`:
  - is not `null`;
  - is an instance of `ValueDataset`;
  - contains the same value as this dataset....otherwise returns `false`.

Instances of this class can be cloned (`PublicCloneable` is implemented), but note that registered listeners are not copied across to the clone.

This class is `Serializable`.

**See Also**

[ValueDataset](#).

## 47.16 KeyedValueDataset

### 47.16.1 Overview

A dataset that contains a single (*key, value*) data item, where the key is an instance of `Comparable` and the value is an instance of `Number`.

### 47.16.2 Methods

This interface extends the [KeyedValue](#) and [Dataset](#) interfaces, and adds no additional methods.

### 47.16.3 Notes

There are currently no charts that specifically require this type of dataset.

## 47.17 KeyedValuesDataset

### 47.17.1 Overview

A *keyed values dataset* is a collection of values where each value is associated with a key. A common use for this type of dataset is in the creation of pie charts.

### 47.17.2 Methods

This interface adds no methods to those it inherits from the [KeyedValues](#) and [Dataset](#) interfaces.

## 47.18 KeyedValues2DDataset

### 47.18.1 Overview

This interface is equivalent to the [CategoryDataset](#) interface. It has been included for completeness in so far as it continues the sequence of names [KeyedValueDataset](#), [KeyedValuesDataset](#), [KeyedValues2DDataset](#).

## 47.19 PieDataset

### 47.19.1 Overview

A *pie dataset* is a collection of values where each value is associated with a key. This type of dataset is most commonly used to create pie charts. As with all the dataset interfaces in JFreeChart, only data reading (not writing) methods are defined. The [DefaultPieDataset](#) class provides a default implementation of this interface.

### 47.19.2 Interface Methods

This interface defines the following methods:

- **public Comparable getKey(int index);**  
Returns the key with the specified index. This method should throw an [IndexOutOfBoundsException](#) if the index is not in the range 0 to `getItemCount() - 1`.
- **public int getIndex(Comparable key);**  
Returns the index of the specified key, or `-1` if there is no such key. This method should throw an [IllegalArgumentException](#) if `key` is `null`.
- **public List getKeys();**  
Returns an unmodifiable list (never `null`) of the keys in the dataset.
- **public Number getValue(Comparable key);**  
Returns the value (possibly `null`) associated with the specified key. This method should throw an [UnknownKeyException](#) if `key` is not defined in the dataset, and an [IllegalArgumentException](#) if `key` is `null`.

### 47.19.3 Notes

Some points to note:

- the `DefaultPieDataset` class provides one implementation of this interface.
- the `DatasetUtilities` class includes some methods for creating a `PieDataset` by slicing a `CategoryDataset` either by row or column.
- you can read a `PieDataset` from a file (in a prespecified XML format) using the `DatasetReader` class.

#### See Also

[CategoryToPieDataset](#), [PiePlot](#).

## 47.20 Series

### 47.20.1 Overview

A useful base class for implementing data series, subclasses include `TimeSeries` and `XYSeries`. This class provides a mechanism for registering *change listeners*, objects that will receive a message (a `SeriesChangeEvent`) every time the series is modified in some way.

### 47.20.2 Constructor

This (abstract) class has two constructors:

► `protected Series(Comparable key);`

Creates a new series with the specified `key`, which should be a unique identifier for the series. This constructor throws an `IllegalArgumentException` if `key` is `null`. In general, you should ensure that the `toString()` method for the `key` returns something readable, because this form is often displayed in charts (in the legend, for example). Since `String` implements the `Comparable` interface, it is very common to just use a `String` for the `key`.

► `protected Series(Comparable key, String description);`

Creates a new series with the specified `key` (identifier) and description. This constructor throws an `IllegalArgumentException` if `key` is `null`.

These constructors are `protected` since you do not create a `Series` directly, but via a subclass.

### 47.20.3 Methods

Every series has a unique, non-`null`, identifier:

► `public Comparable getKey();`

Returns the identifier for the series (never `null`). Calling `toString()` on the returned `key` should give a readable identifier for the series.

► `public void setKey(Comparable key);`

Sets the identifier for the series and sends a `PropertyChangeEvent` (with the property name `key`) to all registered listeners. This method throws an `IllegalArgumentException` if `key` is `null`.

Every series has an optional description:

► `public String getDescription();`

Returns a description for the series (defaults to `null`). This is not currently used by JFreeChart.

► `public void setDescription(String description);`

Sets the description for the series (`null` is permitted) and sends a `PropertyChangeEvent` (with the property name `Description`) to all registered listeners.

To determine if the series contains any data:

► `public abstract int getItemCount();`  
 Returns the number of data items in the series.

► `public boolean isEmpty(); [1.0.7]`  
 Returns `true` if the series is empty (contains no data), and `false` otherwise.

#### 47.20.4 Property Change Listeners

A property change listener mechanism is used for the key and description properties (and, potentially, properties defined by subclasses):

► `public void addPropertyChangeListener(PropertyChangeListener listener);`  
 Registers a listener to receive property change events from this series.

► `public void removePropertyChangeListener(PropertyChangeListener listener);`  
 Deregisters a listener so that it no longer receives property change events from this series.

► `protected void firePropertyChange(String property, Object oldValue, Object newValue);`  
 Sends a `PropertyChangeEvent` with the given name and values to all registered property change listeners.

#### 47.20.5 Change Listeners

This base class provides a mechanism for notifying listeners of changes to the series content:

► `public void fireSeriesChanged();`  
 Sends a `SeriesChangeEvent` to all registered listeners, but only if `getNotify()` returns `true`. Subclasses should call this method (or otherwise handle event notification) every time the data in the series is modified (JFreeChart relies on this mechanism for automatically updating charts).

If you have a lot of changes to make to a series, you may not want a change event to be generated for *every* change. The `notify` flag can be used to enable/disable event notification:

► `public boolean getNotify();`  
 Returns `true` if listeners will be notified of changes to the series, and `false` otherwise.

► `public void setNotify(boolean notify);`  
 Sets the flag that controls whether or not listeners are notified of changes to this series. If `notify` is set to `true`, this method will send a change event to all registered listeners immediately.

To register a change listener (an object that wishes to receive notification whenever the series is changed):

► `public void addChangeListener(SeriesChangeListener listener);`  
 Registers the listener to receive `SeriesChangeEvent` notifications from this series.

To deregister a change listener:

► `public void removeChangeListener(SeriesChangeListener listener);`  
 Deregisters the listener so that it will no longer receive `SeriesChangeEvent` notifications from this series..

A utility method is provided for sending a change event to all registered listeners:

► `protected void notifyListeners(SeriesChangeEvent event);`  
 Sends `event` to all registered listeners.

### 47.20.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj);`

Tests this series for equality with an arbitrary object. Returns `true` if and only if:

- `obj` is an instance of `Series`;
- `obj` has the same `key` and `description` as this series.

The listeners registered with a `Series` are not considered in the equality test.

This class implements `Cloneable` and `Serializable`—subclasses need to be cloneable and serializable so that JFreeChart datasets can be cloned and serialized.

#### See Also

[AbstractSeriesDataset](#), [TimeSeries](#), [XYSeries](#).

## 47.21 SeriesChangeEvent

### 47.21.1 Overview

An event class that is passed to a `SeriesChangeListener` to notify it concerning a change to a `Series`. This is an important event for dataset classes that maintain a collection of series objects—the dataset needs to know when any of its data series has changed, so that it can forward a `DatasetChangeEvent` to its own listeners.

#### See Also

[SeriesChangeListener](#).

## 47.22 SeriesChangeListener

### 47.22.1 Overview

The interface through which series change notifications are posted. Typically a dataset will implement this interface to receive notification of any changes to the individual series in the dataset (which will normally be passed on as a `DatasetChangeEvent`).

### 47.22.2 Methods

This interface defines a single method:

► `public void seriesChanged(SeriesChangeEvent event);`

Receives notification when a series changes.

### 47.22.3 Notes

The `AbstractSeriesDataset` class implements this interface—it will generate a `DatasetChangeEvent` every time it receives notification of a `SeriesChangeEvent`.

## 47.23 SeriesDataset

### 47.23.1 Overview

A base interface that defines a dataset containing zero, one or many data series. This interface is implemented by the `AbstractSeriesDataset` class.

### 47.23.2 Methods

To find out how many series there are in a dataset:

```
➔ public int getSeriesCount();
```

Returns the number of series in the dataset (possibly zero).

To get the identifier for a series:

```
➔ public Comparable getSeriesKey(int series);
```

Returns the key that identifies the series with the specified index (`series` should be in the range 0 to `getSeriesCount() - 1`). This method should return a unique and non-null key for each series. Any instance of `Comparable` can be used as a series key. For labelling (in a chart legend, for instance) the `toString()` method is used to convert the key to a `String`. Note that `String` implements `Comparable`, so you can use instances of `String` as the series keys.

Classes that implement this index should throw an `IllegalArgumentException` if the `series` argument is not in the specified range.

To find the index for a series, given its key:

```
➔ public int indexOf(Comparable seriesKey);
```

Returns the index of the series with the specified key, or `-1` if there is no such series in the dataset. For the sake of backwards compatibility, classes that implement this method should accept a `null` key. However, a series cannot actually have a `null` key, so this method will always return `-1` for this input.

### 47.23.3 Notes

Some points to note:

- this interface is extended by [CategoryDataset](#) and [XYDataset](#).

## 47.24 SeriesException

### 47.24.1 Overview

A general exception that can be thrown by a `Series`.

For example, a time series will not allow duplicate time periods—attempting to add a duplicate time period will throw a `SeriesException`.

## 47.25 SubSeriesDataset

A specialised dataset implementation written by Bill Kelemen. To be documented.

## 47.26 ValueDataset

### 47.26.1 Overview

This interface specifies the API for a dataset representing a single value (`Number` object). A default implementation of this interface is provided by the [DefaultValueDataset](#) class.

### 47.26.2 Methods

This interface includes the following methods (all inherited from the `Value` and `Dataset` interfaces):

```
➔ public Number getValue();
```

Returns the value for the dataset (possibly `null`).

```
► public void addChangeListener(DatasetChangeListener listener);  
Adds a change listener to the dataset. All listeners will be notified whenever the dataset's value changes.
```

```
► public void removeChangeListener(DatasetChangeListener listener);  
Removes a change listener from the dataset so that it no longer receives notification of updates to the dataset's value.
```

As with all datasets in JFreeChart, you can assign a dataset to a group. This facility is not currently used by JFreeChart itself:

```
► public DatasetGroup getGroup();  
Returns the group that the dataset belongs to.
```

```
► public void setGroup(DatasetGroup group);  
Sets the group that the dataset belongs to.
```

### 47.26.3 Notes

Some points to note:

- this type of dataset is employed by the [MeterPlot](#) and [ThermometerPlot](#) classes.

## 47.27 WaferMapDataset

### 47.27.1 Overview

A dataset that can be used with the [WaferMapPlot](#) class.

# Chapter 48

## Package: org.jfree.data.io

### 48.1 Introduction

I/O related classes.

### 48.2 CSV

#### 48.2.1 Overview

To be documented.

# Chapter 49

## Package: org.jfree.data.jdbc

### 49.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

### 49.2 JDBC`CategoryDataset`

#### 49.2.1 Overview

A *category dataset* that reads data from a database via JDBC. The data is cached in memory, and can be refreshed at any time.

#### 49.2.2 Constructors

You can create an empty dataset that establishes its own connection to the database, ready for executing a query:

```
► public JDBCCategoryDataset(String url, String driverName,  
String userName, String password);  
Creates an empty dataset (no query has been executed yet) and establishes a database connection.
```

Alternatively, you can create an empty dataset that will use a pre-existing database connection:

```
► public JDBCCategoryDataset(Connection con);  
Creates an empty dataset (no query has been executed yet) with a pre-existing database connection.
```

If you want to initialise the data via the constructor, rather than creating an empty dataset:

```
► public JDBCCategoryDataset(Connection con, String query);  
Creates a dataset with a pre-existing database connection and executes the specified query.
```

#### 49.2.3 Methods

This class implements all the methods in the `CategoryDataset` interface (by inheriting them from `DefaultCategoryDataset`).

To refresh the data in the dataset, you need to execute a query against the database:

```
► public void executeQuery(String query);  
Refreshes the data (which is cached in memory) for the dataset by executing the specified query. The query can be any valid SQL that returns at least two columns, the first containing VARCHAR data representing categories, and the remaining columns containing numerical data.
```

You can re-execute the query at any time.

**See Also**

[CategoryDataset](#), [DefaultCategoryDataset](#).

## 49.3 JDBC PieDataset

### 49.3.1 Overview

A *pie dataset* that reads data from a database via JDBC. The data is cached in memory, and can be refreshed at any time.

### 49.3.2 Constructors

You can create an empty dataset that establishes its own connection to the database, ready for executing a query:

```
→ public JDBCPieDataset(String url, String driverName, String userName,
String password);
Creates an empty dataset (no query has been executed yet) and establishes a database connection.
```

Alternatively, you can create an empty dataset that will use a pre-existing database connection:

```
→ public JDBCPieDataset(Connection con);
Creates an empty dataset (no query has been executed yet) with a pre-existing database connection.
```

If you want to initialise the data via the constructor, rather than creating an empty dataset:

```
→ public JDBCPieDataset(Connection con, String query);
Creates a dataset with a pre-existing database connection and executes the specified query.
```

### 49.3.3 Methods

This class implements all the methods in the [PieDataset](#) interface (by inheriting them from [DefaultPieDataset](#)).

To refresh the data in the dataset, you need to execute a query against the database:

```
→ public void executeQuery(String query);
Refreshes the data (which is cached in memory) for the dataset by executing the specified query. The query can be any valid SQL that returns two columns, the first containing VARCHAR data representing categories, and the second containing numerical data.
```

You can re-execute the query at any time.

**See Also**

[PieDataset](#), [DefaultPieDataset](#).

## 49.4 JDBC XYDataset

### 49.4.1 Overview

An *XY dataset* that reads data from a database via JDBC. The data is cached in memory, and can be refreshed at any time.

#### 49.4.2 Constructors

You can create an empty dataset that establishes its own connection to the database, ready for executing a query:

```
↳ public JDBCXYDataset(String url, String driverName, String userName,
String password);
Creates an empty dataset (no query has been executed yet) and establishes a database connection.
```

Alternatively, you can create an empty dataset that will use a pre-existing database connection:

```
↳ public JDBCXYDataset(Connection con);
Creates an empty dataset (no query has been executed yet) with a pre-existing database connection.
```

If you want to initialise the data via the constructor, rather than creating an empty dataset:

```
↳ public JDBCXYDataset(Connection con, String query);
Creates a dataset with a pre-existing database connection and executes the specified query.
```

#### 49.4.3 Methods

This class implements all the methods in the [XYDataset](#) interface.

To refresh the data in the dataset, you need to execute a query against the database:

```
↳ public void executeQuery(String query);
Refreshes the data (which is cached in memory) for the dataset by executing the specified query. The query can be any valid SQL that returns at least two columns, the first containing numerical or date data representing x-values, and the remaining column(s) containing numerical data for each series (one series per column).
```

You can re-execute the query at any time.

#### 49.4.4 Notes

There is a demo application [JDBCXYChartDemo](#) in the JFreeChart demo collection that illustrates the use of this class.

#### See Also

[XYDataset](#).

# Chapter 50

## Package: org.jfree.data.statistics

### 50.1 Introduction

This package contains interfaces and classes for representing statistical datasets.

### 50.2 BoxAndWhiskerCalculator

#### 50.2.1 Overview

A utility class for calculating the statistics required for a box-and-whisker plot.

#### 50.2.2 Constructors

This class contains only static methods and, by design, cannot be instantiated.

#### 50.2.3 Methods

To calculate box-and-whisker statistics for a list of values:

```
➔ public static BoxAndWhiskerItem calculateBoxAndWhiskerStatistics(List values);  
Calculates a set of statistics (mean, median, quartiles Q1 and Q3, plus outliers) for a list of  
Number objects.
```

- if `values` is `null`, this method throws a `NullPointerException`;
- if `values` contains any `null` values, this method throws a `NullPointerException`.

To calculate the first quartile value:

```
➔ public static double calculateQ1(List values);  
Returns the first quartile boundary for a list of values. This method REQUIRES the list of  
values to be in ascending order.
```

To calculate the third quartile value:

```
➔ public static double calculateQ3(List values);  
Returns the first quartile boundary for a list of values. This method REQUIRES the list of  
values to be in ascending order.
```

#### 50.2.4 Notes

See also the `Statistics` class.

## 50.3 BoxAndWhiskerCategoryDataset

### 50.3.1 Overview

An interface that extends the `CategoryDataset` interface and returns the values required for a box-and-whisker chart. The dataset represents a two-dimensional table, where each cell in the table contains a complete set of statistics for one box-and-whisker item (a mean, median, quartile boundary values Q1 and Q3, plus information about outliers and farouts).

The `DefaultBoxAndWhiskerCategoryDataset` provides one implementation of this interface.

### 50.3.2 Methods

The interface provides a range of methods for reading the values from the dataset. No update methods are provided, since not every dataset implementation needs to be writeable.

To get the mean for one item in the dataset:

```
→ public Number getMeanValue(int row, int column);
Returns the mean value for an item.

→ public Number getMeanValue(Comparable rowKey, Comparable columnKey);
Returns the mean value for an item.
```

To get the median value for one item in the dataset:

```
→ public Number getMedianValue(int row, int column);
Returns the median value for an item.

→ public Number getMedianValue(Comparable rowKey, Comparable columnKey);
Returns the median value for an item.
```

To get the first quartile boundary value:

```
→ public Number getQ1Value(int row, int column);
Returns the first quartile boundary value.

→ public Number getQ1Value(Comparable rowKey, Comparable columnKey);
Returns the first quartile boundary value.
```

To get the third quartile boundary value:

```
→ public Number getQ3Value(int row, int column);
Returns the third quartile boundary value.

→ public Number getQ3Value(Comparable rowKey, Comparable columnKey);
Returns the third quartile boundary value.
```

To get the minimum regular value (everything lower than this is either an outlier or a farout):

```
→ public Number getMinRegularValue(int row, int column);
Returns the lowest regular value.

→ public Number getMinRegularValue(Comparable rowKey, Comparable columnKey);
Returns the lowest regular value.
```

To get the maximum regular value (everything higher than this is either an outlier or a farout):

```
→ public Number getMaxRegularValue(int row, int column);
Returns the highest regular value.

→ public Number getMaxRegularValue(Comparable rowKey, Comparable columnKey);
Returns the highest regular value.
```

To get the minimum outlier (everything lower than this is a farout value):

```
→ public Number getMinOutlier(int row, int column);
Returns the lowest outlier.
```

► public Number getMinOutlier(Comparable rowKey, Comparable columnKey);  
 Returns the lowest outlier.

To get the maximum outlier (everything higher than this is a farout value):

► public Number getMaxOutlier(int row, int column);  
 Returns the highest outlier.  
 ► public Number getMaxOutlier(Comparable rowKey, Comparable columnKey);  
 Returns the highest outlier.

To get a list of the outlier (and farout) values for an item in the dataset:

► public List getOutliers(int row, int column);  
 Returns a list of the outlier (and farout) values.  
 ► public List getOutliers(Comparable rowKey, Comparable columnKey);  
 Returns a list of the outlier (and farout) values.

## 50.4 BoxAndWhiskerItem

### 50.4.1 Overview

An object that records the statistics and values required for an item in a box-and-whisker plot:

- the mean value;
- the median value;
- the first quartile (Q1) boundary value;
- the third quartile (Q3) boundary value;
- the minimum regular value;
- the maximum regular value;
- the minimum outlier value;
- the maximum outlier value;
- a list of outlier values;

Instances of this class are immutable.

### 50.4.2 Constructors

To create a new instance:

► public BoxAndWhiskerItem(double mean, double median, double q1, double q3,  
 double minRegularValue, double maxRegularValue, double minOutlier,  
 double maxOutlier, List outliers); [1.0.7]  
 Creates a new instance with the specified values.  
 ► public BoxAndWhiskerItem(Number mean, Number median, Number q1, Number q3,  
 Number minRegularValue, Number maxRegularValue, Number minOutlier,  
 Number maxOutlier, List outliers);  
 Creates a new instance with the specified values.

### 50.4.3 General Methods

To get the data values associated with this item:

- ▶ `public Number getMean();`  
Returns the mean value, as specified in the constructor. This may be `null`.
- ▶ `public Number getMedian();`  
Returns the median value, as specified in the constructor. This may be `null`.
- ▶ `public Number getQ1();`  
Returns the Q1 value, as specified in the constructor. This may be `null`.
- ▶ `public Number getQ3();`  
Returns the Q3 value, as specified in the constructor. This may be `null`.
- ▶ `public Number getMinRegularValue();`  
Returns the minimum regular value, as specified in the constructor. This may be `null`.
- ▶ `public Number getMaxRegularValue();`  
Returns the maximum regular value, as specified in the constructor. This may be `null`.
- ▶ `public Number getMinOutlier();`  
Returns the minimum outlier, as specified in the constructor. This may be `null`.
- ▶ `public Number getMaxOutlier();`  
Returns the maximum outlier, as specified in the constructor. This may be `null`.
- ▶ `public List getOutliers();`  
Returns a list of the outliers, as specified in the constructor.

### 50.4.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- ▶ `public boolean equals(Object obj);`  
Tests this item for equality with an arbitrary object.

Instances of this class are immutable<sup>1</sup> and `Serializable`.

### 50.4.5 Notes

Some points to note:

- the `BoxAndWhiskerCalculator` class returns instances of this class from one of its methods;
- the `DefaultBoxAndWhiskerXYDataset` class uses this class to store individual data items.

## 50.5 BoxAndWhiskerXYDataset

### 50.5.1 Overview

An interface that is used to obtain data for a box-and-whisker plot using the `XYPlot` class. This interface extends `XYDataset`.

The `DefaultBoxAndWhiskerXYDataset` class provides one implementation of this interface.

---

<sup>1</sup>Therefore cloning is unnecessary.

### 50.5.2 Interface Methods

To get the mean value for an item:

```
→ public Number getMeanValue(int series, int item);
```

Returns the mean value.

To get the median value for an item:

```
→ public Number getMedianValue(int series, int item);
```

Returns the median value.

To get the first quartile boundary value:

```
→ public Number getQ1Value(int series, int item);
```

Returns the first quartile boundary value.

To get the third quartile boundary value:

```
→ public Number getQ3Value(int series, int item);
```

Returns the third quartile boundary value.

To get the minimum regular value:

```
→ public Number getMinRegularValue(int series, int item);
```

Returns the minimum regular value. Anything lower than this is either an outlier or a farout value.

To get the maximum regular value:

```
→ public Number getMaxRegularValue(int series, int item);
```

Returns the maximum regular value. Anything higher than this is either an outlier or a farout value.

To get the minimum outlier:

```
→ public Number getMinOutlier(int series, int item);
```

Returns the minimum outlier. Anything lower than this is a farout value.

To get the maximum outlier:

```
→ public Number getMaxOutlier(int series, int item);
```

Returns the maximum outlier. Anything higher than this is a farout value.

To get a list of the outlier values:

```
→ public List<Number> getOutliers(int series, int item);
```

Returns a list of the outlier (and farout) values for this item.

To get the outlier coefficient:

```
→ public double getOutlierCoefficient();
```

Returns the outlier coefficient (this is probably redundant).

To get the farout coefficient:

```
→ public double getFaroutCoefficient();
```

Returns the farout coefficient (this is probably redundant).

### 50.5.3 Notes

Some points to note:

- the [XYBoxAndWhiskerRenderer](#) requires a dataset that implements this interface.

## 50.6 DefaultBoxAndWhiskerCategoryDataset

### 50.6.1 Overview

A dataset that can be used to create a box-and-whisker plot using a [BoxAndWhiskerRenderer](#) on a [CategoryPlot](#). This class implements the [BoxAndWhiskerCategoryDataset](#) interface.

## 50.6.2 Constructor

This class defines a single constructor:

```
↳ public DefaultBoxAndWhiskerCategoryDataset();
Creates a new dataset, initially empty.
```

## 50.6.3 Data Access

To access a data item:

```
↳ public BoxAndWhiskerItem getItem(int row, int column);
Returns an item from the dataset.
```

```
↳ public Number getValue(Comparable rowKey, Comparable columnKey);
Equivalent to getMedianValue(rowKey, columnKey)—see below. This method is defined in the
CategoryDataset interface and mapped to the median value.
```

```
↳ public Number getValue(int row, int column);
As above.
```

```
↳ public Number getMeanValue(int row, int column);
Returns the mean value (possibly null) for the item at the specified row and column in the
dataset.
```

```
↳ public Number getMeanValue(Comparable rowKey, Comparable columnKey);
As above.
```

```
↳ public Number getMedianValue(int row, int column);
Returns the median value (possibly null) for the item at the specified row and column in the
dataset.
```

```
↳ public Number getMedianValue(Comparable rowKey, Comparable columnKey);
As above.
```

```
↳ public Number getQ1Value(int row, int column);
Returns the boundary value (possibly null) between the first and second quartiles in the dataset.
```

```
↳ public Number getQ1Value(Comparable rowKey, Comparable columnKey);
As above.
```

```
↳ public Number getQ3Value(int row, int column);
Returns the boundary value (possibly null) between the third and fourth quartiles in the
dataset.
```

```
↳ public Number getQ3Value(Comparable rowKey, Comparable columnKey);
As above.
```

```
↳ public Number getMinRegularValue(int row, int column);
Returns the minimum regular value (possibly null) for the item at the specified row and column
in the dataset.
```

```
↳ public Number getMinRegularValue(Comparable rowKey, Comparable columnKey);
As above.
```

```
↳ public Number getMaxRegularValue(int row, int column);
Returns the maximum regular value (possibly null) for the item at the specified row and column
in the dataset.
```

```
↳ public Number getMaxRegularValue(Comparable rowKey, Comparable columnKey);
As above.
```

```
↳ public Number getMinOutlier(int row, int column);
Returns the minimum outlier value (possibly null) for the item at the specified row and column
in the dataset.
```

```
↳ public Number getMinOutlier(Comparable rowKey, Comparable columnKey);
As above.
```

```
→ public Number getMaxOutlier(int row, int column);
Returns the maximum outlier value (possibly null) for the item at the specified row and column
in the dataset.

→ public Number getMaxOutlier(Comparable rowKey, Comparable columnKey);
As above.

→ public List getOutliers(int row, int column);
Returns a list of the outliers for the specified row and column in the dataset.

→ public List getOutliers(Comparable rowKey, Comparable columnKey);
As above.
```

#### 50.6.4 Row and Column Indices

Each row in the dataset contains the data for a series, and each column represents a category.

```
→ public int getRowCount();
Returns the number of rows (series) in the dataset.

→ public int getRowIndex(Comparable key);
Returns the index of the specified key.

→ public Comparable getKey(int row);
Returns the key for the specified row.

→ public List getRowKeys();
Returns an unmodifiable list of the row keys.

→ public int getColumnCount();
Returns the number of columns (categories) in the dataset.

→ public int getColumnIndex(Comparable key);
returns the index of the specified key.

→ public Comparable getColumnKey(int column);
Returns the key for the specified column.

→ public List getColumnKeys();
Returns an unmodifiable list of the column keys.
```

#### 50.6.5 Adding and Removing Data

To add an item to the dataset:

```
→ public void add(BoxAndWhiskerItem item, Comparable rowKey, Comparable columnKey);
Adds an item to the dataset using the specified row and column keys and sends a DatasetChangeEvent
to all registered listeners. The row key corresponds to the series and the column key corresponds
to the category.
```

For convenience, you can create a new item from a list of raw data values:

```
→ public void add(List list, Comparable rowKey, Comparable columnKey);
Adds an item to the dataset that summarises the raw data in the list. Any null or NaN values
in the list are ignored.
```

To remove an item from the dataset:

```
→ public void remove(Comparable rowKey, Comparable columnKey); [1.0.7]
Removes an item from the dataset and sends a DatasetChangeEvent to all registered listeners.
```

To remove a series from the dataset:

```
→ public void removeRow(int rowIndex); [1.0.7]
Removes an entire row (series) from the dataset and sends a DatasetChangeEvent to all registered
listeners.
```

► public void removeRow(Comparable rowKey); [1.0.7]  
As above.

To remove a category from the dataset:

► public void removeColumn(int columnIndex); [1.0.7]  
Removes an entire column (category) from the dataset and sends a [DatasetChangeEvent](#) to all registered listeners.  
► public void removeColumn(Comparable columnKey); [1.0.7]  
As above.

To clear all items from the dataset:

► public void clear(); [1.0.7]  
Clears all data from the dataset and sends a [DatasetChangeEvent](#) to all registered listeners.

### 50.6.6 Range Bounds

This dataset caches the lower and upper bounds for the data values, and implements the [RangeInfo](#) interface to provide easy access to this information:

► public double getRangeLowerBound(boolean includeInterval);  
Returns the lower bound of the data values (excluding outliers) in the dataset. The `includeInterval` flag is ignored.  
► public double getRangeUpperBound(boolean includeInterval);  
Returns the upper bound of the data values (excluding outliers) in the dataset. The `includeInterval` flag is ignored.  
► public [Range](#) getRangeBounds(boolean includeInterval);  
Returns the bounds of the data value (excluding outliers) in the dataset. The `includeInterval` flag is ignored.

### 50.6.7 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► public boolean equals(Object obj);  
Tests this dataset for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 50.6.8 Notes

There is a demo (`BoxAndWhiskerDemo1.java`) included in the JFreeChart demo collection.

## 50.7 DefaultBoxAndWhiskerXYDataset

### 50.7.1 Overview

A basic implementation of the [BoxAndWhiskerXYDataset](#) interface that records statistics against particular points in time, for a single series only. This is currently the only implementation included with JFreeChart.

### 50.7.2 Constructor

To create a new dataset, initially containing no data:

► public DefaultBoxAndWhiskerXYDataset(Comparable seriesKey);  
Creates a new dataset containing an empty series with the specified `seriesKey`. Although you can specify `null` for the series key, this is not recommended.

### 50.7.3 General Methods

To find general information about the dataset:

► `public int getSeriesCount();`

Returns the number of series in the dataset—for this implementation, the return value is always 1.

► `public Comparable getSeriesKey(int i);`

Returns the key for the specified series—since this dataset can only store one series, the method returns the key specified in the constructor, irrespective of the index value.

► `public int getItemCount(int series);`

Returns the number of items in the (single) series stored by this dataset. The `series` argument is ignored (it's defined in the interface because most datasets can define multiple series, but not this one).

The outlier and far-out co-efficients can be set via the following methods:

► `public double getOutlierCoefficient();`

Returns the outlier co-efficient. The default value is 1.5.

► `public void setOutlierCoefficient(double outlinerCoefficient);`

Sets the outlier co-efficient.

► `public double getFaroutCoefficient();`

Returns the farout co-efficient. The default value is 2.0.

► `public void setFaroutCoefficient(double faroutCoefficient);`

Sets the farout co-efficient.

### 50.7.4 Accessing the Data Values

To access an item from the dataset:

► `public BoxAndWhiskerItem getItem(int series, int item);`

Returns an item from the dataset. Since this dataset stores one series only, the `series` index is ignored.

To access individual values from each data item:

► `public Date getXDate(int series, int item);`

Returns the x-value (a date/time value) for the specified item. The `series` index is ignored.

► `public Number getX(int series, int item);`

Returns the x-value as a Long instance, representing the number of milliseconds since 1-Jan-1970 (the date/time encoding used by Java's `Date` class). Each time this method is called, a new Long instance is created, so you should avoid this method if possible.

► `public Number getY(int series, int item);`

Equivalent to `getMeanValue(series, item)`—see the next method.

► `public Number getMeanValue(int series, int item);`

Returns the mean value (possibly null) for the specified item. The `series` index is ignored.

► `public Number getMedianValue(int series, int item);`

Returns the median value (possibly null) for the specified item. The `series` index is ignored.

► `public Number getQ1Value(int series, int item);`

Returns the Q1 value (possibly null) for the specified item. The `series` index is ignored.

► `public Number getQ3Value(int series, int item);`

Returns the Q3 value (possibly null) for the specified item. The `series` index is ignored.

► `public Number getMinRegularValue(int series, int item);`

Returns the minimum regular value (possibly null) for the specified item. The `series` index is ignored.

```
→ public Number getMaxRegularValue(int series, int item);
Returns the maximum regular value (possibly null) for the specified item. The series index is ignored.
```

```
→ public Number getMinOutlier(int series, int item);
Returns the minimum outlier (possibly null) for the specified item. The series index is ignored.
```

```
→ public Number getMaxOutlier(int series, int item);
Returns the maximum outlier (possibly null) for the specified item. The series index is ignored.
```

```
→ public List<Object> getOutliers(int series, int item);
Returns a list of the outliers for the specified item. The series argument is ignored.
```

### 50.7.5 Adding Data

To add an item to the dataset:

```
→ public void add(Date date, BoxAndWhiskerItem item);
Adds an item to the dataset for the specified date/time, and sends a DatasetChangeEvent to all registered listeners. If date or item is null, this method throws an IllegalArgumentException.
```

Currently there are no methods for removing data items or clearing the dataset.

### 50.7.6 Range Bounds

For efficiency, this class implements the [RangeInfo](#) interface, and caches the appropriate bounding values:

```
→ public double getRangeLowerBound(boolean includeInterval);
Returns the lower bound of the range values, or Double.NaN.
```

```
→ public double getRangeUpperBound(boolean includeInterval);
Returns the upper bound of the range values, or Double.NaN.
```

```
→ public Range getRangeBounds(boolean includeInterval);
Returns the bounds of the range values, or null.
```

### 50.7.7 Notes

Some points to note:

- there are currently no methods to remove data from the dataset;
- a demo ([XYBoxAndWhiskerDemo1.java](#)) is included in the JFreeChart demo collection, to provide an example of this class being used.

## 50.8 DefaultMultiValueCategoryDataset

### 50.8.1 Overview

A dataset that represents a two-dimensional table where each cell in the table can hold zero, one or many numerical values. This class implements the [MultiValueCategoryDataset](#) interface. This class was first introduced in JFreeChart version 1.0.7.

### 50.8.2 Constructor

To create a new dataset:

```
→ public DefaultMultiValueCategoryDataset(); [1.0.7]
Creates a new empty dataset.
```

### 50.8.3 Methods

To add data to the dataset:

```
→ public void add(List values, Comparable rowKey, Comparable columnKey); [1.0.7]
    Adds the values in the given list to the dataset, replacing any existing values for the specified
    row and column. The values are copied from values, with any null and Double.NAN values
    in the list being ignored. If values, rowKey or columnKey are null, this method throws an
    IllegalArgumentException.
```

To get a list of values for a cell in the table:

```
→ public List getValues(int row, int column); [1.0.7]
    Returns a list containing Number objects belonging to the specified cell. The list is unmodifiable,
    and may be empty.

→ public List getValues(Comparable rowKey, Comparable columnKey); [1.0.7]
    Returns a list containing Number objects belonging to the specified cell. The list is unmodifiable,
    and may be empty.
```

The `getValue()` methods inherited from the [CategoryDataset](#) interface are implemented to return the mean of the list of values returned by `getValues()`:

```
→ public Number getValue(Comparable row, Comparable column); [1.0.7]
    Returns the mean of the values at the specified cell in the dataset.

→ public Number getValue(int row, int column); [1.0.7]
    Returns the mean of the values at the specified cell in the dataset.
```

### 50.8.4 Row and Column Keys

To get information about the rows in the dataset:

```
→ public int getRowCount(); [1.0.7]
    Returns the number of rows in the dataset.

→ public List getRowKeys(); [1.0.7]
    Returns an unmodifiable list of the row keys for this dataset.

→ public int getRowIndex(Comparable key); [1.0.7]
    Returns the index corresponding to the specified key.

→ public Comparable getRowKey(int row); [1.0.7]
    Returns the key corresponding to the specified row index.
```

To get information about the columns in the dataset:

```
→ public int getColumnCount(); [1.0.7]
    Returns the number of columns in the dataset.

→ public List getColumnKeys(); [1.0.7]
    Returns an unmodifiable list of the column keys for this dataset.

→ public int getColumnIndex(Comparable key); [1.0.7]
    Returns the index corresponding to the specified key.

→ public Comparable getColumnKey(int column); [1.0.7]
    Returns the key corresponding to the specified column.
```

### 50.8.5 Dataset Bounds

This dataset caches the lower and upper bounds for the values in the dataset:

```
→ public double getRangeLowerBound(boolean includeInterval); [1.0.7]
    Returns the lower bound for the values in the dataset. The includeInterval argument is ignored
    as it doesn't apply for this dataset.
```

► `public double getRangeUpperBound(boolean includeInterval); [1.0.7]`  
 Returns the upper bound for the values in the dataset. The `includeInterval` argument is ignored as it doesn't apply for this dataset.

► `public Range getRangeBounds(boolean includeInterval); [1.0.7]`  
 Returns the bounds for the values in the dataset. The `includeInterval` argument is ignored as it doesn't apply for this dataset.

### 50.8.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj); [1.0.7]`  
 Tests this dataset for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 50.8.7 Notes

Some points to note:

- there are currently no methods to remove data items from the dataset;
- the row and column keys can be any instance of `Comparable`, but it is typical to use a `String`;
- there is a demo (`ScatterRendererDemo1.java`) included in the JFreeChart demo collection.

#### See Also:

[MultiValueCategoryDataset](#)

## 50.9 DefaultStatisticalCategoryDataset

### 50.9.1 Overview

A dataset that stores mean and standard deviation values for each cell in a two dimensional table. Keys (instances of `Comparable`) are used to reference the rows and columns in the table. This class provides a default implementation of the `StatisticalCategoryDataset` interface.

### 50.9.2 Constructors

This class has just one constructor:

► `public DefaultStatisticalCategoryDataset();`  
 Creates a new instance containing no data.

### 50.9.3 General Methods

To find the number of rows in the dataset:

► `public int getRowCount();`  
 Returns the total number of rows in the dataset.

To find the number of columns in the dataset:

► `public int getColumnCount();`  
 Returns the total number of columns in the dataset.

### 50.9.4 Accessing Data

To access the value at a given cell in the table:

```
→ public Number getValue(int row, int column);  
Returns the value at a given cell in the table, which may be null. The value returned is the  
same mean value returned by the getMeanValue(int, int) method.  
→ public Number getValue(Comparable rowKey, Comparable columnKey);  
As for the previous method, but using row and column keys rather than indices.
```

To access the mean value:

```
→ public Number getMeanValue(int row, int column);  
Returns the mean value at a given cell in the table, which may be null.  
→ public Number getMeanValue(Comparable rowKey, Comparable columnKey);  
Returns the mean value at a given cell in the table, which may be null.
```

To access the standard deviation:

```
→ public Number getStdDevValue (int row, int column);  
Returns the standard deviation at a given cell in the table, which may be null.  
→ public Number getStdDevValue(Comparable rowKey, Comparable columnKey);  
Returns the standard deviation at a given cell in the table, which may be null.
```

### 50.9.5 Adding and Removing Data

To add a mean and standard deviation to the dataset:

```
→ public void add(double mean, double standardDeviation, Comparable rowKey, Comparable columnKey);  
Adds the specified mean and standard deviation to a cell in the table.  
→ public void add(Number mean, Number standardDeviation, Comparable rowKey, Comparable columnKey);  
As for the previous method.
```

To remove a data item from the dataset:

```
→ public void remove(Comparable rowKey, Comparable columnKey); [1.0.7]  
Removes an item from the dataset and sends a DatasetChangeEvent to all registered listeners.
```

To remove an entire row from the dataset:

```
→ public void removeRow(int rowIndex); [1.0.7]  
Removes the specified row from the dataset, and sends a DatasetChangeEvent to all registered  
listeners.  
→ public void removeRow(Comparable rowKey); [1.0.7]  
Removes the specified row from the dataset, and sends a DatasetChangeEvent to all registered  
listeners.
```

To remove an entire column from the dataset:

```
→ public void removeColumn(int columnIndex); [1.0.7]  
Removes the specified column from the dataset, and sends a DatasetChangeEvent to all registered  
listeners.  
→ public void removeColumn(Comparable columnKey); [1.0.7]  
Removes the specified column from the dataset, and sends a DatasetChangeEvent to all registered  
listeners.
```

To remove all data from the dataset:

```
→ public void clear(); [1.0.7]  
Removes all data from the dataset and sends a DatasetChangeEvent to all registered listeners.
```

### 50.9.6 Row and Column Keys

The following methods provide information about the column keys:

- `public int getColumnIndex(Comparable key);`  
Returns the column index for the specified key.
- `public Comparable getColumnKey(int column);`  
Returns the key for the specified column.
- `public List getColumnKeys();`  
Returns a list of the column keys.

The following methods provide information about the row keys:

- `public int getRowIndex(Comparable key);`  
Returns the row index for the specified key.
- `public Comparable getRowKey(int row);`  
Returns the key for the specified row.
- `public List getRowKeys();`  
Returns a list of the row keys.

### 50.9.7 Other Methods

This dataset implements the [RangeInfo](#) interface:

- `public Range getRangeBounds(boolean includeInterval);`  
Returns the range of values for this dataset. If `includeInterval` is true, the standard deviation is included in the range.
- `public double getRangeLowerBound(boolean includeInterval);`  
Returns the lower bound of the values for this dataset. If `includeInterval` is true, the standard deviation is included in the range.
- `public double getRangeUpperBound(boolean includeInterval);`  
Returns the upper bound of the values for this dataset. If `includeInterval` is true, the standard deviation is included in the range.

### 50.9.8 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this dataset for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

### 50.9.9 Notes

Some points to note:

- this class is used in a couple of demos in the JFreeChart demo collection—see `StatisticalBarChartDemo1.java` and `StatisticalLineChartDemo1.java`.

#### See Also

[StatisticalCategoryDataset](#).

## 50.10 HistogramBin

### 50.10.1 Overview

This class is used to represent a bin for the [HistogramDataset](#) class.

### 50.10.2 Constructor

To create a new bin:

```
→ public HistogramBin(double startBoundary, double endBoundary);
Creates a new bin with the specified boundary values. If startBoundary is greater than endBoundary,
this method throws an IllegalArgumentException.
```

### 50.10.3 Methods

This class defines the following methods:

```
→ public int getCount();
Returns the number of items in the bin.

→ public void incrementCount();
Increments the count for the bin.

→ public double getStartBoundary();
Returns the start (or lower) boundary for the bin.

→ public double getEndBoundary();
Returns the end (or upper) boundary for the bin.

→ public double getBinWidth();
Returns the bin width, which is calculated as the difference between the start and end boundary
values.
```

### 50.10.4 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

```
→ public boolean equals(Object obj);
Tests this bin for equality with an arbitrary object.
```

Instances of this class are cloneable and serializable.

## 50.11 HistogramDataset

### 50.11.1 Overview

A dataset that can be used with the `XYPlot` and `XYBarRenderer` classes to display a histogram. Three histogram types are supported:

- `FREQUENCY` – displays the number of items falling into each bin range;
- `RELATIVE_FREQUENCY` – displays the percentage of items falling into each bin range;
- `SCALE_AREA_TO_1` – adjusts the bin values so that the overall area of the histogram is 1.0.

This class has some overlap (in the features it provides) with the `SimpleHistogramDataset` class.

### 50.11.2 Constructors

The default constructor creates an empty dataset:

```
→ public HistogramDataset();
Creates an empty dataset with a type of HistogramType.FREQUENCY. You can change the type at
any time by calling the setType() method.
```

### 50.11.3 The Histogram Type

To set the type of histogram:

```
➔ public HistogramType getType();
```

Returns the histogram type.

```
➔ public void setType(HistogramType type);
```

Sets the histogram type and sends a `DatasetChangeEvent` to all registered listeners. If `type` is `null`, this method throws an `IllegalArgumentException`.

### 50.11.4 Adding Data to the Dataset

To add raw data to the dataset, allowing the bin range to be determined automatically to fit the data:

```
➔ public void addSeries(String name, double[] values, int bins);
```

Creates a series within the dataset that summarises the values supplied by allocating them to the specified number of bins. The bin size is calculated to cover the range of values in the array.

To add raw data to the dataset, using a specified bin range:

```
➔ public void addSeries(String name, double[] values, int bins,
double minimum, double maximum);
```

Creates a series within the dataset that summarises the values supplied by allocating them to bins. The bin size is calculated so that the specified number of bins covers the range (*minimum*, *maximum*). If `name` or `values` is `null`, or `bins` is less than 1, this method throws an `IllegalArgumentException`.

For both of the above methods, values that fall outside the bin range will be allocated to the first or last bin, whichever is closer. Values that fall on a bin boundary will be allocated to the *higher* bin.

An important point to note is that the dataset stores the frequency values (bin counts) only, and not the raw data passed to the `addSeries()` methods.

### 50.11.5 Accessing the Dataset Values

This dataset works by returning the appropriate bin dimensions via the `IntervalXYDataset` interface. Most of the methods listed in this section come from that interface.

All methods that accept a `series` argument expect a value in the range 0 to `getIndex() - 1`. If a value outside that range is supplied, an `IndexOutOfBoundsException` will be thrown.

To get the number of series in the dataset:

```
➔ public int getSeriesCount();
```

Returns the number of series in the dataset.

To get the key for a series:

```
➔ public Comparable getSeriesKey(int series);
```

Returns the key for the specified series.

To get the number of items in a series:

```
➔ public int getItemCount(int series);
```

Returns the number of items in the specified series.

To get the data values:

```
➔ public Number getX(int series, int item);
```

Returns the x-value for the specified item. This value represents the center of a bin. See `getStartX()` and `getEndX()` for the bin boundaries.

```
→ public Number getY(int series, int item);
Returns the y-value for the specified item. The value returned is derived from the bin count
according to the dataset type.

→ public Number getStartX(int series, int item);
Returns the lower bound of the range of x-values for the specified item. This corresponds to
the start of the bin for this item.

→ public Number getEndX(int series, int item);
Returns the upper bound of the range of x-values for the specified item. This corresponds to
the end of the bin for this item.

→ public Number getStartY(int series, int item);
Returns the same value as getY().

→ public Number getEndY(int series, int item);
Returns the same value as getY().
```

### 50.11.6 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

```
→ public boolean equals(Object obj);
Tests this dataset for equality with an arbitrary object. This method returns true if:


- obj is not null;
- obj is an instance of HistogramDataset,2
- both datasets contain the same values.

```

Instances of this class are `Cloneable` and `Serializable`.

### 50.11.7 Notes

Some points to note:

- a demo (`HistogramDemo1.java`) is included in the JFreeChart demo collection;
- an alternative implementation that you may find easier to use is [SimpleHistogramDataset](#).

## 50.12 HistogramType

### 50.12.1 Overview

An enumeration of the possible histogram types:

- **FREQUENCY** - a *frequency histogram* shows the number of data items allocated to each bin;
- **RELATIVE\_FREQUENCY** - a *relative frequency histogram* shows the number of data items allocated to each bin as a fraction of the total number of items;
- **SCALE\_AREA\_TO\_1** - similar to a relative frequency histogram, except that the values are scaled so that the overall area represented by the bars is equal to 1.

### 50.12.2 Usage

These values are normally used in the `getType()` and `setType()` methods of the [HistogramDataset](#) class.

---

<sup>2</sup>This needs to be reviewed. In other dataset classes, we test equality against interfaces only—it should probably be the same here.

## 50.13 MeanAndStandardDeviation

### 50.13.1 Overview

A simple class that records the mean and standard deviation for some data. The source data is not known to this class, so the mean and standard deviation values have to be supplied/calculated by external code. This class is used in the [DefaultStatisticalCategoryDataset](#) implementation.

### 50.13.2 Constructors

To create a new instance:

- ▶ `public MeanAndStandardDeviation(double mean, double standardDeviation);`  
Creates a new record with the specified mean and standard deviation.
- ▶ `public MeanAndStandardDeviation(Number mean, Number standardDeviation);`  
Creates a new record with the specified mean and standard deviation (`null` is permitted for either argument).

### 50.13.3 General Methods

To access the mean value:

- ▶ `public Number getMean();`  
Returns the mean, which may be `null`.
- ▶ `public double getMeanValue(); [1.0.7]`  
Returns the mean as a `double` primitive. If `getMean()` returns `null`, this method returns `Double.NaN`.

To access the standard deviation value:

- ▶ `public Number getStandardDeviation();`  
Returns the standard deviation, which may be `null`.
- ▶ `public double getStandardDeviationValue(); [1.0.7]`  
Returns the standard deviation as a `double` primitive. If `getStandardDeviation()` returns `null`, this method returns `Double.NaN`.

### 50.13.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- ▶ `public boolean equals(Object obj);`  
Tests this record for equality with an arbitrary object. This method returns `true` if `obj` is an instance of `MeanAndStandardDeviation` that records the same mean and standard deviation value as this object.

Instances of this class are `Serializable` but, by virtue of being immutable, not `Cloneable`.

### 50.13.5 Notes

This class is used in the [DefaultStatisticalCategoryDataset](#) implementation.

## 50.14 MultiValueCategoryDataset

### 50.14.1 Overview

A dataset that represents a two-dimensional table where each cell in the table can hold zero, one or many numerical values. This interface extends the [CategoryDataset](#) interface. This interface was first introduced in JFreeChart version 1.0.7.

### 50.14.2 Interface Methods

In addition to the methods defined by `CategoryDataset`:

- ↳ `public List getValues(int row, int column); [1.0.7]`  
Returns a list containing `Number` objects belonging to the specified cell. The list may be empty.
- ↳ `public List getValues(Comparable rowKey, Comparable columnKey); [1.0.7]`  
Returns a list containing `Number` objects belonging to the specified cell. The list may be empty.

### 50.14.3 Notes

Some points to note:

- a standard implementation is provided by `DefaultMultiValueCategoryDataset`;
- the `ScatterRenderer` class requires a dataset that implements this interface;
- there is a demo (`ScatterRendererDemo1.java`) included in the JFreeChart demo collection.

## 50.15 Regression

### 50.15.1 Overview

This class provides some utility methods for calculating regression co-efficients. Two regression types are supported:

- *ordinary least squares (OLS)* regression - fitting a line of the form  $y = ax + b$ ;
- *power* regression - fitting a line of the form  $y = ax^b$ .

Figure 50.1 shows an example created using this utility class.

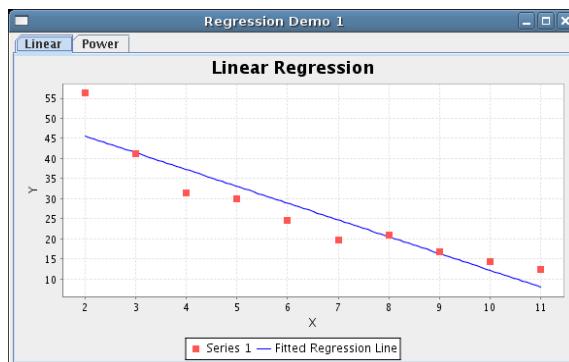


Figure 50.1: A chart displaying a fitted line

### 50.15.2 Methods

To calculate the OLS regression for an array of data values:

- ↳ `public static double[] getOLSRegression(double[][] data);`  
Fits a line of the form  $y = a + bx$  to the given data. The x values are read from `data[i][0]` and the y values are read from `data[i][1]`. There must be at least two items in the array. The result is a new array containing two values, the intercept ( $a$ ) and the slope ( $b$ ).

```
↳ public static double[] getOLSRegression(XYDataset dataset, int series);
```

Fits a line of the form  $y = a + bx$  to the specified series in the dataset (which must contain at least two items). The result is a new array containing two values, the intercept ( $a$ ) and the slope ( $b$ ).

To calculate a power regression for an array of data values:

```
↳ public static double[] getPowerRegression(double[][] data);
```

Performs a power regression on the data. The result is an array containing two values ( $a$  and  $b$ ) from the equation  $y = ax^b$ .

```
↳ public static double[] getPowerRegression(XYDataset dataset, int series);
```

Performs a power regression on the specified series in the dataset. The result is an array containing two values ( $a$  and  $b$ ) from the equation  $y = ax^b$ .

### 50.15.3 Notes

Some points to note:

- no other regression types are supported at present;
- a demo application (`RegressionDemo1.java`) is included in the JFreeChart demo collection.

## 50.16 SimpleHistogramBin

### 50.16.1 Overview

A bin for recording item counts in a `SimpleHistogramDataset`.

### 50.16.2 Constructors

There are two constructors:

```
↳ public SimpleHistogramBin(double lowerBound, double upperBound);
```

Creates a new bin representing the given range (inclusive of the bounds).

```
↳ public SimpleHistogramBin(double lowerBound, double upperBound, boolean includeLowerBound,
                           boolean includeUpperBound);
```

Creates a new bin representing the given range.

### 50.16.3 Methods

To find the bounds for the bin:

```
↳ public double getLowerBound();
```

Returns the lower bound for the bin range.

```
↳ public double getUpperBound();
```

Returns the upper bound for the bin range.

To access the bin's item count:

```
↳ public int getItemCount();
```

Returns the item count for the bin.

```
↳ public void setItemCount(int count);
```

Sets the item count for the bin.

To check if a value belongs to a bin:

```
↳ public boolean accepts(double value);
```

Returns `true` if the given value falls within the bin range, and `false` otherwise.

To determine if two bins overlap:

► public boolean overlapsWith(SimpleHistogramBin bin);  
 Returns true if the given bin overlaps with this bin, and false otherwise.

The following method is used to determine an ordering for a collection of bins:

► public int compareTo(Object obj);  
 Returns the relative order of this bin compared to some object.

#### 50.16.4 Equals, Cloning and Serialization

This class overrides the equals() method:

► public boolean equals(Object obj);  
 Tests the bin for equality with an arbitrary object.

This class is cloneable and serializable.

### 50.17 SimpleHistogramDataset

#### 50.17.1 Overview

A dataset that can be used to create a simple histogram. This is an alternative implementation to the [HistogramDataset](#) class.

#### 50.17.2 Constructor

To create a new dataset:

► public SimpleHistogramDataset(Comparable key);  
 Creates a new dataset, initially empty. The key identifies the series for the dataset—most datasets allow multiple series, but this one allows only one. If key is null, this constructor throws an [IllegalArgumentException](#).<sup>3</sup>

#### 50.17.3 Methods

This dataset can only hold a single data series:

► public int getSeriesCount();  
 Always returns 1.

► public Comparable getSeriesKey(int series);  
 Returns the key used for the single data series stored by this dataset. This key is never null.

The adjustForBinSize flag controls whether or not the bin count is divided by the bin size (width) when returning the y-value for the dataset:

► public boolean getAdjustForBinSize();  
 Returns true if the bin count is adjusted for the bin size, and false otherwise.

► public void setAdjustForBinSize(boolean adjust);  
 Sets the flag that controls whether or not the bin count is adjusted for the bin size.

► public DomainOrder getDomainOrder();  
 Returns DomainOrder.ASCENDING to indicate that the domain values are supplied in ascending order. Some renderers may use this knowledge to optimise the drawing of charts when only a subset of the values is visible.

► public int getItemCount(int series);  
 Returns the number of items in the specified series (note that this dataset can only contain one series).

---

<sup>3</sup>As of version 1.0.7.

```
→ public void addBin(SimpleHistogramBin bin);
    Adds a bin to the dataset. You need to ensure that the bin doesn't overlap any existing bins.

→ public void addObservation(double value);
    Adds a single observation to the appropriate bin.

→ public void addObservation(double value, boolean notify);
    Adds a single observation to the dataset, assigning it to the appropriate bin. The notify flag
    controls whether or not a DatasetChangeEvent is sent to all registered listeners.

→ public void addObservations(double[] values);
    Adds all the values to the dataset and then sends a DatasetChangeEvent to all registered listeners.
```

#### 50.17.4 Dataset Methods

The following methods are specified by the `IntervalXYDataset` interface:

```
→ public Number getX(int series, int item);
    Returns the x-value for an item.

→ public double getXValue(int series, int item);
    Returns the x-value for an item.

→ public Number getY(int series, int item);
    Returns the y-value for an item.

→ public double getYValue(int series, int item);
    Returns the y-value for an item, as a double primitive.

→ public Number getStartX(int series, int item);
    Returns the start of the x-interval for an item.

→ public double getStartXValue(int series, int item);
    Returns the start of the x-interval for an item.

→ public Number getEndX(int series, int item);
    Returns the end value of the x-interval for an item.

→ public double getEndXValue(int series, int item);
    Returns the end value of the x-interval for an item.

→ public Number getStartY(int series, int item);
    This method is mapped to the getY() method.

→ public double getStartYValue(int series, int item);
    This method is mapped to the getYValue() method.

→ public Number getEndY(int series, int item);
    This method is mapped to the getY() method.

→ public double getEndYValue(int series, int item);
    This method is mapped to the getYValue() method.
```

#### 50.17.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj);
    Tests this dataset for equality with an arbitrary object.
```

This class is `Cloneable` and `Serializable`.

#### 50.17.6 Notes

Some points to note:

- a demo (`HistogramDemo2.java`) showing the use of this dataset is included in the JFreeChart demo collection.

## 50.18 StatisticalCategoryDataset

### 50.18.1 Overview

A *statistical category dataset* is a table of data where each data item consists of a mean and a standard deviation (calculated externally on the basis of some other data). This interface is an extension of the [CategoryDataset](#) interface.

This interface is used by the following renderers:

- [StatisticalBarRenderer](#);
- [StatisticalLineAndShapeRenderer](#).

The [DefaultStatisticalCategoryDataset](#) class provides a default implementation of this interface.

### 50.18.2 Interface Methods

This interface extends the [CategoryDataset](#) interface, adding four methods.

To get the mean value for an item in the dataset, using row and column indices:

```
➔ public Number getMeanValue(int row, int column);
```

Returns the mean value (possibly `null`) for one cell in the table.

Alternatively, you can access the same value using the row and column keys:

```
➔ public Number getMeanValue(Comparable rowKey, Comparable columnKey);
```

Returns the mean value (possibly `null`) for one cell in the table.

To get the standard deviation value for an item in the dataset, using row and column indices:

```
➔ public Number getStdDevValue(int row, int column);
```

Returns the standard deviation (possibly `null`) for one cell in the table.

As with the mean value, you can also access the standard deviation using the row and column keys:

```
➔ public Number getStdDevValue(Comparable rowKey, Comparable columnKey);
```

Returns the standard deviation (possibly `null`) for one cell in the table.

### 50.18.3 Notes

Some points to note:

- JFreeChart includes one implementation of this dataset interface—see the [DefaultStatisticalCategoryDataset](#) class.

## 50.19 Statistics

### 50.19.1 Overview

Provides some static utility methods for calculating statistics.

### 50.19.2 Mean and Median

To calculate the mean of an array of non-null values:

```
➔ public static double calculateMean(Number[] values);
```

Calculates and returns the mean of an array of `Number` objects. This is equivalent to `calculateMean(values, true)`—see below.

► **public static double calculateMean(Number[] values, boolean includeNullAndNaN); [1.0.3]**  
 Calculates and returns the mean of an array of values. The `includeNullAndNaN` flag determines whether or not `null` and `NaN` items are included in the calculation—if either is present in the array, the end result is always `Double.NaN`.

- if `values` is `null`, this method throws an `IllegalArgumentException`;
- if `values` is empty (that is, has zero length), this method returns `Double.NaN`.

To calculate the mean of a collection of `Number` objects:

► **public static double calculateMean(Collection values);**  
 Returns the mean of a collection of `Number` objects. This is equivalent to `calculateMean(values, true)`.  
 ► **public static double calculateMean(Collection values, boolean includeNullAndNaN); [1.0.3]**  
 Returns the mean of a collection of `Number` objects. The `includeNullAndNaN` flag determines whether or not `null` and `NaN` items are included in the calculation—if either is present, the end result is `Double.NaN`.

- if `values` is `null`, this method throws an `IllegalArgumentException`;
- if `values` is empty, this method returns `Double.NaN`;
- objects in the collection that are not instances of `Number` (including `null`) are ignored.

To calculate the median of a list of `Number` objects:

► **public static double calculateMedian(List values);**  
 Equivalent to `calculateMedian(values, true)`—see below.  
 ► **public static double calculateMedian(List values, boolean copyAndSort);**  
 Returns the median of a list of `Number` objects. If the list is presented in ascending order by value, you can set the `copyAndSort` flag to `false`, in which case the median is found by examining the `values` list directly. Otherwise, the `copyAndSort` flag must be set to `true`, which causes the list of values to be copied and sorted prior to finding the median value.

- if the list contains `null` values, this method will throw a `NullPointerException`;
- if the list contains non-`null` objects that are not `Number` instances, this method will throw a `ClassCastException`.

To calculate the median of a sub-list of `Number` objects:

► **public static double calculateMedian(List values, int start, int end);**  
 Equivalent to `calculateMedian(values, start, end, true)`—see below.  
 ► **public static double calculateMedian(List values, int start, int end, boolean copyAndSort);**  
 Returns the median of the specified sub-list (from `start` to `end` inclusive).

### 50.19.3 Standard Deviation

To calculate the standard deviation of an array of `Number` objects:

► **public static double getStdDev(Number[] data);**  
 Returns the standard deviation of an array of numbers.

- if `data` is `null` or zero-length, this method throws an `IllegalArgumentException`.

### 50.19.4 Other Methods

To calculate the correlation between two sets of values:

► **public static double getCorrelation(Number[] data1, Number[] data2);**  
 Returns the correlation between two sets of numbers.

- if `data1` or `data2` is `null`, this method throws an `IllegalArgumentException`.

To calculate a least squares regression line through an array of data:

```
► public static double[] getLinearFit(Number[] xData, Number[] yData);  
Returns the intercept (double[0]) and slope (double[1]) of the linear regression line for the  
supplied data points.
```

- if `xData` or `yData` is `null`, this method throws an `IllegalArgumentException`.

To calculate the slope of a least squares regression line:

```
► public static double getSlope(Number[] xData, Number[] yData);  
Returns the slope of the linear regression line.  
  
• if xData or yData is null, this method throws an IllegalArgumentException.  
  
► public static double[][] getMovingAverage(Number[] xData, Number[] yData, int period)  
Calculates moving average data based on the supplied x- and y-values.
```

### 50.19.5 Notes

This class was contributed by Matthew Wright.

# Chapter 51

## Package: org.jfree.data.time

### 51.1 Introduction

This package contains interfaces and classes that are used to represent *time-based* data.

The `TimeSeriesCollection` class is perhaps the most important class in this package. It is used to store one or more `TimeSeries` objects, and provides an implementation of the `XYDataset` interface. This allows it to be used as the dataset for an `XYPlot`).

The `TimePeriodValuesCollection` class performs a similar role, but allows more general (less regular) time periods to be used.

### 51.2 DateRange

#### 51.2.1 Overview

An extension of the `Range` class that is used to represent a date/time range. In JFreeChart, the primary use for this class is for specifying the range of values to display on a `DateAxis`.

#### 51.2.2 Constructors

To create a new date range:

```
➔ public DateRange(Date lower, Date upper);  
Creates a new date range using the specified lower and upper bounds (do not use null for either  
parameter).
```

#### 51.2.3 Notes

Instances of this class are immutable and `Serializable`.

### 51.3 Day

#### 51.3.1 Overview

A *regular time period* that is one day long. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

#### 51.3.2 Usage

A common use for this class is to represent daily data in a time series. For example:

```
TimeSeries series = new TimeSeries("Daily Data");
series.add(new Day(1, SerialDate.MARCH, 2003), 10.2);
series.add(new Day(3, SerialDate.MARCH, 2003), 17.3);
series.add(new Day(4, SerialDate.MARCH, 2003), 14.6);
series.add(new Day(7, SerialDate.MARCH, 2003), null);
```

Note that the `SerialDate` class is defined in the JCommon class library.

### 51.3.3 Constructor

There are several different ways to create a new `Day` instance. You can specify the day, month and year:

► `public Day(int day, int month, int year);`

Creates a new `Day` instance. The `month` argument should be in the range 1 to 12. The `year` argument should be in the range 1900 to 9999.

You can create a `Day` instance based on a `SerialDate` (defined in the JCommon class library):

► `public Day(SerialDate day);`

Creates a new `Day` instance.

You can create a `Day` instance based on a `Date`:

► `public Day(Date time);`

Creates a new `Day` instance.

Finally, the default constructor creates a `Day` instance based on the current system date:

► `public Day();`

Creates a new `Day` instance for the current system date.

### 51.3.4 Methods

There are methods to return the year, month and day-of-the-month:

► `public int getYear();`

Returns the year (in the range 1900 to 9999).

► `public int getMonth();`

Returns the month (in the range 1 to 12).

► `public int getDayOfMonth();`

Returns the day-of-the-month (in the range 1 to 31).

There is no method to *set* these attributes, because this class is immutable.

To return a `SerialDate` instance that represents the same day as this object:

► `public SerialDate getSerialDate();`

Returns the day as a `SerialDate`.

Given a `Day` object, you can create an instance representing the previous day or the next day:

► `public RegularTimePeriod previous();`

Returns the previous day, or `null` if the lower limit of the range is reached.

► `public RegularTimePeriod next();`

Returns the next day, or `null` if the upper limit of the range is reached.

To convert a `Day` object to a `String` object:

► `public String toString();`

Returns a string representing the day.

To convert a `String` object to a `Day` object:

► `public static Day parseDay(String s) throws TimePeriodFormatException;`

Parses the string and, if possible, returns a `Day` object.

### 51.3.5 Notes

Points to note:

- in the current implementation, the day can be in the range 1-Jan-1900 to 31-Dec-9999.
- the `Day` class is immutable, a requirement for all `RegularTimePeriod` subclasses.

## 51.4 DynamicTimeSeriesCollection

### 51.4.1 Overview

This class is a specialised form of time series dataset that is intended to be faster than the more general `TimeSeriesCollection` class. You can use this dataset when you have one or more series containing time series data, all with the same regular date values, and when you need to drop older data as newer data is added.

The underlying data structures used by this dataset are array-based, so updating the dataset is relatively fast.

### 51.4.2 Constructors

To create a new dataset:

```
→ public DynamicTimeSeriesCollection(int nSeries, int nMoments);
Creates a new dataset with the specified number of series. Each series will contain nMoments
observations. By default the x-values are measured using milliseconds.

→ public DynamicTimeSeriesCollection(int nSeries, int nMoments, TimeZone zone);
Creates a new dataset with the specified number of series. Each series will contain nMoments
observations, measured at regular millisecond intervals in the specified time zone.

→ public DynamicTimeSeriesCollection(int nSeries, int nMoments,
RegularTimePeriod timeSample);
Creates a new dataset with the specified number of series. Each series will contain nMoments
observations, measured at regular intervals of the specified time period.

→ public DynamicTimeSeriesCollection(int nSeries, int nMoments,
RegularTimePeriod timeSample, TimeZone zone);
Creates a new dataset with the specified number of series. Each series will contain nMoments
observations, measured at regular intervals of the specified time period.
```

After the dataset is created, call the `setTimeBase()` method to initialise the x-values for the dataset.<sup>1</sup>

### 51.4.3 Methods

To initialise the x-values for the dataset:

```
→ public synchronized long setTimeBase(RegularTimePeriod start);
Initialises the x-values (which are shared by all series in the dataset). The x-values are stored in
an array (the length was specified as nMoments in the constructor) beginning with the specified
start value, and incrementing the time period for each subsequent x-value.
```

The x-values are represented by time periods, but the dataset interface requires a single point in time to be returned as the x-value. These methods allow you to control whether the first, last or middle point in the time period is returned for the x-value:

```
→ public TimePeriodAnchor getXPosition();
Returns the position within each time period that is used as the x-value.
```

---

<sup>1</sup>It would probably make sense to refactor the class so that the x-values are initialised in the constructor.

```
→ public void setXPosition(TimePeriodAnchor position);
Sets the position within each time period that is used as the x-value.
```

To add a complete series to the dataset:

```
→ public void addSeries(float[] values, int seriesIndex,
String seriesName);
Adds/overwrites a set of y-values for the specified series. The x-values are as previously defined by the constructor and the setTimeBase() method.
```

To set the name for a series:

```
→ public void setSeriesName(int seriesIndex, String name);
Sets the name for a series.
```

To add a value to the dataset:

```
→ public void addValue(int seriesIndex, int index, float value);
Adds a value to the specified series.
```

To find out the number of series in the dataset:

```
→ public int getSeriesCount();
Returns the number of series in the dataset.
```

To find out the number of items within a series:

```
→ public int getItemCount(int series);
Returns the number of items in the specified series. For this dataset, all series have the same number of items (specified as nMoments in the constructor).
```

To “advance” the time:

```
→ public synchronized RegularTimePeriod advanceTime();
This method drops the oldest observation for all series and adds a new (zero) observation for the latest time period. Call this method before adding new data values.
```

Internally, the observations for all series are stored in a fixed-length array. To allow for older data to be “dropped” as newer data is added, two indices point to the oldest and newest items in the array:

```
→ public int getOldestIndex();
Returns the index of the oldest item.

→ public int getNewestIndex();
Returns the index of the newest item.
```

To get the oldest and newest time periods:

```
→ public RegularTimePeriod getOldestTime();
Returns the oldest time period.

→ public RegularTimePeriod getNewestTime();
Returns the newest time period.
```

To add a new value for each series:

```
→ public void appendData(float[] newData);
Updates the latest observation for each series in the dataset. This will overwrite the previous observation—you should call the advanceTime() method first if you want to drop an older observation to make room for a newer observation.
```

To add data at a particular index:

```
→ public void appendData(float[] newData, int insertionIndex, int refresh);
Adds one new item for each series in the dataset, and the specified index position.
```

#### 51.4.4 Notes

Some points to note:

- this dataset does not handle negative y-values (it could be implemented, but the original author of the class did not require it).

### 51.5 FixedMillisecond

#### 51.5.1 Overview

A *regular time period* that is one millisecond in length. This class uses the same encoding convention as `java.util.Date`. Unlike the other regular time period classes, `FixedMillisecond` is fixed in real time. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

#### 51.5.2 Constructors

To create a new `FixedMillisecond`:

```
➔ public FixedMillisecond(long millisecond);
Creates a new FixedMillisecond instance. The millisecond argument uses the same encoding as java.util.Date.
```

You can construct a `FixedMillisecond` instance based on a `java.util.Date` instance:

```
➔ public FixedMillisecond(Date time);
Creates a new FixedMillisecond instance representing the same millisecond as the time argument.
```

A default constructor is provided, which creates a `FixedMillisecond` instance based on the current system time:

```
➔ public FixedMillisecond();
Creates a new FixedMillisecond instance based on the current system time.
```

#### 51.5.3 Methods

Given a `FixedMillisecond` object, you can create an instance representing the previous millisecond:

```
➔ public RegularTimePeriod previous();
Returns the previous millisecond, or null if the lower limit of the range is reached.
```

...and the next millisecond:

```
➔ public RegularTimePeriod next();
Returns the next millisecond, or null if the upper limit of the range is reached.
```

#### 51.5.4 Notes

Some points to note:

- this class is just a wrapper for the `java.util.Date` class, to allow it to be used as a `RegularTimePeriod`;
- the `FixedMillisecond` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

### 51.6 Hour

#### 51.6.1 Overview

A *regular time period* one hour in length. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

## 51.6.2 Usage

A common use for this class is to represent hourly data in a time series. For example:

```
TimeSeries series = new TimeSeries("Hourly Data", Hour.class);
Day today = new Day();
series.add(new Hour(3, today), 734.4);
series.add(new Hour(4, today), 453.2);
series.add(new Hour(7, today), 500.2);
series.add(new Hour(8, today), null);
series.add(new Hour(12, today), 734.4);
```

Note that the hours in the `TimeSeries` do not have to be consecutive.

## 51.6.3 Constructor

There are several ways to create a new `Hour` instance. You can specify the hour and day:

► `public Hour(int hour, Day day);`  
Creates a new `Hour` instance. The `hour` argument should be in the range 0 to 23.

Alternatively, you can supply a `java.util.Date`:

► `public Hour(Date time);`  
Creates a new `Hour` instance. The default time zone is used to decode the `Date`.

A default constructor is provided:

► `public Hour();`  
Creates a new `Hour` instance based on the current system time.

## 51.6.4 Methods

To access the hour and day:

► `public int getHour();`  
Returns the hour (in the range 0 to 23).

► `public Day getDay();`  
Returns the day.

There is no method to *set* the hour or the day, because this class is immutable.

Given a `Hour` object, you can create an instance representing the previous hour:

► `public RegularTimePeriod previous();`  
Returns the previous hour, or `null` if the lower limit of the range is reached.

...or the next hour:

► `public RegularTimePeriod next();`  
Returns the next hour, or `null` if the upper limit of the range is reached.

## 51.6.5 Notes

The `Hour` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

## 51.7 Millisecond

### 51.7.1 Overview

A *regular time period* one millisecond in length. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations. Extends `RegularTimePeriod`.

### 51.7.2 Constructors

To construct a `Millisecond` instance:

```
↳ public Millisecond(int millisecond, Second second);
```

Creates a new `Millisecond` instance. The `millisecond` argument should be in the range 0 to 999.

To construct a `Millisecond` instance based on a `java.util.Date`:

```
↳ public Millisecond(Date date);
```

Creates a new `Millisecond` instance.

A default constructor is provided:

```
↳ public Millisecond();
```

Creates a new `Millisecond` instance based on the current system time.

### 51.7.3 Methods

To access the millisecond:

```
↳ public int getMillisecond();
```

Returns the second (in the range 0 to 999).

To access the `Second`:

```
↳ public Second getSecond();
```

Returns the `Second`.

There is no method to *set* the millisecond or the second, because this class is immutable.

Given a `Millisecond` object, you can create an instance representing the previous millisecond:

```
↳ public RegularTimePeriod previous();
```

Returns the previous millisecond, or `null` if the lower limit of the range is reached.

...or the next:

```
↳ public RegularTimePeriod next();
```

Returns the next millisecond, or `null` if the upper limit of the range is reached.

### 51.7.4 Notes

The `Millisecond` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

## 51.8 Minute

### 51.8.1 Overview

A *regular time period* one minute in length. This class is designed to be used with the `TimeSeries` class, but could also be used in other situations.

### 51.8.2 Constructors

There are several ways to create new instances of this class. You can specify the minute and hour:

```
↳ public Minute(int minute, Hour hour);
```

Creates a new `Minute` instance. The `minute` argument should be in the range 0 to 59.

Alternatively, you can supply a `java.util.Date`:

```
↳ public Minute(Date time);
```

Creates a new `Minute` instance based on the supplied date/time.

A default constructor is provided:

```
↳ public Minute();
```

Creates a new `Minute` instance, based on the current system time.

### 51.8.3 Methods

To access the minute and hour:

- `public int getMinute();`  
Returns the minute (in the range 0 to 59).
- `public Hour getHour();`  
Returns the hour.

There is no method to *set* the minute or the day, because this class is immutable.

Given a `Minute` object, you can create an instance representing the previous minute:

- `public RegularTimePeriod previous();`  
Returns the previous minute, or `null` if the lower limit of the range is reached.

...or the next:

- `public RegularTimePeriod next();`  
Returns the next minute, or `null` if the upper limit of the range is reached.

### 51.8.4 Notes

The `Minute` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

## 51.9 Month

### 51.9.1 Overview

A *time period* representing a month in a particular year. This class is designed to be used with the `TimeSeries` class, but could be used in other contexts as well. Extends `RegularTimePeriod`.

### 51.9.2 Constructors

There are several ways to create new instances of this class. You can specify the month and year:

- `public Month(int month, Year year);`  
Creates a new `Month` instance. The `month` argument should be in the range 1 to 12.
- `public Month(int month, int year);`  
Creates a new `Month` instance. The `month` argument should be in the range 1 to 12. The `year` argument should be in the range 1900 to 9999.

Alternatively, you can specify a `java.util.Date`:

- `public Month(Date time);`  
Creates a new `Month` instance.

A default constructor is provided:

- `public Month();`  
Creates a new `Month` instance, based on the current system time.

### 51.9.3 Methods

To access the month and year:

- `public int getMonth();`  
Returns the month (in the range 1 to 12).
- `public Year getYear();`  
Returns the year.
- `public int getYearValue();`  
Returns the year as an `int`.

There is no method to *set* the month or the year, because this class is immutable.

Given a `Month` object, you can create an instance representing the previous month:

► public `RegularTimePeriod` `previous()`;

Returns the previous month, or `null` if the lower limit of the range is reached.

...or the next month:

► public `RegularTimePeriod` `next()`;

Returns the next month, or `null` if the upper limit of the range is reached.

To convert a `Month` object to a `String` object:

► public `String` `toString()`;

Returns a string representing the month.

#### 51.9.4 Notes

Points to note:

- the year can be in the range 1900 to 9999.
- this class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

### 51.10 MovingAverage

#### 51.10.1 Overview

A utility class for calculating a *moving average* for a data series (usually a `TimeSeries`). Moving averages are most commonly used in the analysis of stock prices or other financial data.

#### 51.10.2 An Example

An example is perhaps the best way to illustrate how moving averages are calculated. A sample dataset containing daily data and a corresponding three-day moving average is presented in Table 51.1.

Date:	Value:	3 Day Moving Average:
11-Aug-2003	11.2	-
13-Aug-2003	13.8	-
17-Aug-2003	14.1	14.100
18-Aug-2003	12.7	13.400
19-Aug-2003	16.5	14.433
20-Aug-2003	15.6	14.933
25-Aug-2003	19.8	19.800
27-Aug-2003	10.7	15.250
28-Aug-2003	14.3	12.500

Table 51.1: A sample moving average

The code to calculate this moving average is:

```
TimeSeries series = new TimeSeries("Series 1", Day.class);
series.add(new Day(11, SerialDate.AUGUST, 2003), 11.2);
series.add(new Day(13, SerialDate.AUGUST, 2003), 13.8);
series.add(new Day(17, SerialDate.AUGUST, 2003), 14.1);
series.add(new Day(18, SerialDate.AUGUST, 2003), 12.7);
series.add(new Day(19, SerialDate.AUGUST, 2003), 16.5);
series.add(new Day(20, SerialDate.AUGUST, 2003), 15.6);
series.add(new Day(25, SerialDate.AUGUST, 2003), 19.8);
series.add(new Day(27, SerialDate.AUGUST, 2003), 10.7);
series.add(new Day(28, SerialDate.AUGUST, 2003), 14.3);
```

```
TimeSeries mavg = MovingAverage.createMovingAverage(
    source, "Moving Average", 3, 3
);
```

In this example, we have chosen to skip the average calculation for the first three days (11, 12 and 13 August) of the time series (note that there are only two observations in this three day period for the example series). For each of the other dates, an average value is calculated by taking the three days up to and including the particular date. For example, for 19 August, the values for 17, 18 and 19 August are averaged to give a value of 14.433:

$$[14.1 + 12.7 + 16.5] / 3 = 43.3 / 3 = 14.433$$

Similarly, the value for 25 August is the average of the values for 23, 24 and 25 August—but in this case no values are available for 23 or 24 August, so only the value from 25 August is used.

### 51.10.3 Methods

To calculate a moving average for a time series:

```
► public static TimeSeries createMovingAverage(TimeSeries source, String name, int periodCount,
    int skip);
```

Creates a new series containing moving average values based on the `source` series. The new series will be called `name`. The `periodCount` specifies the number of periods over which the average is calculated, and `skip` controls the initial number of periods for which no average is calculated (usually 0 or `periodCount - 1`).

To calculate a moving average for each time series in a collection:

```
► public static TimeSeriesCollection createMovingAverage(
    TimeSeriesCollection source, String suffix, int periodCount, int skip)
```

Returns a new collection containing a moving average time series for each series in the source collection. The names of the moving average series are derived by appending the specified suffix to the source series name.

An alternative means of calculating a moving average is to count back a fixed number of points, irrespective of the “age” of each point:

```
► public static TimeSeries createPointMovingAverage(TimeSeries source, String name, int pointCount)
Creates a new series containing moving average values based on the source series.
```

### 51.10.4 Notes

The `MovingAverageDemo1` class in the JFreeChart demo collection provides one example of how to use this class.

## 51.11 Quarter

### 51.11.1 Overview

A calendar quarter—this class extends `RegularTimePeriod`.

### 51.11.2 Usage

A common use for this class is representing quarterly data in a time series:

```
TimeSeries series = new TimeSeries("Quarterly Data", Quarter.class);
series.add(new Quarter(1, 2001), 500.2);
series.add(new Quarter(2, 2001), 694.1);
series.add(new Quarter(3, 2001), 734.4);
series.add(new Quarter(4, 2001), 453.2);
series.add(new Quarter(1, 2002), 500.2);
series.add(new Quarter(2, 2002), null);
series.add(new Quarter(3, 2002), 734.4);
series.add(new Quarter(4, 2002), 453.2);
```

### 51.11.3 Constructor

There are several ways to create a new `Quarter` instance. You can specify the quarter and year:

- ↳ `public Quarter(int quarter, Year year);`  
Creates a new `Quarter` instance. The `quarter` argument should be in the range 1 to 4.
- ↳ `public Quarter(int quarter, int year);`  
Creates a new `Quarter` instance.

Alternatively, you can supply a `java.util.Date`:

- ↳ `public Quarter(Date time);`  
Creates a new `Quarter` instance.

A default constructor is provided:

- ↳ `public Quarter();`  
Creates a new `Quarter` instance based on the current system time.

### 51.11.4 Methods

To access the quarter and year:

- ↳ `public int getQuarter();`  
Returns the quarter (in the range 1 to 4).
- ↳ `public Year getYear();`  
Returns the year.

There is no method to *set* the quarter or the year, because this class is immutable.

Given a `Quarter` object, you can create an instance representing the previous or next quarter:

- ↳ `public RegularTimePeriod previous();`  
Returns the previous quarter, or `null` if the lower limit of the range is reached.
- ↳ `public RegularTimePeriod next();`  
Returns the next quarter, or `null` if the upper limit of the range is reached.

To convert a `Quarter` object to a `String` object:

- ↳ `public String toString();`  
Returns a string representing the quarter.

### 51.11.5 Notes

Points to note:

- the year can be in the range 1900 to 9999.
- this class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

## 51.12 RegularTimePeriod

### 51.12.1 Overview

An abstract class that represents a *time period* that occurs at some regular interval. A number of concrete subclasses have been implemented: `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`, `Millisecond` and `FixedMillisecond`.

### 51.12.2 Time Zones

The time periods represented by this class and its subclasses typically “float” with respect to any specific time zone. For example, if you define a `Day` object to represent 1-Apr-2002, then that is the day it represents *no matter where you are in the world*. Of course, against a real time line, 1-Apr-2002 in (say) New Zealand is not the same as 1-Apr-2002 in (say) France. But *sometimes* you want to treat them as if they were the same, and that is what this class does.<sup>2</sup>

### 51.12.3 Conversion To/From Date Objects

Occasionally you may want to convert a `RegularTimePeriod` object into an instance of `java.util.Date`. The latter class represents a precise moment in real time (as the number of milliseconds since January 1, 1970, 00:00:00.000 GMT), so to do the conversion you have to “peg” the `RegularTimePeriod` instance to a particular time zone.

The `getStart()` and `getEnd()` methods provide this facility, using the default timezone. In addition, there are other methods to return the first, last and middle milliseconds for the time period, using the default time zone, a user supplied timezone, or a `Calendar` with the timezone preset.

### 51.12.4 Methods

Given a `RegularTimePeriod` instance, you can create another instance representing the previous or next time period:

- `public abstract RegularTimePeriod previous();`  
Returns the previous time period, or `null` if the current time period is the first in the supported range.
- `public abstract RegularTimePeriod next();`  
Returns the next time period, or `null` if the current time period is the last in the supported range.

To assist in converting the time period to a `java.util.Date` object, the following methods peg the time period to a particular time zone and return the first and last millisecond of the time period (using the same encoding convention as `java.util.Date`):

- `public long getFirstMillisecond();`  
Returns the first millisecond of the time period, evaluated using the default timezone.
- `public long getFirstMillisecond(TimeZone zone);`  
Returns the first millisecond of the time period, evaluated using a particular timezone.
- `public abstract long getFirstMillisecond(Calendar calendar);`  
Returns the first millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).
- `public long getMiddleMillisecond();`  
Returns the middle millisecond of the time period, evaluated using the default timezone.
- `public long getMiddleMillisecond(TimeZone zone);`  
Returns the middle millisecond of the time period, evaluated using a particular timezone.
- `public long getMiddleMillisecond(Calendar calendar);`  
Returns the middle millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).
- `public long getLastMillisecond();`  
The last millisecond of the time period, evaluated using the default timezone.
- `public long getLastMillisecond(TimeZone zone);`  
Returns the last millisecond of the time period, evaluated using a particular timezone.

---

<sup>2</sup>For example, an accountant might be adding up sales for all the subsidiaries of a multinational company. Sales on 1-Apr-2002 in New Zealand are added to sales on 1-Apr-2002 in France, even though the real time periods are offset from one another.

```
➔ public abstract long getLastMillisecond(Calendar calendar);
Returns the last millisecond of the time period, evaluated using the supplied calendar (which
incorporates a timezone).
```

### 51.12.5 Notes

Points to note:

- this class and its subclasses can be used with the [TimeSeries](#) class.
- all [RegularTimePeriod](#) subclasses are required to be immutable.
- known subclasses include: [Year](#), [Quarter](#), [Month](#), [Week](#), [Day](#), [Hour](#), [Minute](#), [Second](#), [Millisecond](#) and [FixedMillisecond](#).

## 51.13 Second

### 51.13.1 Overview

A *regular time period* that is one second long. This class is designed to be used with the [TimeSeries](#) class, but could also be used in other situations. Extends [RegularTimePeriod](#).

### 51.13.2 Constructors

There are several ways to create new instances of this class. You can specify the minute and second:

```
➔ public Second(int second, Minute minute);
Creates a new Second instance. The second argument should be in the range 0 to 59.
```

Alternatively, you can supply a [java.util.Date](#):

```
➔ public Second(Date date);
Creates a new Second instance.
```

A default constructor is provided:

```
➔ public Second();
Creates a new Second instance based on the current system time.
```

### 51.13.3 Methods

To access the second and minute:

```
➔ public int getSecond();
Returns the second (in the range 0 to 59).

➔ public Minute getMinute();
Returns the minute.
```

There is no method to *set* the second or the minute, because this class is immutable.

Given a [Second](#) object, you can create an instance representing the previous second or the next second:

```
➔ public RegularTimePeriod previous();
Returns the previous second, or null if the lower limit of the range is reached.

➔ public TimePeriod next();
Returns the next second, or null if the upper limit of the range is reached.
```

### 51.13.4 Notes

The [Second](#) class is immutable. This is a requirement for all [RegularTimePeriod](#) subclasses.

## 51.14 SimpleTimePeriod

### 51.14.1 Overview

This class represents a fixed period of time with millisecond precision (implements the `TimePeriod` interface).

### 51.14.2 Constructor

To create a new instance:

```
➔ public SimpleTimePeriod(Date start, Date end);  
Creates a new time period with the specified start and end.
```

### 51.14.3 Methods

To return the start and end dates:

```
➔ public Date getStart();  
Returns the start date (or time) for the period.  
  
➔ public Date getEnd();  
Returns the end date (or time) for the period.
```

To test for equality with an arbitrary object:

```
➔ public boolean equals(Object obj);  
Tests whether this time period is equal to an arbitrary object. This method will return true if  
obj is an instance of TimePeriod that has the same start and end date/time values.
```

### 51.14.4 Notes

Some points to note:

- instances of this class are immutable;
- implements the `Serializable` interface;

## 51.15 TimePeriod

### 51.15.1 Overview

A period of time defined by two `java.util.Date` instances representing the start and end of the time period. This interface is implemented by:

- the `SimpleTimePeriod` class;
- the `RegularTimePeriod` base class and all its subclasses.

### 51.15.2 Methods

To get the start and end of the time period:

```
➔ public Date getStart();  
Returns the start of the time period.  
  
➔ public Date getEnd();  
Returns the end of the time period.
```

### 51.15.3 Notes

If you write your own class that implements this interface, you are advised to:

- ensure that instances of your class are immutable—this means that dataset classes that use your class can only be modified or updated in ways that they are aware of (so that the appropriate change events are generated);
- ensure that instances of your class are serializable—otherwise your datasets might not be serializable.

#### See Also:

[TimePeriodValue](#).

## 51.16 TimePeriodAnchor

### 51.16.1 Overview

An enumeration of the three possible *time period anchor positions*:

- START - the start of the time period;
- MIDDLE - the middle of the time period;
- END - the end of the time period.

These are used by the [TimeSeriesCollection](#) and [TimePeriodValuesCollection](#) classes to determine how x-values are derived from the underlying time periods when these classes are used as [XYDataset](#) instances.

## 51.17 TimePeriodFormatException

### 51.17.1 Overview

An exception that can be thrown by the methods used to convert time periods to strings, and vice versa.

## 51.18 TimePeriodValue

### 51.18.1 Overview

An object that represents a time period with an associated value, used to represent each item in a [TimePeriodValues](#) collection.

### 51.18.2 Constructors

To create a new [TimePeriodValue](#) object:

```
➔ public TimePeriodValue(TimePeriod period, Number value);
Creates a new data item that associates a value (null permitted) with a period. The period argument should not be null.
```

For convenience, you can also use the following constructor:

```
➔ public TimePeriodValue(TimePeriod period, double value);
Creates a new data item that associates a value with a period. The period argument should not be null.
```

### 51.18.3 Methods

There are methods for accessing the `period` and `value` attributes. You can update the value but not the period (this allows other classes to maintain a collection of `TimePeriodValue` objects in some order that is based on the `period`, without the risk of that order being compromised by a change to a particular item):

- `public TimePeriod getPeriod();`  
Returns the time period for this item, which should not be `null`.
- `public Number getValue();`  
Returns the value for this item. This may be `null`.
- `public void setValue(Number value);`  
Sets the value for this item (`null` is permitted, and represents an unknown value).

### 51.18.4 Equals, Cloning and Serialization

To test this object for equality with an arbitrary object:

- `public boolean equals(Object obj);`  
Returns `true` if this instance is equal to `obj`, and `false` otherwise.

Instances of this class are cloneable, on the assumption that the `TimePeriod` instance is immutable. Similarly, instances of this class are serializable, provided that the `TimePeriod` instance is serializable.

## 51.19 TimePeriodValues

### 51.19.1 Overview

A structure containing zero, one or many `TimePeriodValue` objects. The time periods are arbitrary, can overlap, and are maintained in the order they are added to this instance. This class is used to represent one data series in a `TimePeriodValuesCollection`.

This class is similar to the `TimeSeries` class, but it allows arbitrary time periods to be defined, rather than the more regular time periods required by the `TimeSeries` class. Use this class when you need less structure and more flexibility.

### 51.19.2 Constructors

To create a new instance:

- `public TimePeriodValues(String name);`  
Creates a new instance, initially empty, with the specified name.
- `public TimePeriodValues(String name, String domain, String range);`  
Creates a new instance, initially empty, with the specified name, domain description and range description.

### 51.19.3 Domain and Range Descriptions

This class allows text descriptions for the domain and range to be specified. These are not used anywhere within JFreeChart, but are available for your application to use:

- `public String getDomainDescription();`  
Returns a description for the domain values (that is, the time periods). This method can return `null`. The default value is “Time”.
- `public void setDomainDescription(String description);`  
Sets the description for the domain values (that is, the time periods). The description may be `null`. This method fires a `PropertyChangeEvent` with the property name `Domain`.

► `public String getRangeDescription();`  
 Returns a description for the range values (that is, the y-values). This method can return `null`.  
 The default value is “Value”.

► `public void setRangeDescription(String description);`  
 Sets the description for the range values. This method fires a `PropertyChangeEvent` with the  
 property name `Range`.

#### 51.19.4 Data Access Methods

The following methods can be used to access the data items defined in this structure:

► `public int getItemCount();`  
 Returns the number of data items in the series.

► `public TimePeriodValue getDataItem(int index);`  
 Returns the data item at the specified index. If `index` is not in the range 0 to `getItemCount()` - 1, this method throws an exception.

► `public TimePeriod getTimePeriod(int index);`  
 Returns the time period at the specified index. If `index` is not in the range 0 to `getItemCount()` - 1, this method throws an exception. Note that if you want both the time period and the associated value, you should call `getDataItem(int)`.

► `public Number getValue(int index);`  
 Returns the value at the specified index. If `index` is not in the range 0 to `getItemCount()` - 1, this method throws an exception.

► `public void add(TimePeriodValue item);`  
 Adds an item to the end of this series and sends a `SeriesChangeEvent` to all registered listeners. If `item` is `null`, this method throws an `IllegalArgumentException`.

► `public void add(TimePeriod period, double value);`  
 Adds an observation as the last item in the series and sends a `SeriesChangeEvent` to all registered listeners. If `period` is `null`, this method throws an `IllegalArgumentException`.

► `public void add(TimePeriod period, Number value);`  
 Adds an observation as the last item in the series and sends a `SeriesChangeEvent` to all registered listeners. If `period` is `null`, this method throws an `IllegalArgumentException`.

► `public void update(int index, Number value);`  
 Updates the item with the specified index—the time period is not changed, but the associated value is replaced by the specified value (which may be `null`).

► `public void delete(int start, int end);`  
 Deletes all the items between the specified indices.

► `public TimePeriodValues createCopy(int start, int end) throws CloneNotSupportedException;`  
 Returns an independent copy of the items between the specified indices.

#### 51.19.5 Equals, Cloning and Serialization

This class overrides the equals method:

► `public boolean equals(Object obj);`  
 Tests this instance for equality with an arbitrary object.

To support the general cloning and serialization of charts and their datasets, this class is both cloneable and serializable.

### 51.19.6 Other Methods

The following methods provide information about the items in the dataset with the minimum and maximum x-values. It is something of an implementation detail that these are public—you most likely won't need to use these methods:

- `public int getMinStartIndex();`  
Returns the index of the item with the time period that starts earliest in time.
- `public int getMaxStartIndex();`  
Returns the index of the item with the time period that starts latest in time.
- `public int getMinMiddleIndex();`  
Returns the index of the item with the time period with the earliest mid-point.
- `public int getMaxMiddleIndex();`  
Returns the index of the item with the time period with the latest mid-point.
- `public int getMinEndIndex();`  
Returns the index of the item with the time period that finishes earliest in time.
- `public int getMaxEndIndex();`  
Returns the index of the item with the time period that finishes latest in time.

## 51.20 TimePeriodValuesCollection

### 51.20.1 Overview

A collection of `TimePeriodValues` objects—this class implements the `XYDataset` interface (and, furthermore, the `IntervalXYDataset` interface). This class provides a more flexible (but less structured) alternative to the `TimeSeriesCollection` class. This class extends `AbstractIntervalXYDataset`.

### 51.20.2 Usage

The `TimePeriodValuesDemo1` application, included in the JFreeChart demo collection, provides an example of how to use this class.

### 51.20.3 Constructors

To create a new, empty collection:

- `public TimePeriodValuesCollection();`  
Creates a new empty collection. After creation, you can add `TimePeriodValues` objects using the `addSeries()` method.
- `public TimePeriodValuesCollection(TimePeriodValues series);`  
Creates a new collection containing a single series (you can add more series later, if you need to). If `series` is `null`, it is ignored.

### 51.20.4 General Methods

To find the number of series in the dataset:

- `public int getSeriesCount();`  
Returns the number of series in the collection.

To obtain the key for a series:

- `public Comparable getSeriesKey(int series);`  
Returns a series key. Calling the `toString()` method on this key should give a human-readable name for the series (it is used, by default, in the chart's legend).

To obtain the data for one series:

```
→ public TimePeriodValues getSeries(int series);
Returns a series containing zero, one or many TimePeriodValue instances.
```

To add a series to the dataset:

```
→ public void addSeries(TimePeriodValues series);
Adds a series to the dataset, and sends a DatasetChangeEvent to all registered listeners. If series is null, this method throws an IllegalArgumentException.
```

To remove a series from the dataset:

```
→ public void removeSeries(TimePeriodValues series);
Removes the series from the dataset and sends a DatasetChangeEvent to all registered listeners. If series is null, this method throws an IllegalArgumentException.

→ public void removeSeries(int index);
Removes a series from the dataset and sends a DatasetChangeEvent to all registered listeners.

→ public int getItemCount(int series);
Returns the number of data items in the specified series.
```

### 51.20.5 Alignment of X-Values

Some renderers (for example, the XYLineAndShapeRenderer) will require a single x-value to plot each data item, even though the data value applies to a range of x-values defined by a TimePeriod. To resolve these cases, the dataset defines an anchor that determines which point within a time period is used to represent the x-value for that data item:

```
→ public TimePeriodAnchor getXPosition();
Returns the x-value anchor point for this dataset. This method never returns null. The default value is TimePeriodAnchor.MIDDLE.

→ public void setXPosition(TimePeriodAnchor position);
Sets the x-value anchor point for this dataset. If position is null, this method throws an IllegalArgumentException.
```

### 51.20.6 Other Methods

To get the x-values from the dataset:

```
→ public Number getX(int series, int item);
Returns the x-value for the specified item within the given series.

→ public Number getStartX(int series, int item);
Returns the start x-value for the specified item within the given series.

→ public Number getEndX(int series, int item);
Returns the end x-value for the specified item within the given series.

→ public Number getY(int series, int item);
Returns the y-value for the specified item within the given series.

→ public Number getStartY(int series, int item);
Returns the start y-value for the specified item within the given series.

→ public Number getEndY(int series, int item);
Returns the end y-value for the specified item within the given series.
```

### 51.20.7 DomainInfo Methods

This class implements the DomainInfo interface, with the following methods:

```
→ public double getDomainLowerBound(boolean includeInterval);
Returns the lower bound of the range of x-values in the dataset.

→ public double getDomainUpperBound(boolean includeInterval);
Returns the upper bound of the range of x-values in the dataset.

→ public Range getDomainBounds(boolean includeInterval);
Returns the range of x-values in the dataset.
```

### 51.20.8 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
    Tests this dataset for equality with an arbitrary object.
```

### 51.20.9 Notes

Some points to note:

- this class implements the `DomainInfo` interface;
- a demo (`TimePeriodValuesDemo1.java`) is included in the JFreeChart demo collection.

#### See Also:

[TimePeriodValues](#).

## 51.21 TimeSeries

### 51.21.1 Overview

A time series is a data structure that associates numeric values with particular time periods. In other words, a collection of data values in the form  $(timeperiod, value)$ . The time periods are represented by subclasses of `RegularTimePeriod`, including `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second`, `Millisecond` and `FixedMillisecond`. The values are represented by the `Number` class. The value `null` can be used to indicate missing or unknown values.

### 51.21.2 Usage

A time series may contain zero, one or many time periods with associated data values. You can assign a `null` value to a time period, and you can skip time periods completely. You cannot add duplicate time periods to a time series. Different subclasses of `RegularTimePeriod` cannot be mixed within one time series.

Here is an example showing how to create a series with quarterly data:

```
TimeSeries series = new TimeSeries("Quarterly Data", Quarter.class);
series.add(new Quarter(1, 2001), 500.2);
series.add(new Quarter(2, 2001), 694.1);
series.add(new Quarter(3, 2001), 734.4);
series.add(new Quarter(4, 2001), 453.2);
series.add(new Quarter(1, 2002), 500.2);
series.add(new Quarter(2, 2002), null);
series.add(new Quarter(3, 2002), 734.4);
series.add(new Quarter(4, 2002), 453.2);
```

One or more `TimeSeries` objects can be aggregated to form a dataset for a chart using the `TimeSeriesCollection` class.

A demo application (`TimeSeriesDemo1.java`) is included in the JFreeChart demo collection.

### 51.21.3 Constructors

To create a named time series containing no data:

```
➔ public TimeSeries(String name);
Creates an empty time series for daily data (that is, one value per day). The supplied name is
used as the series key.
```

To create a time series for a frequency other than daily, use this constructor:

```
→ public TimeSeries(String name, Class timePeriodClass);
```

Creates an empty time series. The caller specifies the time period by specifying the class of the [RegularTimePeriod](#) subclass (for example, `Month.class`).

The final constructor allows you to specify descriptions for the domain and range of the data:

```
→ public TimeSeries(String name, String domain, String range, Class timePeriodClass);
```

Creates an empty time series. The caller specifies the time period, plus strings describing the domain and range.

#### 51.21.4 Attributes

Each instance of `TimeSeries` has the following attributes:

Attribute:	Description:
<code>key</code>	The series key (inherited from <a href="#">Series</a> ). The <code>toString()</code> form of the key is often displayed in the legend for a chart.
<code>domainDescription</code>	A description of the time period domain (for example, “Quarter”). The default is “Time”.
<code>rangeDescription</code>	A description of the value range (for example, “Price”). The default is “Value”.
<code>maximumItemCount</code>	The maximum number of items that the series will record. Once this limit is reached, the oldest observation is dropped whenever a new observation is added.
<code>historyCount</code>	The number of time periods defining a “window” for the data. Starting with the latest observation, the window extends back for this number of time periods. Any data older than the window is discarded.

#### 51.21.5 General Methods

To find the class of time period recorded by this series:

```
→ public Class getTimePeriodClass();
```

Returns the class of [RegularTimePeriod](#) stored by this time series. This is specified in the constructor, and is used to ensure consistency among the items stored in the series.

To access the domain and range descriptions:

```
→ public String getDomainDescription();
```

Returns a general description for the domain values (that is, the time periods). This may be `null`.

```
→ public void setDomainDescription(String description);
```

Sets the general description for the domain values and sends a [PropertyChangeEvent](#) (with the property name `Domain`) to all registered listeners.

```
→ public String getRangeDescription();
```

Returns a general description for the range values (that is, the y-values). This may be `null`.

```
→ public void setRangeDescription(String description);
```

Sets the general description for the range values and sends a [PropertyChangeEvent](#) (with the property name `Range`) to all registered listeners.

#### 51.21.6 Data Retrieval Methods

To find out how many data items are in a series:

```
→ public int getItemCount()
```

Returns the number of data items in the series (possibly zero).

To get a list of the items in the dataset:

```
→ public List getItems();
```

Returns an unmodifiable list of the items in the dataset—the items in the list are instances of `TimeSeriesDataItem`. *NOTE: in fact, the items themselves can be modified, so this method is poorly implemented. If you do modify any item directly, no event notification will happen, and other objects that monitor changes to the time series may end up in a bad state.*

► public Number getValue(int index);  
 Returns the value (possibly `null`) at the specified index in the series. This method throws an `IndexOutOfBoundsException` if `index` is not in the range 0 to `getItemCount()` - 1.

► public Number getValue(RegularTimePeriod period);  
 Returns the value (which may be `null`) for the specified period, or `null` if there is no such period.  
 Note: to determine whether or not `period` is defined for the series, use the `getIndex(RegularTimePeriod)` method.

To retrieve an item from the time series:

► public TimeSeriesDataItem getDataItem(RegularTimePeriod period)  
 Returns the data item for the specified time period, or `null` if there is no such item. This method throws an `IllegalArgumentException` if `period` is `null`.

► public TimeSeriesDataItem getDataItem(int index)  
 Returns the data item with the specified `index` (zero-based). This method throws an `IndexOutOfBoundsException` if `index` is not in the range 0 to `getItemCount()` - 1.

► public RegularTimePeriod getTimePeriod(int index);  
 Returns the time period at the specified index in the series. This method throws an `IndexOutOfBoundsException` if `index` is not in the range 0 to `getItemCount()` - 1.

► public int getIndex(RegularTimePeriod period);  
 Returns the index of the item in the series that corresponds to the specified `period`. If no item with the specified period is found in the series, this method returns a negative index (-`n`-1 where `n` is the index of the position that the specified period would occupy). This method throws an `IllegalArgumentException` if `period` is `null`.

### 51.21.7 Adding/Updating Items

There are a range of methods for adding/updating items in the time series. Each modification to the series will typically send a `SeriesChangeEvent` to all registered listeners, except when:

- an add or update method with a `notify` parameter is used, and `false` is passed in;
- a previous call to `setNotify` has set the `notify` flag to `null`.

To add a value to a time series:

► public void add(RegularTimePeriod period, double value);  
 Adds a new item to the time series and sends a `SeriesChangeEvent` to all registered listeners (unless event notification has been disabled).

► public void add(RegularTimePeriod period, double value, boolean notify);  
 Adds a new item to the time series and, if requested, sends a `SeriesChangeEvent` to all registered listeners.

► public void add(RegularTimePeriod period, Number value);  
 Adds a new value (`null` permitted) to the time series and sends a `SeriesChangeEvent` to all registered listeners. This method throws a `SeriesException` if the time period is not unique within the series.

► public void add(RegularTimePeriod period, Number value, boolean notify);  
 Adds a new value (`null` permitted) to the time series and, if requested, sends a `SeriesChangeEvent` to all registered listeners. This method throws a `SeriesException` if the time period is not unique within the series.

► public void add(TimeSeriesDataItem item);  
 Adds the specified `item` to the time series and sends a `SeriesChangeEvent` to all registered listeners.

► public void add(TimeSeriesDataItem item, boolean notify);  
 Adds the specified `item` to the time series and, if requested, sends a `SeriesChangeEvent` to all registered listeners.

To modify the value of an existing item:

- `public void update(RegularTimePeriod period, Number value);`  
Modifies the value of an existing item in the time series and sends a `SeriesChangeEvent` to all registered listeners.
- `public void update(int index, Number value);`  
Modifies the value of an existing item in the time series and sends a `SeriesChangeEvent` to all registered listeners.

Often you want to set the value for a time period without being concerned about whether or not there is an existing value in the time series—the following methods can be used for that case:

- `public TimeSeriesDataItem addOrUpdate(RegularTimePeriod period, double value);`  
Adds a new item or updates an existing item, as appropriate, and sends a `SeriesChangeEvent` to all registered listeners.
- `public TimeSeriesDataItem addOrUpdate(RegularTimePeriod period, Number value);`  
Adds a new item or updates an existing item, as appropriate, and sends a `SeriesChangeEvent` to all registered listeners.

The following method allows you to add all the values from one series to this series:

- `public TimeSeries addAndOrUpdate(TimeSeries series);`  
Adds all the values from `series` to this time series and sends a `SeriesChangeEvent` to all registered listeners. This method returns a new series containing just the overwritten items from this series (useful for implementing undo actions).

### 51.21.8 Removing Items

To remove an item from a series:

- `public void delete(RegularTimePeriod period);`  
Deletes the item with the specified `period` and sends a `SeriesChangeEvent` to all registered listeners. If there is no item for the `period`, this method does nothing. If `period` is `null`, this method throws an `IllegalArgumentException`.

To remove a range of items from a series:

- `public void delete(int start, int end);`  
Deletes the data items in the specified index range (inclusive) and then sends a `SeriesChangeEvent` to all registered listeners. This method will throw an `IllegalArgumentException` if `end` is less than `start`.

To remove all items from a series:

- `public void clear();`  
Clears all items from the series and sends a `SeriesChangeEvent` to all registered listeners. If the series doesn't contain any items to begin with, this method does nothing.

### 51.21.9 Copying Subsets

To copy a subset of the data items from this series into a new series:

- `public TimeSeries createCopy(int start, int end) throws CloneNotSupportedException;`  
Creates a new time series that is a copy of this time series, but containing only the items in the specified index range. This method throws an `IllegalArgumentException` if `start < 0` or `end < start`. If this time series is empty, the `start` and `end` arguments are ignored, and this method returns a clone of this (empty) series.
- `public TimeSeries createCopy(RegularTimePeriod start, RegularTimePeriod end) throws CloneNotSupportedException;`  
Creates a new time series that is a copy of this time series, but containing only the items from `start` to `end` inclusive. This method throws an `IllegalArgumentException` if either argument is `null`, or if `start` is after `end`.

### 51.21.10 Discarding Old Data Items

You can create a time series that automatically discards “old” data items whenever new items are added. There are two attributes, `maximumItemCount` and `maximumItemAge`, that can be used to achieve this (in slightly different ways).

The `maximumItemCount` attribute fixes the *maximum number* of items that the series can hold—if a new item is added such that the series contains more than `maximumItemCount` items, the oldest item in the series is discarded (permanently):

► public int `getMaximumItemCount()`;

Returns the maximum number of items that can be held in the series. If a new item is added to the series such that the maximum item count will be exceeded, then the oldest item in the series is discarded. The default value is `Integer.MAX_VALUE` (which, for practical purposes, means “no limit”).

► public void `setMaximumItemCount(int maximum)`;

Sets the maximum number of items that will be retained by the series. If the series already contains more than `maximum` items, the oldest items are discarded until the series contains exactly `maximum` items, and a `SeriesChangeEvent` is sent to registered listeners. If `maximum` is negative, an `IllegalArgumentException` is thrown.

The `maximumItemAge` attribute fixes the *maximum age* of the items that the series can hold. Recall that a `TimeSeries` instance stores its data items using a particular subclass of `RegularTimePeriod` (`Year`, `Month`, `Day`, `Hour`, and so on—see the `getTimePeriodClass()` method). The maximum age of the items in a time series is specified in terms of the number (`maximumItemAge`) of the particular time period used by the series.

For example, suppose that you have a time series containing monthly data, and you want to automatically discard any items that are more than one year old. To do this, set the `maximumItemAge` to 12, and the time series will discard any data item that is more than 12 months older *than the most recent item in the series*:

► public int `getMaximumItemAge()`;

Returns the maximum age of items in the series (measured as a number of time periods relative to the most recent item in the series). The default value is `Integer.MAX_VALUE` (unless you change this, no items will be removed due to their age).

► public void `setMaximumItemAge(int periods)`;

Sets the `maximumItemAge` attribute, which specifies the maximum age of data items in the series (in terms of the `RegularTimePeriod` type used by this series). Whenever a new data value is added, any data items that are older than the limit specified by `maximumItemAge` are automatically discarded.

► public void `removeAgedItems(boolean notify)`;

Removes any items in the time series that exceed the maximum item age (see `getMaximumItemAge()`), relative to the most recent item in the series. If `notify` is `true`, a `SeriesChangeEvent` is sent to all registered listeners.

► public void `removeAgedItems(long latest, boolean notify)`;

Removes any items in the time series that exceed the maximum item age, relative to `latest` (specified in milliseconds since 1-Jan-1970). If `notify` is `true`, a `SeriesChangeEvent` is sent to all registered listeners.

### 51.21.11 Utility Methods

The following utility methods are provided for convenience (none of these methods is used internally by JFreeChart):

► public `RegularTimePeriod getNextTimePeriod()`;

Returns the time period following the last time period in the series. For example, this could be used by a time series editor to add a new item at the end of the series.

```
→ public Collection getTimePeriods();
Returns a (new) collection containing references to all the time periods in this time series.

→ public Collection getTimePeriodsUniqueToOtherSeries(TimeSeries series);
Returns a (new) collection containing all the time periods in series that are NOT present in this series.
```

### 51.21.12 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object object);
Tests this time series for equality with an arbitrary object.
```

To clone the time series:

```
→ public Object clone() throws CloneNotSupportedException;
Returns a clone of this time series.
```

### 51.21.13 Notes

Some points to note:

- you can calculate the moving average of a time series using the [MovingAverage](#) utility class;

#### See Also

[TimePeriod](#), [TimeSeriesCollection](#).

## 51.22 TimeSeriesCollection

### 51.22.1 Overview

A collection of `TimeSeries` objects that can be used as the dataset for a time series chart (this class implements the `XYDataset` and `IntervalXYDataset` interfaces).

### 51.22.2 Usage

A demo (`TimeSeriesDemo.java`) is included in the JFreeChart demo collection.

### 51.22.3 Constructors

To create an *empty* time series collection:

```
→ public TimeSeriesCollection();
Creates a new (empty) collection that is pegged to the default TimeZone.

→ public TimeSeriesCollection(TimeZone zone);
Creates a new (empty) collection that is pegged to the specified TimeZone. If zone is null, the default time zone is used.
```

To create a collection containing a single time series (more can be added later):

```
→ public TimeSeriesCollection(TimeSeries series);
Creates a new collection, containing the specified series, that is pegged to the default TimeZone. If series is null, the collection will be empty.

→ public TimeSeriesCollection(TimeSeries series, TimeZone zone);
Creates a new collection, containing the specified series, that is pegged to the specified time zone. If series is null, the collection will be empty. If zone is null, the default time zone is used.
```

Once a collection has been constructed, you are free to add any number of additional time series to the collection.

### 51.22.4 Adding and Removing Series

You can add additional `TimeSeries` objects to the collection, or remove existing series from the collection, at any time—a `DatasetChangeEvent` will be fired for each update.

To add a series to the collection:

```
► public void addSeries(TimeSeries series);
```

Adds the series to the collection and sends a `DatasetChangeEvent` to all registered listeners.

To remove a series from the collection:

```
► public void removeSeries(TimeSeries series);
```

Removes a series from the collection and sends a `DatasetChangeEvent` to all registered listeners.

```
► public void removeSeries(int index);
```

Removes a series from the collection and sends a `DatasetChangeEvent` to all registered listeners.

To remove all series from the dataset:

```
► public void removeAllSeries();
```

Removes all series from the dataset.

### 51.22.5 Fetching X and Y Values

This class implements the `XYDataset` interface, so it needs to provide methods for accessing X and Y values.

To get the x-value for an item within a series:

```
► public Number getX(int series, int item);
```

Returns the x-value for an item within a series. The value returned is the number of milliseconds since 1 January 1970, 00:00:00 GMT.

```
► public double getXValue(int series, int item);
```

Returns the x-value for an item within a series. The value returned is the number of milliseconds since 1 January 1970, 00:00:00 GMT.

Each x-value must be derived from the `RegularTimePeriod` for the item in the specified series. Several factors control the conversion of the time period to a fixed point in time. The first is the time zone for the `TimeSeriesCollection`—this can be specified in the constructor. The second is the anchor point, which controls whether the x-value is positioned at the start, middle or end of the time period:

```
► public TimePeriodAnchor getXPosition();
```

Returns the anchor position used to derive the x-value for a time period within a series.

```
► public void setXPosition(TimePeriodAnchor anchor);
```

Sets the anchor point (START, MIDDLE, or END) within each time period that is used as the x-value for a data item.

To get the y-value for an item within a series:

```
► public Number getY(int series, int item);
```

Returns the y-value for an item within a series—this may be `null`.

### 51.22.6 The Range of X Values

To find the range of x-values contained in the collection:

```
► public Range getDomainRange();
```

Returns the range of values in the domain for this dataset.

```
► public Number getMinimumDomainValue();
```

Returns the minimum domain value (or x-value).

```
► public Number getMaximumDomainValue();
```

Returns the maximum domain value (or x-value).

Given that this class implements the `IntervalXYDataset` interface, which can specify an interval for each x-value, we need to be careful about how the range of x-values is determined. The `domainIsPointsInTime` flag controls the treatment of time periods in the collection when the overall range of values is being calculated. There are two possibilities:

- consider each time period as a single point, which is the case when the collection is being used as an `XYDataset`;
- consider each time period as a range of values, which is the case when the collection is being used as an `IntervalXYDataset`.

If the `domainIsPointsInTime` flag is set to `true` (the default), the former treatment is applied, and if it is set to `false` the latter treatment is applied.

► `public boolean getDomainIsPointsInTime();`  
 Returns a flag that indicates whether the domain values are considered to be points in time, or intervals.

► `public void setDomainIsPointsInTime(boolean flag);`  
 Sets the flag that controls whether the domain values are considered to be points in time or intervals, then sends a `DatasetChangeEvent` to all registered listeners. This impacts the result returned by the `getDomainRange()` method.

### 51.22.7 Other Methods

To find out how many `TimeSeries` objects are in the collection:

► `public int getSeriesCount();`  
 Returns the number of time series objects in the collection.

To get a list of all the series in the collection:

► `public List getSeries();`  
 Returns an unmodifiable list of the series within the collection.

To get a reference to a particular series:

► `public TimeSeries getSeries(int series);`  
 Returns a reference to a series in the collection.  
 ► `public TimeSeries getSeries(String name);`  
 Returns a reference to the named series.

To get the name of a series:

► `public String getSeriesName(int series);`  
 Returns the name of a series in the collection. This method is provided for convenience.

To get the number of items in a series:

► `public int getItemCount(int series);`  
 Returns the number of items in a series. This method is implemented as a requirement of the `XYDataset` interface.

The `DomainInfo` interface requires the following method, which returns the overall range of x-values contained in the collection:

► `public Range getDomainRange();`  
 Returns the overall range of x-values contained in the collection. The result is affected by the current setting of the `domainIsPointsInTime` attribute—see section ?? for details.

To get the indices of the time periods that surround a specific millisecond:

► `public int[] getSurroundingItems(int series, long milliseconds);`  
 Returns an array containing two indices for the time periods that surround the specified time.

### 51.22.8 Equality, Cloning and Serialization

This class is `Serializable` but not `Cloneable`.

To test for equality:

```
➔ public boolean equals(Object obj);
Tests the collection for equality with an arbitrary object.
```

Two collections are considered equal when:

- both collections contain the same number of `TimeSeries` objects;
- each `TimeSeries` object is equal to the corresponding series in the other collection;
- the other attributes of the collection are the same.

### 51.22.9 Notes

Points to note:

- this class extends `AbstractSeriesDataset` to provide some of the basic series information.
- this class implements the `XYDataset` and `IntervalXYDataset` interfaces.

## 51.23 TimeSeriesDataItem

### 51.23.1 Overview

This class associates a `Number` with a `RegularTimePeriod`, and is used by the `TimeSeries` class to record individual data items.

### 51.23.2 Usage

You won't normally use this class directly—the `TimeSeries` class will create instances as required.

### 51.23.3 Constructors

To create a new item:

```
➔ public TimeSeriesDataItem(RegularTimePeriod period, Number value);
Creates a new item that associates the specified period and value. You can use null to represent
a missing or unknown value, but null is not permitted for the period argument.

➔ public TimeSeriesDataItem(RegularTimePeriod period, double value);
Creates a new item that associates the specified period and value.
```

### 51.23.4 Methods

To get the time period for the item:

```
➔ public RegularTimePeriod getPeriod();
Returns the period for the item (the period is immutable and never null.)
```

To get/set the value for the item:

```
➔ public Number getValue();
Returns the value for the item (or null to represent a missing or unknown value).

➔ public void setValue(final Number value);
Sets the value for the item (use null to represent a missing or unknown value).
```

### 51.23.5 Notes

This class has a number of important features:

- the class implements the `Comparable` interface, allowing data items to be sorted into time order using standard Java API calls;
- the time period element is immutable, so that when a collection of objects is held in sorted order, the sorted property cannot inadvertently be broken;
- the class implements the `Cloneable` interface, so that instances of this class can be easily cloned;
- the class implements the `Serializable` interface.

## 51.24 TimeSeriesTableModel

An initial attempt to display a time series in a `JTable`.

## 51.25 TimeTableXYDataset

### 51.25.1 Overview

A dataset that represent a table of values where each column represents a series. Each row contains the values (possibly `null`) that correspond to a particular time period (represented by any subclass of `RegularTimePeriod`). This class implements the `TableXYDataset` interface and so is useful for creating stacked area and bar charts with time-based data.

### 51.25.2 Constructors

The following constructors are available:

- `public TimeTableXYDataset();`  
Creates a new (empty) dataset that uses the default `TimeZone` and `Locale`.
- `public TimeTableXYDataset(TimeZone zone);`  
Creates a new (empty) dataset that uses the specified `TimeZone` and the default `Locale`. Passing `null` for the `zone` argument is not permitted.
- `public TimeTableXYDataset(TimeZone zone, Locale locale);`  
Creates a new (empty) dataset that uses the specified `TimeZone` and `Locale`. Passing `null` is not permitted for either argument.

### 51.25.3 Adding and Removing Data

To add a data item:

- `public void add(RegularTimePeriod period, double y, String seriesName);`  
Adds a value corresponding to the specified time period for a particular series (if there is an existing value, it is overwritten). A `DatasetChangeEvent` is sent to all registered listeners.
- `public void add(RegularTimePeriod period, Number y, String seriesName, boolean notify);`  
Adds a value (`null` permitted) corresponding to the specified time period for a particular series (if there is an existing value, it is overwritten). If `notify` is `true`, a `DatasetChangeEvent` is sent to all registered listeners.

To remove a data item:

```
→ public void remove(RegularTimePeriod period, String seriesName);
Removes the data item for the specified period and seriesName. If there are no other items for the series, the series will be removed from the dataset. If there are no other items for the specified time period, it will be removed from the dataset (thus shrinking the overall size of the table).

→ public void remove(RegularTimePeriod period, String seriesName, boolean notify);
Removes the data item for the specified period and seriesName. If there are no other items for the series, the series will be removed from the dataset. If there are no other items for the specified time period, it will be removed from the dataset (thus shrinking the overall size of the table). If notify is true, a DatasetChangeEvent is sent to all registered listeners.
```

To clear all data from the dataset:

```
→ public void clear(); [1.0.7]
Clears all data from the dataset (if it is not already empty) and sends a DatasetChangeEvent to all registered listeners.
```

#### 51.25.4 Methods

For determining an appropriate axis range, JFreeChart needs to determine the minimum and maximum domain values (or x-values) in the dataset. This can vary slightly depending on whether each x-value is evaluated as a “point in time” or a “period of time” (the range will be slightly larger if each x-value covers a period of time rather than a single point in time). You can set a flag in the dataset to determine the behaviour:

```
→ public boolean getDomainIsPointsInTime();
Returns a flag that determines whether the domain values are “points in time” or “periods of time”.

→ public void setDomainIsPointsInTime(boolean flag);
Sets a flag that determines whether the domain values are “points in time” or “periods of time”.
```

The x-values are represented by time periods. The actual x-value can be the start, middle or end of the time period:

```
→ public TimePeriodAnchor getXPosition();
Returns the anchor point within each time period that determines the x-value for that time period.

→ public void setXPosition(TimePeriodAnchor anchor);
Sets the anchor point (start, middle or end) within each time period that determines the x-value for that time period.

→ public int getItemCount();
Returns the number of items in each series (recall that the TableXYDataset interface requires that all series share the same x-values, which means that all series have the same number of items).

→ public int getItemCount(int series);
This method is required by the XYDataset interface—for this dataset, it returns the same value as getItemCount().

→ public int getSeriesCount();
Returns the number of series in the dataset.

→ public String getSeriesName(int series);
Returns the name of a series.

→ public Number getX(int series, int item);
Returns the x-value for an item within a series. For this dataset, the value will be represented in milliseconds since 1-Jan-1970.

→ public Number getStartX(int series, int item);
Returns the start value of the x-interval for an item within a series.
```

```

    ➔ public Number getEndX(int series, int item);
    Returns the end value of the x-interval for an item within a series.

    ➔ public Number getY(int series, int item);
    Returns the y-value for an item within a series.

    ➔ public Number getStartY(int series, int item);
    Returns the start value of the y-interval for an item within a series.

    ➔ public Number getEndY(int series, int item);
    Returns the end value of the y-interval for an item within a series.

    ➔ public Number getMinimumDomainValue();
    Returns the lowest x-value in the dataset.

    ➔ public Number getMaximumDomainValue();
    Returns the highest x-value in the dataset.

    ➔ public Range getDomainRange();
    Returns a range for the x-values in the dataset.

```

### 51.25.5 Notes

Some points to note:

- a demo application (`StackedXYBarChartDemo2.java`) is included in the JFreeChart demo collection.

#### See Also:

[StackedXYAreaRenderer](#), [StackedXYBarRenderer](#), [TableXYDataset](#).

## 51.26 Week

### 51.26.1 Overview

A subclass of `RegularTimePeriod` that represents one week in a particular year. This class is designed to be used with the `TimeSeries` class, but (hopefully) is general enough to be used in other situations.

As far as possible, this class tries to follow the same definition of a “week” as used by Java’s `Calendar` class. The weeks are numbered from 1 to 53 with:

- week 1 of a given year often begins during December of the previous year, but always ends in January of the given year;
- week 53 is often not required, in which case it is considered to have zero length.

Different locales make different assumptions about the first day of the week, and these differences are taken into account when mapping a `Week` instance to the time line.

### 51.26.2 Constructors

To construct a `Week` instance:

```

    ➔ public Week(int week, Year year);
    Creates a new Week instance. The week argument should be in the range 1 to 53.

    ➔ public Week(int week, int year);
    Creates a new Week instance.

```

To construct a `Week` instance based on a `java.util.Date`:

```

    ➔ public Week(Date time);
    Creates a new Week instance.

    ➔ public Week();
    Creates a new Week instance based on the current system time.

```

### 51.26.3 Methods

To access the week:

```
➔ public int getWeek();
```

Returns the week (in the range 1 to 53).

To access the year:

```
➔ public Year getYear();
```

Returns the year.

There is no method to *set* the week or the year, because this class is immutable.

Given a `Week` object, you can create an instance representing the previous week or the next week:

```
➔ public RegularTimePeriod previous();
```

Returns the previous week, or `null` if the lower limit of the range is reached.

```
➔ public RegularTimePeriod next();
```

Returns the next week, or `null` if the upper limit of the range is reached.

To convert a `Week` object to a `String` object:

```
➔ public String toString();
```

Returns a string representing the week.

### 51.26.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Week` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

**See Also:**

[Year](#).

## 51.27 Year

### 51.27.1 Overview

A class that represents a calendar year (for example, “2003”). This class extends `RegularTimePeriod`.

### 51.27.2 Usage

A typical use for this class is for creating `TimeSeries` objects for *annual data*. For example:

```
TimeSeries t1 = new TimeSeries("Series 1", "Year", "Value", Year.class);
t1.add(new Year(1990), new Double(50.1));
t1.add(new Year(1991), new Double(12.3));
t1.add(new Year(1992), new Double(23.9));
t1.add(new Year(1993), new Double(83.4));
t1.add(new Year(1994), new Double(-34.7));
t1.add(new Year(1995), new Double(76.5));
t1.add(new Year(1996), new Double(10.0));
t1.add(new Year(1997), new Double(-14.7));
t1.add(new Year(1998), new Double(43.9));
t1.add(new Year(1999), new Double(49.6));
t1.add(new Year(2000), new Double(37.2));
t1.add(new Year(2001), new Double(17.1));
```

### 51.27.3 Constructors

To create a new year:

► `public Year(int year);`

Creates a new `Year` instance. The `year` argument should be in the range 1900 to 9999.

To construct a `Year` instance based on a `java.util.Date`:

► `public Year(Date time);`

Creates a new `Year` instance.

A default constructor is provided:

► `public Year();`

Creates a new `Year` instance based on the current system time.

### 51.27.4 Methods

To access the year:

► `public int getYear();`

Returns the year.

There is no method to *set* the year, because this class is immutable.

Given a `Year` object, you can create an instance representing the previous year:

► `public RegularTimePeriod previous();`

Returns the previous year, or `null` if the lower limit of the range is reached.

...or the next:

► `public RegularTimePeriod next();`

Returns the next year, or `null` if the upper limit of the range is reached.

To convert a `Year` object to a `String` object:

► `public String toString();`

Returns a string representing the year.

To convert a `String` object to a `Year` object:

► `public static Year parseYear(String s) throws TimePeriodFormatException;`

Parses the string and, if possible, returns a `Year` object.

### 51.27.5 Notes

Some points to note:

- in the current implementation, the year can be in the range 1900 to 9999.
- the `Year` class is immutable—this is a requirement for all `RegularTimePeriod` subclasses.

# Chapter 52

## Package: org.jfree.data.time.ohlc

### 52.1 Introduction

This package contains a collection of classes that provide a dataset that implements the `OHLCDataSet` interface. This dataset is typically used to represent data from financial markets—the open and close values refer to the opening and closing prices for a trading session, while the high and low values refer to the highest and lowest trading prices during the session.

*This package was first introduced in JFreeChart version 1.0.4.*

### 52.2 OHLC

#### 52.2.1 Overview

A simple class that records the four values required by the `OHLCItem` class. You shouldn't need to use this class directly.

*This class was first introduced in JFreeChart version 1.0.4.*

#### 52.2.2 Constructor

To create a new instance:

► `public OHLC(double open, double high, double low, double close); [1.0.4]`  
Creates a new instance with the specified data values.

#### 52.2.3 Methods

The following methods provide access to the values supplied to the constructor:

► `public double getOpen(); [1.0.4]`  
Returns the open value.  
► `public double getClose(); [1.0.4]`  
Returns the close value.  
► `public double getHigh(); [1.0.4]`  
Returns the high value.  
► `public double getLow(); [1.0.4]`  
Returns the low value.

### 52.2.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj); [1.0.4]
    Tests this instance for equality with an arbitrary object.
```

Instances of this class are `Serializable` but not `Cloneable` (cloning is pointless, because instances are immutable).

### 52.2.5 Notes

This class performs no validation of the data values (for example, it doesn't check that the low value is less than or equal to the high value).

## 52.3 OHLCItem

### 52.3.1 Overview

A data item that can be stored in an `OHLCSeries`. This class extends `ComparableObjectItem`.

*This class was first introduced in JFreeChart version 1.0.4.*

### 52.3.2 Constructor

This class defines a single constructor:

```
➔ public OHLCItem(RegularTimePeriod period, double open, double high, double low, double close);
    [1.0.4]
    Creates a new instance with the specified values.
```

### 52.3.3 Methods

The following methods provide access to the attributes specified in the constructor:

```
➔ public RegularTimePeriod getPeriod(); [1.0.4]
    Returns the time period for the data values.

➔ public double getYValue(); [1.0.4]
    Returns getCloseValue().

➔ public double getOpenValue(); [1.0.4]
    Returns the open value.

➔ public double getHighValue(); [1.0.4]
    Returns the high value.

➔ public double getLowValue(); [1.0.4]
    Returns the low value.

➔ public double getCloseValue(); [1.0.4]
    Returns the close value.
```

### 52.3.4 Notes

Some points to note:

- this class is used by the `OHLCSeries` class, you should need to use this class directly in your own code;
- internally, this class makes use of the `OHLC` class.

## 52.4 OHLCSeries

### 52.4.1 Overview

An `OHLCSeries` represents an ordered list of items where each item records four values (open, high, low and close) against a time period. One or more instances of this class can be added to an `OHLCSeriesCollection` to create a dataset for a chart.

*This class was first introduced in JFreeChart version 1.0.4.*

### 52.4.2 Constructor

This class defines a single constructor:

```
► public OHLCSeries(Comparable key); [1.0.4]
Creates a new empty series with the specified key (which must not be null). The key (often a
String) is used to identify the series.
```

### 52.4.3 Methods

In addition to the methods inherited from `ComparableObjectSeries`:

```
► public RegularTimePeriod getPeriod(int index); [1.0.4]
Returns the time period for the specified item in the series. If index is not in the range 0 to
getItemCount(), this method throws an IndexOutOfBoundsException.

► public ComparableObjectItem getDataItem(int index); [1.0.4]
Returns the data record for the specified item in the series. This will be an instance of OHLCItem.
If index is not in the range 0 to getItemCount(), this method throws an IndexOutOfBoundsException.

► public void add(RegularTimePeriod period, double open, double high, double low,
double close); [1.0.4]
Adds a new item to the series and sends a SeriesChangeEvent to all registered listeners. Attempting
to add an item for a period that already exists in the series will cause a SeriesException.
All the data items in the series should use the same class of period, adding a different type of
period will cause an IllegalArgumentException.
```

### 52.4.4 Equals, Cloning and Serialization

The inherited `equals()` method is sufficient to distinguish differences between two instances of this class. Instances of this class are `Cloneable` and `Serializable`.

### 52.4.5 Notes

This class is a subclass of `ComparableObjectSeries`.

#### See Also:

`OHLCSeriesCollection`.

## 52.5 OHLCSeriesCollection

### 52.5.1 Overview

An `OHLCSeriesCollection` contains zero, one, or many `OHLCSeries` instances, and presents this data via the `OHLCDataSet` interface. In other words, this class is a *dataset* that can be used to create charts that require open-high-low-close data (this kind of data is usually related to financial markets trading).

*This class was first introduced in JFreeChart version 1.0.4.*

### 52.5.2 Constructor

This class defines a single constructor:

► `public OHLCSeriesCollection(); [1.0.4]`  
Creates a new empty collection.

### 52.5.3 Methods

To add a series:

► `public void addSeries(OHLCSeries series); [1.0.4]`  
Adds the specified `series` to the collection. If `series` is `null`, this method throws an `IllegalArgumentException`. Note that this method will allow you to add a series with a key that is not unique within the collection, but you should avoid doing this.

To get the number of series in the collection:

► `public int getSeriesCount(); [1.0.4]`  
Returns the number of series in the collection.

► `public Comparable getSeriesKey(int series); [1.0.4]`  
Returns the key for a series in the collection.

► `public int getItemCount(int series); [1.0.4]`  
Returns the number of items in a series.

► `public OHLCSeries getSeries(int series); [1.0.4]`  
Returns a series from the collection.

► `protected synchronized long getX(RegularTimePeriod period); [1.0.4]`  
Returns the millisecond value for the specified period.

► `public double getXValue(int series, int item); [1.0.4]`  
Returns the x-value for an item in a series.

► `public double getOpenValue(int series, int item); [1.0.4]`  
Returns the open value for an item in a series.

► `public double getHighValue(int series, int item); [1.0.4]`  
Returns the high value for an item in a series.

► `public double getLowValue(int series, int item); [1.0.4]`  
Returns the low value for an item in a series.

► `public double getCloseValue(int series, int item); [1.0.4]`  
Returns the close value for an item in a series.

► `public double getVolumeValue(int series, int item); [1.0.4]`  
Returns `Double.NaN`, because this dataset does not store any trading volume information.

The following methods are equivalent to the above, but return `Number` instances instead of `double` primitives. Each call to these methods allocates a new object instance, so you should avoid using these methods if possible:

► `public Number getX(int series, int item); [1.0.4]`  
Returns the x-value for an item in a series.

► `public Number getY(int series, int item); [1.0.4]`  
Returns the y-value (same as the close value) for an item in a series.

► `public Number getOpen(int series, int item); [1.0.4]`  
Returns the open value for an item in a series.

► `public Number getClose(int series, int item); [1.0.4]`  
Returns the close value for an item in a series.

► `public Number getHigh(int series, int item); [1.0.4]`  
Returns the high value for an item in a series.

► `public Number getLow(int series, int item); [1.0.4]`  
Returns the low value for an item in a series.

► `public Number getVolume(int series, int item); [1.0.4]`  
Returns `null` as this dataset does not store volume data.

#### 52.5.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj); [1.0.4]`  
Tests this dataset for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### 52.5.5 Notes

This dataset does not store trading volume information.

**See Also:**

[DefaultHighLowDataset](#), [DefaultOHLCDataset](#).

# Chapter 53

## Package: org.jfree.data.xml

### 53.1 Introduction

This package contains interfaces and classes that provide basic support for reading datasets from XML files. In the current release, there is support for `PieDataset` and `CategoryDataset`. It is intended that other dataset types will be supported in the future.

### 53.2 Usage

In normal usage, you will access the facilities provided by this package via methods in the `DatasetReader` class. The following examples are provided in the JFreeChart demo collection:

- `XMLBarChartDemo.java`
- `XMLPieChartDemo.java`

### 53.3 CategoryDatasetHandler

#### 53.3.1 Overview

A SAX handler that creates a `CategoryDataset` by processing the elements in an XML document.

#### 53.3.2 Usage

In most cases, you won't need to use this class directly. Instead, use the `DatasetReader` class. For an example, see the `XMLBarChartDemo` included in the JFreeChart demo collection.

#### 53.3.3 XML Format

The format supported by the handler is illustrated by the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!!-- Sample data for JFreeChart. -->
<CategoryDataset>
  <Series name = "Series 1">
    <Item>
      <Key>Category 1</Key>
      <Value>15.4</Value>
    </Item>
    <Item>
      <Key>Category 2</Key>
      <Value>12.7</Value>
```

```

</Item>
<Item>
  <Key>Category 3</Key>
  <Value>5.7</Value>
</Item>
<Item>
  <Key>Category 4</Key>
  <Value>9.1</Value>
</Item>
</Series>

<Series name = "Series 2">
  <Item>
    <Key>Category 1</Key>
    <Value>45.4</Value>
  </Item>
  <Item>
    <Key>Category 2</Key>
    <Value>73.7</Value>
  </Item>
  <Item>
    <Key>Category 3</Key>
    <Value>23.7</Value>
  </Item>
  <Item>
    <Key>Category 4</Key>
    <Value>19.4</Value>
  </Item>
</Series>

</CategoryDataset>

```

The `<CategoryDataset>` element can contain any number of `<Series>` elements, and each `<Series>` element can contain any number of `<Item>` elements.

### 53.3.4 Notes

This class delegates work to the [CategorySeriesHandler](#) class.

## 53.4 CategorySeriesHandler

### 53.4.1 Overview

A SAX handler that reads a `<Series>` sub-element within a category dataset XML file. Work is delegated to this class by the [CategoryDatasetHandler](#) class.

## 53.5 DatasetReader

### 53.5.1 Overview

This class contains utility methods for reading datasets from XML files. In the current release, support is included for [PieDataset](#) and [CategoryDataset](#).

### 53.5.2 Usage

Two applications ([XMLPieChartDemo](#) and [XMLBarChartDemo](#)) that demonstrate how to use this class are included in the JFreeChart demo collection.

## 53.6 DatasetTags

### 53.6.1 Overview

An interface that defines constants for the literal text used in the element tags within the XML documents.

Attribute:	Value:
PIEDATASET_TAG	PieDataset
CATEGORYDATASET_TAG	CategoryDataset
SERIES_TAG	Series
ITEM_TAG	Item
KEY_TAG	Key
VALUE_TAG	Value

Table 53.1: Attributes for the `DatasetTags` interface

## 53.7 ItemHandler

### 53.7.1 Overview

A SAX handler that reads a *key/value* pair.

### 53.7.2 Usage

You should not need to use this class directly. Work is delegated to this handler by the `PieDatasetHandler` class.

### 53.7.3 Notes

This class delegates some work to the `KeyHandler` class.

## 53.8 KeyHandler

### 53.8.1 Overview

A SAX handler that reads a *key* element from an XML file.

### 53.8.2 Usage

You should not need to use this class directly. Work is delegated to this class by the `ItemHandler` class.

### 53.8.3 Notes

A key can be any instance of `Comparable`, but the handler always uses the `String` class to represent keys.

## 53.9 PieDatasetHandler

### 53.9.1 Overview

A SAX handler for reading a `PieDataset` from an XML file.

### 53.9.2 Usage

In most cases, you won't need to use this class directly. Instead, use the `DatasetReader` class. For an example, see the `XMLPieChartDemo` application included in the JFreeChart demo collection.

### 53.9.3 XML Format

The format supported by the handler is illustrated by the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- A sample pie dataset for JFreeChart. -->

<PieDataset>
  <Item>
    <Key>Java</Key>
    <Value>15.4</Value>
  </Item>
  <Item>
    <Key>C++</Key>
    <Value>12.7</Value>
  </Item>
  <Item>
    <Key>PHP</Key>
    <Value>5.7</Value>
  </Item>
  <Item>
    <Key>Python</Key>
    <Value>9.1</Value>
  </Item>
</PieDataset>
```

The `<PieDataset>` element can contain any number of `<Item>` elements.

### 53.9.4 Notes

This class delegates some work to the [ItemHandler](#) class.

## 53.10 RootHandler

### 53.10.1 Overview

The base handler class that provides support for a “sub-handler stack”. While processing an XML element, a handler can push a sub-handler onto the stack and delegate work to it (usually the processing of a sub-element). When the sub-handler is finished its work, it gets popped from the stack, and the original handler resumes control. In this way, nested elements within the XML file can be processed by different classes.

## 53.11 ValueHandler

### 53.11.1 Overview

A SAX handler that processes numerical values.

# Chapter 54

## Package: org.jfree.data.xy

### 54.1 Introduction

This package contains the [XYDataset](#) interface, extensions and implementing classes. These are used to supply data to the [XYItemRenderer](#) instances that are managed by the [XYPlot](#) class.

### 54.2 AbstractIntervalXYDataset

#### 54.2.1 Overview

A base class that can be used to implement an [IntervalXYDataset](#) (extends [AbstractXYDataset](#)).

#### 54.2.2 Methods

This class implements methods that return `double` primitives for the start and end values of the x and y-intervals:

```
→ public double getStartXValue(int series, int item);  
    Returns the start value for the x-interval.  
  
→ public double getEndXValue(int series, int item);  
    Returns the end value for the x-interval.  
  
→ public double getStartYValue(int series, int item);  
    Returns the start value for the y-interval.  
  
→ public double getEndYValue(int series, int item);  
    Returns the end value for the y-interval.
```

The above methods rely on the corresponding methods that return `Number` objects being implemented—see the [IntervalXYDataset](#) interface for details.

### 54.3 AbstractXYDataset

#### 54.3.1 Overview

A base class that can be used to implement an [XYDataset](#). This provides default implementations of the accessor methods that return `double` primitives. In dataset implementations where the data is actually stored using primitives, it is highly recommended that you override these methods to fetch those primitive values directly (to avoid the creation of temporary `Number` object instances).

### 54.3.2 Methods

This class implements methods that return `double` primitives for the x and y values:

- `public double getXValue(int series, int item);`  
Returns the x-value. This method relies on the `getX()` method being implemented.
- `public double getYValue(int series, int item);`  
Returns the y-value. If the value is missing or unknown, this method will return `Double.NaN`.

The above methods rely on the `getX()` and `getY()` methods being implemented—see the [XYZDataset](#) interface for details.

#### See Also

[AbstractIntervalXYDataset](#), [AbstractXYZDataset](#).

## 54.4 AbstractXYZDataset

### 54.4.1 Overview

An abstract base class that can be used to implement the [XYZDataset](#) interface. This class extends [AbstractXYDataset](#) to provide a default implementation of the `getZValue()` method.

### 54.4.2 Methods

This class implements a method that returns a `double` primitive for the z value:

- `public double getZValue(int series, int item);`  
Returns the z-value. This method relies on the `getZ()` method to access the z-value.

## 54.5 CategoryTableXYDataset

### 54.5.1 Overview

A dataset that implements the [TableXYDataset](#) interface, so that it can be used with the [StackedXYAreaRenderer](#) and [StackedXYBarRenderer](#) classes.

### 54.5.2 Constructor

To create a new dataset:

- `public CategoryTableXYDataset();`  
Creates a new dataset.

### 54.5.3 Adding and Removing Data

When adding and removing data, bear in mind that all series must share the same set of x-values (this is required by the [TableXYDataset](#) interface). When you add a new x-value to one series, the same x-value is implicitly added to all the other series (with a `null` y-value).

To add an item to a series:

- `public void add(double x, double y, String seriesName);`  
Adds a new item for the specified series and sends a [DatasetChangeEvent](#) to all registered listeners.
- `public void add(Number x, Number y, String seriesName, boolean notify);`  
Adds a new item for the specified series and, if requested, sends a [DatasetChangeEvent](#) to all registered listeners.

To remove an item:

```
→ public void remove(double x, String seriesName);
Removes the item with the specified x-value from a series and sends a DatasetChangeEvent to all registered listeners.
```

```
→ public void remove(Number x, String seriesName, boolean notify);
Removes the item with the specified x-value from a series and, if requested, sends a DatasetChangeEvent to all registered listeners.
```

#### 54.5.4 Accessing the Data Values

To access the data values:

```
→ public Number getX(int series, int item);
Returns the x-value for an item in a series.
```

```
→ public Number getY(int series, int item);
Returns the y-value for an item in a series (this may be null).
```

#### 54.5.5 X-Intervals

This dataset can derive an x-interval about the x-value, in order to support the requirements of the [IntervalXYDataset](#) interface:

```
→ public Number getStartX(int series, int item);
Returns the start value of the x-interval for an item in a series.
```

```
→ public Number getEndX(int series, int item);
Returns the end value of the x-interval for an item in a series.
```

No y-interval is defined, so the following methods return the same value as `getY()`:

```
→ public Number getStartY(int series, int item);
Returns the same value as getY().
```

```
→ public Number getEndY(int series, int item);
Returns the same value as getY().
```

To control the x-interval width, the following methods are provided:

```
→ public double getIntervalPositionFactor();
Returns the interval position factor, which controls how the x-interval is positioned relative to the x-value.
```

```
→ public void setIntervalPositionFactor(double d);
Sets the interval position factor. This is a number between 0.0 and 1.0, where 0.5 means the x-interval is centered over the x-value.
```

```
→ public double getIntervalWidth();
Returns the interval width. The default value is 1.0.
```

```
→ public void setIntervalWidth(double d);
Sets the interval width.
```

```
→ public boolean isAutoWidth();
Returns the flag the controls whether the interval width is automatically calculated.
```

```
→ public void setAutoWidth(boolean b);
Sets the flag that controls whether the interval width is automatically calculated.
```

### 54.5.6 Other Methods

Other methods include:

- ▶ `public int getSeriesCount();`  
Returns the number of series in the dataset.
- ▶ `public String getSeriesName(int series);`  
Returns the name of a series.
- ▶ `public int getItemCount();`  
Returns the number of items for each series in the dataset.
- ▶ `public int getItemCount(int series);`  
Returns the number of items for a specific series. Since the `TableXYDataset` interface requires all the series to have the same number of items, this method returns the same value as `getItemCount()`.
- ▶ `public Range getDomainRange();`  
Returns the range of x-values represented by the dataset. This takes into account the interval width.
- ▶ `public Number getMaximumDomainValue();`  
Returns the maximum x-value in the dataset.
- ▶ `public Number getMinimumDomainValue();`  
Returns the minimum x-value in the dataset.

#### See Also

[TableXYDataset](#).

## 54.6 DefaultHighLowDataset

### 54.6.1 Overview

A dataset that stores a single series consisting of *open-high-low-close* pricing data (typical trading data in financial markets). This dataset implements the `OHLCDataset` interface. The dataset is read-only, once it is created you cannot update the values.

You should also look at the `DefaultOHLCDataset` class, which performs a very similar role using a slightly different approach.

### 54.6.2 Constructors

To create a new instance:

- ▶ `public DefaultHighLowDataset(Comparable seriesKey, Date[] date, double[] high, double[] low, double[] open, double[] close, double[] volume);`  
Creates a new dataset with a single series containing the values specified in the supplied arrays. None of the arguments should be `null`, and all the arrays should have the same length. Internally, the dataset will store the supplied values as `Number` object instances.

### 54.6.3 General Methods

This dataset implements the methods required by the `OHLCDataset` interface. Note that the series index argument in all methods is ignored, because this dataset always contains exactly one series.

- ▶ `public int getSeriesCount();`  
Returns 1, because this dataset always contains exactly one series.
- ▶ `public Comparable getSeriesKey(int series);`  
Returns the series key (never `null`). The `series` argument is ignored.

```

→ public int getItemCount(int series);
Returns the number of items in the series. The series argument is ignored.

→ public Number getX(int series, int item);
Returns the x-value for the specified item. The series argument is ignored. Note that this
method creates a new Long instance every time it is called. You can avoid creating a new object
instance by calling the (inherited) getXValue(int, int) method instead.

→ public Date getXDate(int series, int item);
Returns the x-value for the specified item, as a Date (which is how the value is stored internally).
The series argument is ignored.

→ public Number getY(int series, int item);
Equivalent to getClose(int, int)—see below.

→ public Number getHigh(int series, int item);
Returns the high price for the specified item. The series argument is ignored.

→ public double getHighValue(int series, int item);
Returns the high price for the specified item, as a double. The series argument is ignored.

→ public Number getLow(int series, int item);
Returns the low price for the specified item. The series argument is ignored.

→ public double getLowValue(int series, int item);
Returns the low price for the specified item, as a double. The series argument is ignored.

→ public Number getOpen(int series, int item);
Returns the opening price for the specified item. The series argument is ignored.

→ public double getOpenValue(int series, int item);
Returns the opening price for the specified item, as a double. The series argument is ignored.

→ public Number getClose(int series, int item);
Returns the closing price for the specified item. The series argument is ignored.

→ public double getCloseValue(int series, int item);
Returns the closing price for the specified item, as a double. The series argument is ignored.

→ public Number getVolume(int series, int item);
Returns the volume for the specified item. The series argument is ignored.

→ public double getVolumeValue(int series, int item);
Returns the volume for the specified item, as a double. The series argument is ignored.

→ public static Number[] createNumberArray(double[] data);
A utility method that converts an array of double primitives to an array of Number instances.
This is used in the constructor to convert the supplied arrays. If data is null, this method
throws a NullPointerException.

```

#### 54.6.4 Equals, Cloning and Serialization

This class does not provide an override of the `equals(Object)` method (but should—it will be done for 1.0.4). Instances of this class are `Cloneable`<sup>1</sup> and `Serializable`.

### 54.7 DefaultIntervalXYDataset

#### 54.7.1 Overview

A dataset that implements the `IntervalXYDataset` interface, using primitive (`double`) arrays for the data storage. This class allows you to add arbitrary data, but doesn't (currently) provide any facilities for updating or removing data.

*This class was first introduced in JFreeChart version 1.0.3.*

---

<sup>1</sup>The dataset is immutable, so cloning isn't really necessary.

### 54.7.2 Constructors

To create a new instance:

► `public DefaultIntervalXYDataset(); [1.0.3]`

Creates a new dataset, initially empty. Use the `addSeries()` method to populate the dataset.

### 54.7.3 General Methods

The following methods provide basic information about the series in the dataset:

► `public int getSeriesCount(); [1.0.3]`

Returns the number of series in the dataset.

► `public Comparable getSeriesKey(int series); [1.0.3]`

Returns the key for a series in the dataset.

► `public int getItemCount(int series); [1.0.3]`

Returns the number of items in a series.

### 54.7.4 Accessing Dataset Values

This dataset stores its data using arrays of `double` primitives, so if possible you should use the accessor methods that return `double` primitives in preference to the methods that return `Number` instances:

► `public double getXValue(int series, int item); [1.0.3]`

Returns the x-value for an item in the specified series, as a `double`.

► `public double getYValue(int series, int item); [1.0.3]`

Returns the y-value for an item in the specified series, as a `double`.

► `public double getStartXValue(int series, int item); [1.0.3]`

Returns the starting x-value for an item in the specified series, as a `double`.

► `public double getEndXValue(int series, int item); [1.0.3]`

Returns the ending x-value for an item in the specified series, as a `double`.

► `public double getStartYValue(int series, int item); [1.0.3]`

Returns the starting y-value for an item in the specified series, as a `double`.

► `public double getEndYValue(int series, int item); [1.0.3]`

Returns the ending y-value for an item in the specified series, as a `double`.

A set of corresponding methods returns the data values as `Number` objects. All these methods create a new `Number` instance *each time they are called*, so you should avoid calling them if possible.

► `public Number getX(int series, int item); [1.0.3]`

Returns the value from `getXValue(int, int)`, wrapped in a new `Double` instance.

► `public Number getY(int series, int item); [1.0.3]`

Returns the value from `getYValue(int, int)`, wrapped in a new `Double` instance.

► `public Number getStartX(int series, int item); [1.0.3]`

Returns the value from `getStartXValue(int, int)`, wrapped in a new `Double` instance.

► `public Number getEndX(int series, int item); [1.0.3]`

Returns the value from `getEndXValue(int, int)`, wrapped in a new `Double` instance.

► `public Number getStartY(int series, int item); [1.0.3]`

Returns the value from `getStartYValue(int, int)`, wrapped in a new `Double` instance.

► `public Number getEndY(int series, int item); [1.0.3]`

Returns the value from `getEndYValue(int, int)`, wrapped in a new `Double` instance.

### 54.7.5 Adding Data

To add data to the dataset:

```
► public void addSeries(Comparable seriesKey, double[][] data); [1.0.3]
    Adds a new series to the dataset. You should use a seriesKey that is immutable and Serializable,
    otherwise the dataset will not support cloning and serialization.
```

The `data` array should contain six sub-arrays, of equal length:

- `data[0]` contains the x-values;
- `data[1]` contains the starting values of the x-intervals;
- `data[2]` contains the ending values of the x-intervals;
- `data[3]` contains the y-values;
- `data[4]` contains the starting values of the y-intervals;
- `data[5]` contains the ending values of the y-intervals.

The format of the `data[] []` array in the `addSeries()` method is best illustrated by an example. The following code (copied from `XYBarChartDemo6.java`) creates a new dataset containing a single series with four data items:

```
DefaultIntervalXYDataset dataset = new DefaultIntervalXYDataset();
double[] x = {1.0, 2.0, 3.0, 4.0};
double[] startx = {0.9, 1.8, 2.7, 3.6};
double[] endx = {1.1, 2.2, 3.3, 4.4};
double[] y = {1.0, 2.0, 3.0, 4.0};
double[] starty = {0.9, 1.8, 2.7, 3.6};
double[] endy = {1.1, 2.2, 3.3, 4.4};
double[][] data = new double[][] {x, startx, endx, y, starty, endy};
dataset.addSeries("Series 1", data);
```

Currently, no methods are provided to remove or modify a series in the dataset.

### 54.7.6 Equals, Cloning and Serialization

This dataset overrides the `equals(Object)` method:

```
► public boolean equals(Object obj);
    Tests this dataset for equality with an arbitrary object.
```

Instances of this class are `Cloneable`<sup>2</sup> and `Serializable`.

### 54.7.7 Notes

Some points to note:

- a demo (`XYBarChartDemo6.java`) is included in the JFreeChart demo collection;
- an alternative dataset implementation is provided by the `XYIntervalSeries` and `XYIntervalSeriesCollection` classes.

#### See Also

[IntervalXYDataset](#).

---

<sup>2</sup>Cloning is broken in 1.0.3 but should be working in 1.0.4.

## 54.8 DefaultOHLCDataset

### 54.8.1 Overview

A dataset that stores a single series consisting of *open-high-low-close* pricing data (typical trading data in financial markets). This dataset implements the `OHLCDataset` interface. The dataset is read-only, once it is created you cannot update the values.

Two alternative implementation of the `OHLCDataset` interface are:

- `DefaultHighLowDataset`;
- `OHLCSeriesCollection`.

### 54.8.2 Constructors

To create a new dataset:

```
→ public DefaultOHLCDataset(String name, OHLCDATAItem[] data);
Creates a new dataset. The dataset has one series with the specified name and data items. The items should be in date order (or you should call the sortDataByDate() method immediately after creating the dataset).
```

### 54.8.3 Methods

This dataset implements the methods required by the `OHLCDataset` interface. Note that the series index argument in all methods is ignored, because this dataset always contains exactly one series.

To get the series key:

```
→ public Comparable getSeriesKey(int series);
Returns the name of the specified series. Since this dataset only supports one series, the same name is returned irrespective of the series argument.
```

A range of methods provide access to the data values for each item in the dataset:

```
→ public Number getX(int series, int item);
Returns the x-value for the specified item as a Long. The series argument is ignored.

→ public Date getXDate(int series, int item);
Returns the x-value for the specified item as a Date. The series argument is ignored.

→ public Number getY(int series, int item);
Returns the closing price for the specified item. The series argument is ignored.

→ public Number getHigh(int series, int item);
Returns the high value for the specified item. The series argument is ignored.

→ public double getHighValue(int series, int item);
Returns the high value for the specified item as a double. The series argument is ignored.

→ public Number getLow(int series, int item);
Returns the low value for the specified item. The series argument is ignored.

→ public double getLowValue(int series, int item);
Returns the low value for the specified item as a double. The series argument is ignored.

→ public Number getOpen(int series, int item);
Returns the open value for the specified item. The series argument is ignored.

→ public double getOpenValue(int series, int item);
Returns the open value for the specified item as a double. The series argument is ignored.

→ public Number getClose(int series, int item);
Returns the close value for the specified item. The series argument is ignored.
```

- ▶ `public double getCloseValue(int series, int item);`  
Returns the close value for the specified item as a `double`. The `series` argument is ignored.
- ▶ `public Number getVolume(int series, int item);`  
Returns the volume value for the specified item. The `series` argument is ignored.
- ▶ `public double getVolumeValue(int series, int item);`  
Returns the volume value for the specified item as a `double`. The `series` argument is ignored.

To sort the data items into date order:

- ▶ `public void sortDataByDate();`  
Sorts the array of data items into ascending order by date.

#### 54.8.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- ▶ `public boolean equals(Object obj);`  
Tests this dataset for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

#### See Also

[DefaultHighLowDataset](#), [OHLCDataSet](#), [OHLCSeriesCollection](#).

### 54.9 DefaultTableXYDataset

#### 54.9.1 Overview

An implementation of the `XYDataset` interface where all series share a common set of x-values. This dataset implements `TableXYDataset` and so can be used to create stacked area charts.

#### 54.9.2 Constructor

To create a new dataset:

- ▶ `public DefaultTableXYDataset();`  
Creates a new empty dataset with `autoPrune` set to `false` (see the next constructor).
- ▶ `public DefaultTableXYDataset(boolean autoPrune);`  
Creates a new empty dataset. The `autoPrune` flag controls whether or not x-values are automatically removed when they have no corresponding y-values.

#### 54.9.3 Accessing Series

The dataset stores zero, one or many series of data items. Each series is represented by an `XYSeries`. The following methods provide access to the series in the dataset:

- ▶ `public int getSeriesCount();`  
Returns the number of series contained within this dataset.
- ▶ `public XYSeries getSeries(int series);`  
Returns a series from the dataset.
- ▶ `public Comparable getSeriesKey(int series);`  
Returns the key for the specified series.
- ▶ `public int getItemCount(int series);`  
Returns the number of items in the specified series. Note that this dataset ensures that all series share the same set of x-values, so all series have the same number of items.
- ▶ `public int getItemCount();`  
Returns the number of items in each series (this dataset ensures that all series have the same number of items).

#### 54.9.4 Accessing Data Values

To access particular values from the dataset:

- ▶ `public Number getX(int series, int item);`  
Returns the x-value for an item in a particular series.
- ▶ `public Number getStartX(int series, int item);`  
Returns the start of the x-interval for an item in a particular series.
- ▶ `public Number getEndX(int series, int item);`  
Returns the end of the x-interval for an item in a particular series.
- ▶ `public Number getY(int series, int index);`  
Returns the y-value (possibly `null`) for an item in a particular series.
- ▶ `public Number getStartY(int series, int item);`  
Returns the start of the y-interval for an item in a particular series. Since no y-interval is defined, this method always returns the y-value.
- ▶ `public Number getEndY(int series, int item);`  
Returns the end of the y-interval for an item in a particular series. Since no y-interval is defined, this method always returns the y-value.

#### 54.9.5 Adding and Removing Data

The following methods can be used to add and remove series from the dataset:

- ▶ `public void addSeries(XYSeries series);`  
Adds a series to the dataset and sends a `DatasetChangeEvent` to all registered listeners.
- ▶ `public void removeAllSeries();`  
Removes all series from the dataset and sends a `DatasetChangeEvent` to all registered listeners.
- ▶ `public void removeSeries(XYSeries series);`  
Removes a series from the dataset and sends a `DatasetChangeEvent` to all registered listeners.
- ▶ `public void removeSeries(int series);`  
Removes a series from the dataset and sends a `DatasetChangeEvent` to all registered listeners.
- ▶ `public void removeAllValuesForX(Number x);`  
Removes the item in each series that corresponds to the specified x-value, and sends a `DatasetChangeEvent` to all registered listeners.
- ▶ `public void prune();`  
Removes any x-values from the dataset that have no corresponding y-values.
- ▶ `public void updateXPoints();`  
Refreshes the cached list of x-points.

#### 54.9.6 Domain Intervals

This dataset has methods that enable you to control the “manufacture” of x-intervals for the specified x-values. This enables the dataset to be used to create bar charts, for instance.

- ▶ `public boolean isAutoWidth();`  
Returns the flag that indicates whether the interval width is automatically calculated.
- ▶ `public void setAutoWidth(boolean b);`  
Sets the flag that controls whether the interval width is automatically calculated.
- ▶ `public double getIntervalWidth();`  
Returns the x-interval width.
- ▶ `public void setIntervalWidth(double d);`  
Sets the x-interval width.

```
→ public double getIntervalPositionFactor();
Returns the interval position factor.

→ public void setIntervalPositionFactor(double d);
Sets the interval position factor. This is a value between 0.0 and 1.0 that controls how the
x-interval is positioned around the x-value. 0.0 means the x-value is at the left end of the
interval, 0.5 means that the x-value is centered within the interval and 1.0 means that the
x-value is at the right end of the interval.
```

#### 54.9.7 Other Methods

Other methods include:

```
→ public boolean equals(Object obj);
Tests this dataset for equality with an arbitrary object.

→ public int hashCode();
Returns a hash code for the dataset.

→ public Range getDomainRange();
Returns the range of values in the domain (taking into account the x-interval).

→ public Number getMaximumDomainValue();
Returns the maximum domain value (taking into account the x-interval).

→ public Number getMinimumDomainValue();
Returns the minimum domain value (taking into account the x-interval).
```

To find the state of the `autoPrune` flag:

```
→ public boolean isAutoPrune();
Returns a flag that controls whether or not x-values are automatically removed when they have
no corresponding y-values. This flag is set in the constructor and cannot be altered.

→ public void seriesChanged(SeriesChangeEvent event);
This method receives events that signal when a series contained within the dataset has changed.
You shouldn't need to call this method directly.
```

#### 54.9.8 Notes

Some points to note:

- for data where the x-values are dates, consider using the [TimeTableXYDataset](#) class;
- a demo ([StackedXYBarChartDemo1.java](#)) is included in the JFreeChart demo collection.

#### See Also

[TableXYDataset](#).

### 54.10 DefaultWindDataset

#### 54.10.1 Overview

A default implementation of the [WindDataset](#) interface. This dataset can be used with an [XYPlot](#) and a [WindItemRenderer](#) to create a wind chart.

### 54.10.2 Constructors

This class has several constructors—the format of the `data[][][]` argument is discussed at the end of this section.

- `public DefaultWindDataset(Object[][][] data);`  
Creates a new dataset containing the specified data. Series names will be generated automatically. If `data` is `null`, this method throws a `NullPointerException`.
- `public DefaultWindDataset(String[] seriesNames, Object[][][] data);`  
Creates a new dataset containing the specified data. If `data` is `null`, this method throws a `NullPointerException`.
- `public DefaultWindDataset(List seriesKeys, Object[][][] data);`  
Creates a new dataset containing the specified data. If `data` is `null`, this method throws a `NullPointerException`.

The default constructor creates a new empty dataset—but since there are currently no methods for adding data to an existing dataset, it isn't especially useful:

- `public DefaultWindDataset();`  
Creates a new (empty) dataset. Since there are currently no methods for adding data to an existing dataset, you probably should use a different constructor.

#### The `data[][][]` Argument

Multi-dimensional arrays can be difficult to use, but unfortunately this class provides no other method of adding data to the dataset. To create a data array for the constructors above, bear in mind that the array is indexed by series, then item, then data value. Each item has three values, as follows:

- `data[series][item][0]` – this is the x-value, which can be either a `Date` or a `Number`;
- `data[series][item][1]` – this is the wind direction, and should be a `Number` in the range 0 to 12;
- `data[series][item][2]` – this is the wind force, and should be a `Number` in the range 0 to 12.

The following code creates a data array containing one series, and illustrates the structure of the array:

```
Object[] item1 = new Object[] {date1, new Integer(6), new Integer(2)};
Object[] item2 = new Object[] {date2, new Integer(2), new Integer(8)};
Object[] item3 = new Object[] {date3, new Integer(4), new Integer(7)};
Object[] item4 = new Object[] {date4, new Integer(1), new Integer(4)};
Object[] item5 = new Object[] {date5, new Integer(9), new Integer(12)};

Object[][] series1 = new Object[][] {item1, item2, item3, item4, item5};
Object[][][] data = new Object[][][] {series1};

DefaultWindDataset dataset = new DefaultWindDataset(data);
```

See the code in `WindChartDemo1.java` for a similar example.

### 54.10.3 Methods

To find the number of series in the dataset:

- `public int getSeriesCount();`  
Returns the number of series in the dataset.

To find the number of items within a series:

- `public int getItemCount(int series);`  
Returns the number of items within a series. This method throws an `IllegalArgumentException` if `series` is not in the range 0 to `getSeriesCount() - 1`.

To get the key for a series:

```
➔ public Comparable getSeriesKey(int series);
```

Returns the key for a series. The key's `toString()` method is used to create a label for the series. This method throws an `IllegalArgumentException` if `series` is not in the range 0 to `getSeriesCount() - 1`.

To access the values from the dataset:

```
➔ public Number getX(int series, int item);
```

Returns the x-value for an item within a series. Since the wind observations are typically time-based, this method will normally return a number encoded as “milliseconds since 1-Jan-1970” (the encoding used by Java's `Date` class).

```
➔ public Number getY(int series, int item);
```

This method is mapped to the `getWindForce()` method, and is implemented only because it is required by the `XYDataset` interface. It is not used by the `WindItemRenderer`.

```
➔ public Number getWindDirection(int series, int item);
```

Returns the wind direction, a number in the range 0 to 12 (corresponding to the numbers on a upside-down clock face).<sup>3</sup>

```
➔ public Number getWindForce(int series, int item);
```

Returns the wind force, on the Beaufort scale (a number in the range 0 to 12).

The following utility method is used by one of the constructors:<sup>4</sup>

```
➔ public static List seriesNameListFromDataArray(Object[][] data);
```

Returns a list containing `data.length` series names, in the form `Series 1, Series 2, ..., Series N` (where `N` is `data.length - 1`). This method is used to automatically generate series names for the constructor that doesn't have series names explicitly defined.

#### 54.10.4 Notes

Some points to note:

- although the dataset supports multiple series, the `WindItemRenderer` seems designed for just a single series;
- a demo (`WindChartDemo1.java`) is included in the JFreeChart demo collection.

### 54.11 DefaultXYDataset

#### 54.11.1 Overview

An implementation of the `XYDataset` interface where the data is stored in `double[]` arrays, which is more compact relative to alternatives that store data using `Number` instances.

*This class was first introduced in JFreeChart version 1.0.2.*

#### 54.11.2 Constructor

To create a new dataset:

```
➔ public DefaultXYDataset(); [1.0.2]
```

Creates a new dataset that is initially empty. Use the `addSeries()` method to add data items to the dataset.

---

<sup>3</sup>I don't know if this is standard for wind charts, or if the original contributed code contains a bug.

<sup>4</sup>It should not have been a public method, but it made it into the 1.0.0 API so it remains public.

### 54.11.3 Accessing Series

The dataset stores zero, one or many series of data items. Each series is represented by a `double[][]` array. The following methods provide access to the series in the dataset:

- ▶ `public int getSeriesCount(); [1.0.2]`  
Returns the number of series contained within this dataset.
- ▶ `public Comparable getSeriesKey(int series); [1.0.2]`  
Returns the key for the specified series (where `series` is in the range 0 to `getSeriesCount() - 1`). This method throws an `IllegalArgumentException` if `series` is not within the required range.
- ▶ `public int indexOf(Comparable seriesKey); [1.0.2]`  
Returns the index of the specified series, or -1 if there is no such series. If `seriesKey` is `null`, this method returns -1 (because no series can have a `null` key).
- ▶ `public int getItemCount(int series); [1.0.2]`  
Returns the number of items in the specified series.

### 54.11.4 Accessing Data Values

To find the order of the data items:

- ▶ `public DomainOrder getDomainOrder(); [1.0.2]`  
Returns `DomainOrder.NONE`, because this dataset makes no guarantees about the order of the x-values in the dataset.

To access particular values from the dataset:

- ▶ `public double getXValue(int series, int item); [1.0.2]`  
Returns the x-value for an item within a series. If `series` is not in the range 0 to `getSeriesCount() - 1`, this method will throw an `ArrayIndexOutOfBoundsException`. Similarly, the same exception type is thrown if `item` is not in the range 0 to `getItemCount(series) - 1`.
- ▶ `public Number getX(int series, int item); [1.0.2]`  
Returns the x-value for an item within a series. A new `Double` instance is created each time this method is called, so it is better to call `getXValue()` if at all possible.
- ▶ `public double getYValue(int series, int item); [1.0.2]`  
Returns the y-value for an item within a series. If `series` is not in the range 0 to `getSeriesCount() - 1`, this method will throw an `ArrayIndexOutOfBoundsException`. Similarly, the same exception type is thrown if `item` is not in the range 0 to `getItemCount(series) - 1`.
- ▶ `public Number getY(int series, int item); [1.0.2]`  
Returns the y-value for an item within a series. A new `Double` instance is created each time this method is called, so it is better to call `getYValue()` if at all possible.

### 54.11.5 Adding and Removing Data

The following methods can be used to add and remove series from the dataset:

- ▶ `public void addSeries(Comparable seriesKey, double[][] data); [1.0.2]`  
Adds a data series to the dataset (or overwrites an old series if there is already one with the specified key). The `seriesKey` should be an immutable object that is cloneable and serializable (for example, a `String` instance), otherwise the dataset will no longer be cloneable and serializable.

The `data` array should contain two subarrays, one containing the x-values (`data[0]`) and the other containing the y-values (`data[1]`). The `XYSeries` class has a `toArray()` method that creates an array in the required format.

- ▶ `public void removeSeries(Comparable seriesKey); [1.0.2]`  
Removes the series with the specified key.

### 54.11.6 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

► `public boolean equals(Object obj); [1.0.2]`

Tests this `DefaultXYDataset` for equality with an arbitrary object, returning `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `DefaultXYDataset`;
- both datasets have the same number of series, each containing exactly the same values.

Instances of this class are `Cloneable`:

► `public Object clone() throws CloneNotSupportedException; [1.0.2]`

Returns an independent copy of this dataset. The `CloneNotSupportedException` will only be thrown if the dataset contains a series key that is not cloneable.

Instances of this class are `Serializable` (provided that you use series keys that are serializable).

### 54.11.7 Notes

Some points to note:

- this class was introduced in JFreeChart version 1.0.2;
- a couple of demos (`DefaultXYDatasetDemo1-2.java`) are included in the JFreeChart demo collection;
- an alternative implementation of the `XYZDataset` interface is provided by the `XYSeriesCollection` class.

## 54.12 DefaultXYZDataset

### 54.12.1 Overview

A default implementation of the `XYZDataset` interface. This class was first introduced in JFreeChart 1.0.2.

### 54.12.2 Constructor

To create a new dataset:

► `public DefaultXYZDataset(); [1.0.2]`

Creates a new dataset, initially empty.

### 54.12.3 Methods

To find the number of series in the dataset:

► `public int getSeriesCount(); [1.0.2]`

Returns the number of series in the dataset.

To get the key for a series:

► `public Comparable getSeriesKey(int series); [1.0.2]`

Returns the key for the specified series. This method throws an `IllegalArgumentException` if `series` is not in the range 0 to `getSeriesCount() - 1`.

► `public int indexOf(Comparable seriesKey); [1.0.2]`

Returns the index of the specified series key, or -1 if there is no series in the dataset with the given key.

► public DomainOrder getDomainOrder(); [1.0.2]  
 Returns `DomainOrder.NONE` to indicate that the x-values are not guaranteed to be in any particular order.

► public int getItemCount(int series); [1.0.2]  
 Returns the number of items in the specified series. This method throws an `IllegalArgumentException` if `series` is not in the range 0 to `getSeriesCount() - 1`.

To get the data values:

► public double getXValue(int series, int item); [1.0.2]  
 Returns the x-value for an item in the specified series. This method throws an `ArrayIndexOutOfBoundsException` exception if `series` or `item` is outside the range of available data items.

► public double getYValue(int series, int item); [1.0.2]  
 Returns the y-value for an item in the specified series. This method throws an `ArrayIndexOutOfBoundsException` exception if `series` or `item` is outside the range of available data items.

► public double getZValue(int series, int item); [1.0.2]  
 Returns the z-value for an item in the specified series. This method throws an `ArrayIndexOutOfBoundsException` exception if `series` or `item` is outside the range of available data items.

Equivalent methods (required by the `XYZDataset`) interface that return `Number` objects rather than `double` primitives are:<sup>5</sup>

► public Number getX(int series, int item); [1.0.2]  
 Returns the x-value for an item in the specified series. This creates a new `Double` instance every time it is called, so use `getXValue()` if possible. This method throws an `ArrayIndexOutOfBoundsException` exception if `series` or `item` is outside the range of available data items.

► public Number getY(int series, int item); [1.0.2]  
 Returns the y-value for an item in the specified series. This creates a new `Double` instance every time it is called, so use `getYValue()` if possible. This method throws an `ArrayIndexOutOfBoundsException` exception if `series` or `item` is outside the range of available data items.

► public Number getZ(int series, int item); [1.0.2]  
 Returns the z-value for an item in the specified series. This creates a new `Double` instance every time it is called, so use `getZValue()` if possible. This method throws an `ArrayIndexOutOfBoundsException` exception if `series` or `item` is outside the range of available data items.

#### 54.12.4 Adding and Removing Series

To add a series:

► public void addSeries(Comparable seriesKey, double[][] data); [1.0.2]  
 Adds a new series to the dataset. If there is an existing series with the specified key, it will be replaced.

Removing a series:

► public void removeSeries(Comparable seriesKey); [1.0.2]  
 Removes the series with the specified key.

#### 54.12.5 Equals, Cloning and Serialization

This class overrides the `equals(Object)` method:

► public boolean equals(Object obj); [1.0.2]  
 Tests this dataset for equality with an arbitrary object, returning `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `DefaultXYZDataset`;
- both datasets contain identical series in the same order.

Instances of this class are cloneable and serializable.

<sup>5</sup>Each call to these methods creates a new `Number` instance, so you should avoid these methods if possible.

## 54.13 IntervalXYDataset

### 54.13.1 Overview

A dataset that returns an interval for each of the x and y dimensions. This interface extends the [XYDataset](#) interface. General classes that implement this interface are:

- [DefaultIntervalXYDataset](#);
- [XYIntervalSeriesCollection](#).

More specialised implementations of the interface include:

- [HistogramDataset](#);
- [SimpleHistogramDataset](#);
- [TimePeriodValuesCollection](#);
- [TimeSeriesCollection](#);
- [TimeTableXYDataset](#);
- [XYBarDataset](#).

Renderers that make use of the additional data points provided by this dataset include:

- [DeviationRenderer](#);
- [XYBarRenderer](#);
- [XYErrorRenderer](#).

### 54.13.2 Interface Methods

To get the start value of the x-interval:

```
➔ public double getStartXValue(int series, int item);
```

Returns the start value of the x-interval for an item within a series.

To get the end value of the x-interval:

```
➔ public double getEndXValue(int series, int item);
```

Returns the end value of the x-interval for an item within a series.

To get the start value of the y-interval:

```
➔ public double getStartYValue(int series, int item);
```

Returns the start value of the y-interval for an item within a series.

To get the end value of the y-interval:

```
➔ public double getEndYValue(int series, int item);
```

Returns the end value of the y-interval for an item within a series.

Similar methods that return [Number](#) objects rather than [double](#) primitives are also provided. In general, you should only call these alternate methods when you know the underlying dataset actually stores the data as [Number](#) instances, otherwise a new [Number](#) may be allocated for each call:

```
➔ public Number getStartX(int series, int item);
```

Returns the start value of the x-interval for an item within a series.

```
➔ public Number getEndX(int series, int item);
```

Returns the end value of the x-interval for an item within a series.

```
➔ public Number getStartY(int series, int item);
```

Returns the start value of the y-interval for an item within a series.

```
➔ public Number getEndY(int series, int item);
```

Returns the end value of the y-interval for an item within a series.

**See Also:**

[XYDataset](#), [IntervalXYZDataset](#).

## 54.14 IntervalXYDelegate

### 54.14.1 Overview

This class contains the logic required to “manufacture” intervals around the x-values in an [XYDataset](#), enabling a regular [XYDataset](#) to be extended to an [IntervalXYZDataset](#).

### 54.14.2 Usage

This class is used internally by the JFreeChart Class Library. In general, you won’t need to use this class directly.

### 54.14.3 Constructors

To create a new delegate:

```
→ public IntervalXYDelegate(XYDataset dataset);
Creates a new delegate with autoWidth set to true.

→ public IntervalXYDelegate(XYDataset dataset, boolean autoWidth);
Creates a new delegate that determines the x-intervals for the given dataset. The autoWidth flag controls whether or not the interval width is automatically calculated. For the automatic calculation, the width is set to the distance between the two closest x-values in the dataset.
```

### 54.14.4 Methods

The `autoWidth` flag controls whether or not the widths of the x-intervals returned by this class are automatically calculated. The default is `true`, which results in the x-interval size being equal to the gap between the nearest two x-values in the dataset:

```
→ public boolean isAutoWidth();
Returns the autoWidth flag.

→ public void setAutoWidth(boolean b);
Sets the autoWidth flag.
```

If `autoWidth` is `false`, then the interval width is controlled by the `intervalWidth` setting:

```
→ public double getIntervalWidth();
Returns the interval width.

→ public void setIntervalWidth(double w);
Sets the interval width (must be positive).
```

The `intervalPositionFactor` controls the positioning of the x-interval about its x-value. The default is 0.5 which centers the interval about the x-value:

```
→ public double getIntervalPositionFactor();
Returns the intervalPositionFactor.

→ public void setIntervalPositionFactor(double d);
Sets the intervalPositionFactor. This is a value between 0.0 and 1.0 where 0.5 is centred.

→ public Number getStartX(int series, int item);
Returns the start value for the x-interval of the specified item.

→ public Number getEndX(int series, int item);
Returns the end value for the x-interval of the specified item.
```

```
→ public double getDomainLowerBound(boolean includeInterval);
Returns the lower bound of the range of x-values. The includeInterval flag determines whether or not the x-interval is taken into account when finding the lower bound.

→ public double getDomainUpperBound(boolean includeInterval);
Returns the upper bound of the range of x-values. The includeInterval flag determines whether or not the x-interval is taken into account when finding the upper bound.

→ public Range getDomainBounds(boolean includeInterval);
Returns the range of x-values. The includeInterval flag determines whether or not the x-interval is taken into account when finding the range.
```

#### 54.14.5 Other Methods

```
→ public void itemAdded(int series, int item);
Updates the automatic width when an item is added (it seems this method is only called from the CategoryTableXYDataset class).

→ public void itemRemoved(double x);
Updates the automatic width when an item is removed (it seems this method is only called from the CategoryTableXYDataset class).

→ public void seriesAdded(int series);
Updates the width calculation when a series is added—called by the XYSeriesCollection class.

→ public void seriesRemoved();
Updates the width calculation when a series is removed—called by the XYSeriesCollection and DefaultTableXYDataset classes.
```

#### 54.14.6 Equals, Cloning and Serialization

To test this delegate for equality with an arbitrary object:

```
→ public boolean equals(Object obj);
Tests the delegate for equality with obj.

→ public Object clone() throws CloneNotSupportedException;
Returns a clone of the delegate.
```

#### 54.14.7 Notes

The class is used by the [CategoryTableXYDataset](#), [DefaultTableXYDataset](#), and [XYSeriesCollection](#) classes.

### 54.15 IntervalXYZDataset

#### 54.15.1 Overview

An extension of the [XYZDataset](#) interface, analogous to the [IntervalXYDataset](#) extension of the [XYDataset](#) interface.

#### 54.15.2 Notes

There are no classes that implement this interface at present.

### 54.16 MatrixSeries

#### 54.16.1 Overview

A [MatrixSeries](#) represents a set of ( $x$ ,  $y$ ,  $z$ ) values as a 2 by 2 matrix of z-values (the  $x$  and  $y$  values are derived from the column and row indices from the matrix). This class is used with the [MatrixSeriesCollection](#) class.

### 54.16.2 Constructors

To create a new instance:

```
► public MatrixSeries(String name, int rows, int columns);
```

Creates a new matrix with the specified number of rows and columns. All the values in the matrix are initialised to zero.

### 54.16.3 Methods

In addition to the methods inherited from the `Series` class, the following are methods are defined:

```
► public int getItemCount();
```

Returns the number of items in the matrix (the number of rows times the number of columns).

```
► public int getColumnsCount();
```

Returns the number of columns in the matrix, as defined at construction time.

```
► public int getRowCount();
```

Returns the number of rows in the matrix, as defined at construction time.

```
► public Number getItem(int itemIndex);
```

Returns the z-value at the specified index. Note that this method creates a new `Double` instance every time it is called. Another way to get the z-value is to call `get(int, int)`.

```
► public int getItemColumn(int itemIndex);
```

Returns the column index for a given item index.

```
► public int getItemRow(int itemIndex);
```

Returns the row index for a given item index.

```
► public double get(int i, int j);
```

Returns the z-value at the given position in the matrix.

```
► public void update(int i, int j, double mij);
```

Updates the z-value at the given position in the matrix, and sends a `SeriesChangeEvent` to all registered listeners.<sup>6</sup>

```
► public void zeroAll();
```

Resets all values in the matrix to zero, and sends a `SeriesChangeEvent` to all registered listeners.

### 54.16.4 Equals, Cloning and Serialization

This class overrides the equals method:

```
► public boolean equals(Object obj);
```

Tests this matrix for equality with an arbitrary object.

Instances of this class are `Cloneable` and `Serializable`.

## 54.17 MatrixSeriesCollection

### 54.17.1 Overview

An `XYZDataset` that is constructed from a collection of `MatrixSeries` instances.

### 54.17.2 Constructors

To create a new collection:

```
► public MatrixSeriesCollection();
```

Creates a new instance that is empty (contains no `MatrixSeries` instances).

```
► public MatrixSeriesCollection(MatrixSeries series);
```

Creates a new instance containing the specified `MatrixSeries`. If `series` is `null`, the collection is initialized as an empty series.

---

<sup>6</sup>Often, the only listener is a `MatrixSeriesCollection` that this `MatrixSeries` has been added to.

### 54.17.3 General Methods

In addition to the methods defined in `AbstractXYZDataset`, this class defines:

- ▶ `public int getSeriesCount();`  
Returns the number of series in the dataset.
- ▶ `public int getItemCount(int seriesIndex);`  
Returns the number of items in the specified series.

To access the matrix that represents a series:

- ▶ `public MatrixSeries getSeries(int seriesIndex);`  
Returns the matrix that represents the values for a series.
- ▶ `public Comparable getSeriesKey(int seriesIndex);`  
Returns the key (or name) for a series.

To access the data values for an item:

- ▶ `public Number getX(int seriesIndex, int itemIndex);`  
Returns the x-value for an item in a series.
- ▶ `public Number getY(int seriesIndex, int itemIndex);`  
Returns the y-value for an item in a series.
- ▶ `public Number getZ(int seriesIndex, int itemIndex);`  
Returns the z-value for an item in a series.

### 54.17.4 Adding and Removing Series

Each series in the collection is represented by a two dimensional matrix (an instance of `MatrixSeries`).

- ▶ `public void addSeries(MatrixSeries series);`  
Adds a series to the collection and sends a `DatasetChangeEvent` to all registered listeners.
- ▶ `public void removeSeries(MatrixSeries series);`  
Removes a series from the dataset and sends a `DatasetChangeEvent` to all registered listeners.
- ▶ `public void removeSeries(int seriesIndex);`  
Removes a series from the dataset and sends a `DatasetChangeEvent` to all registered listeners.
- ▶ `public void removeAllSeries();`  
Removes all series from the dataset and sends a `DatasetChangeEvent` to all registered listeners.

### 54.17.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- ▶ `public boolean equals(Object obj);`  
Tests this instance for equality with an arbitrary object.
- ▶ `public int hashCode();`  
Returns a hash code for this instance.

## 54.18 NormalizedMatrixSeries

### 54.18.1 Overview

An extension of `MatrixSeries` where the matrix values are returned as normalised values (that is, they are divided by the sum of all the values in the matrix).

### 54.18.2 Constructor

To create a new instance:

```
► public NormalizedMatrixSeries(String name, int rows, int columns);
```

Creates a new matrix with the specified number of rows and columns, with all values initially zero.

### 54.18.3 General Methods

```
► public double getScaleFactor();
```

Returns the scale factor. The default value is 1.0.

```
► public void setScaleFactor(double factor);
```

Sets the scale factor.

```
► public Number getItem(int itemIndex);
```

Returns the value in the matrix with the specified index. The value is normalised (divided by the sum of all matrix values) and then scaled.

```
► public void update(int i, int j, double mij);
```

Updates an item in the matrix, adjusting the sum of all data items accordingly.

```
► public void zeroAll();
```

Clears all items in that matrix to zero.

## 54.19 OHLCDataItem

### 54.19.1 Overview

A data item that associates several values (typically related to the trading of a financial security) with a Date:

- *open value* - the opening value at the start of the day's trading;
- *high value* - the highest value during the day's trading;
- *low value* - the lowest value during the day's trading;
- *close value* - the closing value at the end of the day's trading;
- *volume* - the trading volume (number of securities traded);

This class implements the `Comparable` interface to define a natural ordering (by date) for a collection of items.

### 54.19.2 Constructor

To create a new instance:

```
► public OHLCDataItem(Date date, double open, double high, double low, double close, double volume);
```

Creates a new data item that associates the specified values with a particular date.

### 54.19.3 Methods

To access the attributes for this data item:

```
► public Date getDate();
```

Returns the date that the values are associated with.

```
► public Number getOpen();
```

Returns the opening price for the day's trading.

```
► public Number getHigh();
Returns the highest price for the day's trading.

► public Number getLow();
Returns the lowest price for the day's trading.

► public Number getClose();
Returns the closing price for the day's trading.

► public Number getVolume();
Returns the number of securities bought/sold during the day's trading.
```

The following method is implemented as required by the `Comparable` interface, and determines a natural ordering (by date) for a collection of data items:

```
► public int compareTo(Object object);
Compares this data item to an arbitrary object, returning -1, 0 or +1 according to the relative
order of the two objects.
```

#### See Also

[OHLCdataset](#).

## 54.20 OHLCdataset

### 54.20.1 Overview

A dataset that supplies data in the form of *open-high-low-close* items. These typically relate to trading data (prices or rates) in financial markets: the open and close values represent the prices at the opening and closing of the trading period, while the high and low values represent the highest and lowest price during the trading period.

Another value returned by this dataset is the *volume*. This represents the volume of trading, and is usually the number of units of the commodity traded during a period. If this data is not available, `null` is returned.

This interface is an extension of the [XYdataset](#) interface.

### 54.20.2 Interface Methods

In addition to the methods inherited from [XYdataset](#), this interface defines:

```
► public double getHighValue(int series, int item);
Returns the high value for an item in a series.

► public double getLowValue(int series, int item);
Returns the low value for an item in a series.

► public double getOpenValue(int series, int item);
Returns the open value for an item in a series.

► public double getCloseValue(int series, int item);
Returns the close value for an item in a series.

► public double getVolumeValue(int series, int item);
Returns the trading volume for an item in a series, or Double.NaN if no trading volume is
recorded.
```

The same values can be obtained from methods that return `Number` objects rather than `double` primitives—in some cases, the underlying dataset may create a new `Number` instance for every call to one of these methods, so if possible try to avoid using these methods unless you know that the dataset does in fact store the data in object form:

```
► public Number getHigh(int series, int item);
Returns the high value for an item within a series.
```

```

→ public Number getLow(int series, int item);
Returns the low value for an item within a series.

→ public Number getOpen(int series, int item);
Returns the open value for an item within a series.

→ public Number getClose(int series, int item);
Returns the close value for an item within a series.

→ public Number getVolume(int series, int item);
Returns the volume value for an item within a series, or null if no volume is recorded.

```

### 54.20.3 Notes

Some points to note:

- this interface is implemented by several classes:
  - `DefaultHighLowDataset`;
  - `DefaultOHLCdataset`;
  - `OHLCseriesCollection`.
- this dataset is used by the `CandlestickRenderer` and `HighLowRenderer` classes.

#### See Also

[XYDataset](#).

## 54.21 TableXYDataset

### 54.21.1 Overview

This interface is an extension of the `XYDataset` interface. By implementing this interface, a dataset is declaring that all series share a common set of x-values—this is required by renderers that “stack” values (for example, the `StackedXYAreaRenderer`). Classes that implement this interface include:

- `DefaultTableXYDataset`;
- `CategoryTableXYDataset`;
- `TimeTableXYDataset`.

### 54.21.2 Interface Methods

This interface adds a single method:

```

→ public int getItemCount();
Returns the number of items in each series (all series must have the same number of items).

```

## 54.22 Vector

### 54.22.1 Overview

A vector in 2D space, used as a building block for the `VectorSeriesCollection` dataset. Instances of this class are immutable.

### 54.22.2 Constructor

To construct an instance:

```
► public Vector(double x, double y);
```

Creates a new vector.

### 54.22.3 Methods

```
► public double getX();
```

Returns the x-component for the vector.

```
► public double getY();
```

Returns the y-component for the vector.

### 54.22.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
```

Tests this vector for equality with an arbitrary object.

Instances of this class are immutable (so cloning is unnecessary) and `Serializable`.

### 54.22.5 Notes

This class is commonly used with the `VectorSeriesCollection` dataset.

## 54.23 VectorDataItem

### 54.23.1 Overview

A data item that records data in the form  $(x, y, dx, dy)$ . This class is intended for internal use by the `VectorSeries` class—you shouldn't need to use it directly.

### 54.23.2 Constructor

To construct an instance:

```
► public VectorDataItem(double x, double y, double deltaX, double deltaY);
```

Creates a new instance with the specified data values.

### 54.23.3 Methods

The following methods provide access to the data values stored by this class:

```
► public double getXValue();
```

Returns the x-value.

```
► public double getYValue();
```

Returns the y-value.

```
► public double getVectorX();
```

Returns the vector x-value.

```
► public double getVectorY();
```

Returns the vector y-value.

### 54.23.4 Equals, Cloning and Serialization

The `equals()` method from the super class is functional.

Instances of this class are `Cloneable` and `Serializable`.

### 54.23.5 Notes

This class is intended for use by the `VectorSeries` class.

## 54.24 VectorSeries

### 54.24.1 Overview

A series containing zero, one or many vectors fixed at specific (x, y) points in 2D space. A `VectorSeries` is commonly added to a `VectorSeriesCollection` to form a dataset that can be used to create vector plots (see the `VectorRenderer` class). Every series has an identifier known as the series key. This class extends the `ComparableObjectSeries` class.

### 54.24.2 Constructors

To create a new instance:

- ▶ `public VectorSeries(Comparable key);`  
Equivalent to `VectorSeries(key, false, true)`—see the next constructor.
- ▶ `public VectorSeries(Comparable key, boolean autoSort, boolean allowDuplicateXValues);`  
Creates a new series with the specified key. Initially, the series contains no data. The `autoSort` and `allowDuplicateXValues` are attributes of the super class which aren't all that useful to this class—typically you should just use the default values as specified in the previous constructor.

### 54.24.3 Adding and Removing Data Items

To add a new data item to a series:

- ▶ `public void add(double x, double y, double deltaX, double deltaY);`  
Adds the specified data item to the series and sends a `SeriesChangeEvent` to all registered listeners.
- ▶ `public ComparableObjectItem remove(int index);`  
Removes the data item at the specified index in the series. If `index` is not in the range 0 to `getItemCount()` - 1, this method throws an `IllegalArgumentException`.
- ▶ `public void clear();`  
Clears all items from the series and sends a `SeriesChangeEvent` to all registered listeners.

### 54.24.4 Data Access

To access the data values in the series:

- ▶ `public double getXValue(int index);`  
Returns the x-value for an item in the series. If `index` is not in the range 0 to `getItemCount()` - 1, this method throws an `IllegalArgumentException`.
- ▶ `public double getYValue(int index);`  
Returns the y-value for an item in the series. If `index` is not in the range 0 to `getItemCount()` - 1, this method throws an `IllegalArgumentException`.
- ▶ `public double getVectorXValue(int index);`  
Returns the vector x-value for an item in the series. If `index` is not in the range 0 to `getItemCount()` - 1, this method throws an `IllegalArgumentException`.
- ▶ `public double getVectorYValue(int index);`  
Returns the vector y-value for an item in the series. If `index` is not in the range 0 to `getItemCount()` - 1, this method throws an `IllegalArgumentException`.

The data items in the series are stored as instances of `VectorXYDataItem`—to make these accessible to the `VectorSeriesCollection` class, the following method override is provided:

- ▶ `public ComparableObjectItem getDataItem(int index);`  
Returns a data item from the series. You shouldn't normally need to call this method directly.

### 54.24.5 Equals, Cloning and Serialization

The inherited `equals()` method is sufficient to distinguish between instances.

Instances of this class are `Cloneable` and `Serializable`.

### 54.24.6 Notes

Some points to note:

- the core behaviour provided by the `ComparableObjectSeries` class is important to the implementation of this class;
- when a series is added to a `VectorSeriesCollection`, the collection registers as a series listener, and any change to the series is picked up by the series collection and reported via a `DatasetChangeEvent`.

## 54.25 VectorSeriesCollection

### 54.25.1 Overview

A dataset that implements the `VectorXYDataset` interface, by storing data in the form of zero, one or many `VectorSeries` objects.

### 54.25.2 Constructors

This class defines a single constructor:

```
➔ public VectorSeriesCollection();
Creates a new dataset, initially containing no data.
```

### 54.25.3 Series Information

This dataset can contain zero, one or many data series (instances of `VectorSeries`). To determine how many series the dataset contains:

```
➔ public int getSeriesCount();
Returns the number of series in the dataset.
```

To obtain a reference to a particular series:

```
➔ public VectorSeries getSeries(int series);
Returns one series from the dataset. If series is not in the range 0 to getSeriesCount() - 1, this method throws an IllegalArgumentException.
```

To get the key that identifies a series:

```
➔ public Comparable getSeriesKey(int series);
Returns the key for the specified series. If series is not in the range 0 to getSeriesCount() - 1, this method throws an IllegalArgumentException.
```

To find the number of data items within a series:

```
➔ public int getItemCount(int series);
Returns the number of items in the specified series. If series is not in the range 0 to getSeriesCount() - 1, this method throws an IllegalArgumentException.
```

#### 54.25.4 Adding and Removing Series

To add a series:

```
→ public void addSeries(VectorSeries series);
    Adds the specified series to the dataset. If series is null, this method throws an IllegalArgumentException.
```

To remove a series:

```
→ public boolean removeSeries(VectorSeries series);
    Removes a series from the dataset and sends a DatasetChangeEvent to all registered listeners.
    If the specified series does not belong in the dataset, this method returns false. If series is
    null, this method throws an IllegalArgumentException.
```

```
→ public void removeAllSeries();
    Removes all series from the dataset and sends a DatasetChangeEvent to all registered listeners.
```

#### 54.25.5 Data Access

The following methods provide access to the data values stored by the dataset:

```
→ public double getXValue(int series, int item);
    Returns the x-value for an item in a series.

→ public double getYValue(int series, int item);
    Returns the y-value for an item in a series.

→ public double getVectorXValue(int series, int item);
    Returns the x-component of the vector for an item in a series.

→ public double getVectorYValue(int series, int item);
    Returns the y-component of the vector for an item in a series.
```

The same data can be obtained in object form:

```
→ public Vector getVector(int series, int item);
    Returns the vector for an item in a series. This method returns the actual vector stored in the
    underlying data structure (that is, no new Vector instance is created).

→ public Number getX(int series, int item);
    Returns the x-value for an item in a series. This method creates a new Double instance every
    time it is called—if possible, use the getXValue() method instead.

→ public Number getY(int series, int item);
    Returns the y-value for an item in a series. This method creates a new Double instance every
    time it is called—if possible, use the getYValue() method instead.
```

#### 54.25.6 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj);
    Tests this dataset for equality with an arbitrary object. This method returns true if:
        • obj is an instance of VectorSeriesCollection;
        • obj contains the same data values as this dataset.
```

Instances of this class are `Cloneable` and `Serializable`.

#### 54.25.7 Notes

Some points to note:

- a demo (`VectorPlotDemo1.java`) is included in the JFreeChart demo collection.

**See Also**

[VectorSeries](#), [VectorXYDataset](#).

## 54.26 VectorXYDataset

### 54.26.1 Overview

A dataset interface used by the [VectorRenderer](#) class. This interface is an extension of the [XYDataset](#) interface.

### 54.26.2 Interface Methods

The following methods are defined by this interface, in addition to those defined in the [XYDataset](#) interface:

- `public double getVectorXValue(int series, int item);`  
Returns the vector's x-component for an item within the specified series.
- `public double getVectorYValue(int series, int item);`  
Returns the vector's y-component for an item within the specified series.
- `public Vector getVector(int series, int item);`  
Returns the vector for an item within the specified series. This method can return `null`. In some cases, classes that implement this method may need to construct a new [Vector](#) instance each time this method is called so, if possible, you should use the `getVectorXValue()` and `getVectorYValue()` methods in preference to this method.

### 54.26.3 Notes

Some points to note:

- the [VectorSeriesCollection](#) class implements this interface.

## 54.27 WindDataset

### 54.27.1 Overview

A specialised dataset interface that represents observations (through time) of the wind direction and force. This dataset extends [XYDataset](#) and is used by the [WindItemRenderer](#) class.

### 54.27.2 Methods

This interface adds the following methods to those inherited from the [XYDataset](#) interface:

- `public Number getWindDirection(int series, int item);`  
Returns the wind direction. This should be a value in the range 0 to 12, corresponding to the positions on an upside-down clock face.<sup>7</sup>
- `public Number getWindForce(int series, int item);`  
Returns the wind force. This should be a value in the range 0 to 12, defined by the Beaufort scale (see [http://en.wikipedia.org/wiki/Beaufort\\_scale](http://en.wikipedia.org/wiki/Beaufort_scale)).

### 54.27.3 Notes

Some points to note:

- this interface is implemented by [DefaultWindDataset](#);
- a demo ([WindChartDemo1.java](#)) is included in the JFreeChart demo collection.

---

<sup>7</sup>I don't know if this is the standard for wind charts, or a bug in the original code contribution.

## 54.28 XisSymbolic

### 54.28.1 Overview

An interface that can be implemented by an `XYDataset` in order to link the (integer) x-values with symbols.

### 54.28.2 Methods

The following methods are defined by the interface:

- `public String[] getXSymbolicValues();`  
Returns an array of symbols to associate with (integral) data values.
- `public String getXSymbolicValue(int series, int item);`  
Returns the symbolic x-value for an item within a series.
- `public String getXSymbolicValue(Integer val);`  
Returns the symbolic x-value associated with a specific integer value.

### 54.28.3 Notes

None of the standard datasets implement this interface.

## 54.29 XYBarDataset

### 54.29.1 Overview

A dataset wrapper class that can convert any `XYDataset` into an `IntervalXYDataset` (so that the dataset can be used with renderers that require this extended interface, such as the `XYBarRenderer` class). This class extends `AbstractIntervalXYDataset`.

### 54.29.2 Constructor

To create a new dataset wrapper:

- `public XYBarDataset(XYDataset underlying, double barWidth);`  
Creates a wrapper for the underlying dataset, effectively converting it into an `IntervalXYDataset`.

### 54.29.3 Methods

To access the underlying dataset:

- `public XYDataset getUnderlyingDataset(); [1.0.4]`  
Returns the underlying dataset (never `null`) that was set in the constructor.

To access the bar width:

- `public double getBarWidth(); [1.0.4]`  
Returns the bar width. The starting and ending x-values for this dataset are computed by fetching the x-value for the underlying dataset, then adding/subtracting half the bar width.
- `public void setBarWidth(double barWidth); [1.0.4]`  
Sets the bar width and sends a `DatasetChangeEvent` to all registered listeners.

This dataset registers itself as a listener on the underlying dataset, and receives change events via the following method:

- `public void datasetChanged(DatasetChangeEvent event);`  
Receives a change event from the underlying dataset. The dataset responds by raising its own change event.

#### 54.29.4 Dataset Methods

The following methods are implemented to support the [IntervalXYDataset](#) interface:

- ▶ `public int getSeriesCount();`  
Returns the number of series in the underlying dataset.
- ▶ `public Comparable getSeriesKey(int series);`  
Returns the key for the specified series. If `series` is not in the range 0 to `getSeriesCount()`, this method throws an exception.
- ▶ `public int getItemCount(int series);`  
Returns the number of items in the specified series. If `series` is not in the range 0 to `getSeriesCount()`, this method throws an exception.

The following methods returns the data values as `double` primitives:

- ▶ `public double getXValue(int series, int item);`  
Returns the x-value for an item in a series—this is obtained from the underlying dataset.
- ▶ `public double getYValue(int series, int item);`  
Returns the y-value for an item in a series—this is obtained from the underlying dataset.
- ▶ `public double getStartXValue(int series, int item);`  
Returns the start value in the interval about the x-value—this is computed by subtracting half the `barWidth` from the x-value obtained from the underlying dataset.
- ▶ `public double getEndXValue(int series, int item);`  
Returns the end value in the interval about the x-value—this is computed by adding half the `barWidth` to the x-value obtained from the underlying dataset.
- ▶ `public double getStartYValue(int series, int item);`  
Returns the same result as `getYValue(series, item)`.
- ▶ `public double getEndYValue(int series, int item);`  
Returns the same result as `getYValue(series, item)`.

The following methods return the data values as `Number` instances:

- ▶ `public Number getX(int series, int item);`  
Returns the x-value for an item in a series—this is obtained from the underlying dataset.
- ▶ `public Number getY(int series, int item);`  
Returns the y-value for an item in a series—this is obtained from the underlying dataset..
- ▶ `public Number getStartX(int series, int item);`  
Returns the start value in the interval about the x-value—this is computed by subtracting half the `barWidth` from the x-value obtained from the underlying dataset.
- ▶ `public Number getEndX(int series, int item);`  
Returns the end value in the interval about the x-value—this is computed by adding half the `barWidth` to the x-value obtained from the underlying dataset.
- ▶ `public Number getStartY(int series, int item);`  
Returns the same result as `getY(series, item)`.
- ▶ `public Number getEndY(int series, int item);`  
Returns the same result as `getY(series, item)`.

#### 54.29.5 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- ▶ `public boolean equals(Object obj);`  
Tests this dataset for equality with an arbitrary object. This method returns `true` if and only if:
  - `obj` is not `null`;

- `obj` is an instance of `XYBarDataset`;
- this dataset has the same `barWidth` and an underlying dataset that is equal to the corresponding attributes in `obj`.

Instances of this class are `Cloneable` and `Serializable`. Note that the underlying dataset is only deep-cloned if it implements the `PublicCloneable` interface.

#### 54.29.6 Notes

Some points to note:

- a demo (`XYBarChartDemo4.java`) is included in the JFreeChart demo collection.

### 54.30 XYCoordinate

#### 54.30.1 Overview

Represents a point (x, y) in 2D space.

#### 54.30.2 Constructors

To create a new instance:

- `public XYCoordinate();`  
Equivalent to `XYCoordinate(0.0, 0.0)`—see the next constructor.
- `public XYCoordinate(double x, double y);`  
Creates a new instance for the specified (x, y) coordinate.

#### 54.30.3 Methods

To retrieve the coordinate values:

- `public double getX();`  
Returns the x-value.
- `public double getY();`  
Returns the y-value.

This class implements `Comparable`:

- `public int compareTo(Object obj);`  
Returns an integer that can be used to sort coordinates.

The `toString()` method is overridden for debugging purposes:

- `public String toString();`  
Returns a string representing this instance—useful for debugging.

#### 54.30.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this coordinate for equality with an arbitrary object.

#### 54.30.5 Notes

This class is used internally by the `VectorDataItem` class.

## 54.31 XYDataItem

### 54.31.1 Overview

This class represents a pair  $(x, y)$  of `Number` objects. The x-value should always be defined, but the y-value can be set to `null` to represent a missing or unknown value.

### 54.31.2 Constructors

To create a new data item:

```
➔ public XYDataItem(Number x, Number y);
Creates a new data item. A null y-value is permitted (to represent a missing or unknown
value).  

➔ public XYDataItem(double x, double y);
Creates a new data item.
```

### 54.31.3 Methods

To access the x and y values:

```
➔ public Number getX();
Returns the x-value (never null).  

➔ public double getXValue(); [1.0..9]
Returns the x-value as a double primitive.  

➔ public Number getY();
Returns the y-value (possibly null).  

➔ public double getYValue(); [1.0..9]
Returns the y-value as a double primitive. If the y-value is null, this method returns Double.NaN.
```

To set the y-value:

```
➔ public void setY(Number y);
Sets the y-value (null is permitted). Note that there is no corresponding method to set the
x-value.
```

### 54.31.4 Notes

Some notes:

- this class implements the `Comparable` interface, and implements ordering by x-values.
- this class parallels the [TimeSeriesDataItem](#) class.

## 54.32 XYDataset

### 54.32.1 Overview

An interface that defines a collection of data in the form of  $(x, y)$  values. The dataset can consist of zero, one or many data series. This interface extends [SeriesDataset](#). The  $(x, y)$  values in one series are completely independent of the  $(x, y)$  value in any other series in the dataset (that is, x-values are not “shared” between series).

This is the standard dataset used by the [XYPlot](#) class, with concrete implementations provided by:

- [XYSeriesCollection](#);
- [TimeSeriesCollection](#).

Extensions of this interface include:

- [IntervalXYDataset](#);
- [OHLCDataSet](#);
- [XYZDataSet](#);
- [TableXYDataSet](#).

### 54.32.2 Number Objects vs Primitives

For a long time, `XYDataset` used only `Number` objects to represent data values. From version 0.9.19 onwards, additional methods that return the x and y values as `double` primitives have been added. These are not replacements for the existing methods, but are intended to allow for more efficient dataset implementations for specific requirements (such as large datasets for scientific data).

A number of developers have asked “why not just use `double` primitives exclusively?”. The main reasons for having the dataset interface support `Number` objects are:

- it allows `null` to be used to indicate an unknown or missing data value;
- the use of Java’s collection classes as the storage for datasets requires `Number` objects to be used anyway;
- objects can be more conveniently displayed using standard Java components such as Swing’s `JTable`.

Wherever possible, JFreeChart will call the methods that return the `double` primitives. This makes it possible to implement a dataset with primitives only (which is more space efficient for large datasets), and construct `Number` objects only when the non-primitive methods are called.

### 54.32.3 Domain Order

The “domain order” refers to the order of the x-values in the dataset. If it is known that the x-values are ordered (in, say, ascending order) a renderer might be able to speed up the drawing process. The following method provides a hint about the order of the x-values in the dataset:

► `public DomainOrder getDomainOrder();`

Returns the order of items (by x-value) in the dataset, which must be one of:

- `DomainOrder.NONE` - the items in the dataset are stored in no particular order;
- `DomainOrder.ASCENDING` - the items in the dataset are guaranteed to be stored in ascending order of x-values;
- `DomainOrder.DESCENDING` - the items in the dataset are guaranteed to be stored in descending order of x-values.

This field might be used by some renderers to speed up chart drawing in cases where the renderer can assume that the data values are ordered.

### 54.32.4 Other Methods

This interface inherits methods from [SeriesDataset](#).

To get the number of items in a series:

► `public int getItemCount(int series);`

Returns the number of data items in a series. The `series` argument should be in the range 0 to `getSeriesCount() - 1`. Classes that implement this interface should throw an `IllegalArgumentException` if `series` is not within the specified range.

To get the *x-value* for an item within a series:

```
➔ public double getXValue(int series, int item);
Returns the x-value for an item within a series.

➔ public Number getX(int series, int item);
Returns the x-value for an item within a series (never null). Some implementations will create
a new Number instance every time this method is called, so it is usually more efficient to call
getXValue(series, item) instead.
```

For both of the methods above, the `series` argument must be in the range 0 to `getSeriesCount()` - 1 and the `item` argument must be in the range 0 to `getItemCount(series)` - 1. If the arguments are not within the specified range, a runtime exception must be thrown. For the sake of efficiency (these methods get called a lot), the type of runtime exception thrown is up to the implementing class (it is typically an `ArrayIndexOutOfBoundsException` or an `IllegalArgumentException`).

To get the *y*-value for an item within a series:

```
➔ public double getYValue(int series, int item);
Returns the y-value for am item within a series. If this method returns Double.NaN, there are
two possibilities: the value is missing/unknown (equivalent to null) or the value really is "not
a number". The only way to distinguish these cases (if you need to) is to check the value
returned by the getYValue() method to see if it is null.

➔ public Number getY(int series, int item);
Returns the y-value for an item within a series (possibly null, which indicates a missing or
unknown value).
```

As for the corresponding x-value methods, for both of the methods above, the `series` argument must be in the range 0 to `getSeriesCount()` - 1 and the `item` argument must be in the range 0 to `getItemCount(series)` - 1. If the arguments are not within the specified range, a runtime exception must be thrown. For the sake of efficiency (these methods get called a lot), the type of runtime exception thrown is up to the implementing class (it is typically an `ArrayIndexOutOfBoundsException` or an `IllegalArgumentException`).

### 54.32.5 Notes

Some points to note:

- this interface extends the `SeriesDataset` interface;
- the interface allows `null` y-values (to represent missing data) but does not allow `null` x-values.<sup>8</sup>

#### See Also:

[SeriesDataset](#), [IntervalXYDataset](#).

## 54.33 XYDatasetTableModel

### 54.33.1 Overview

A simple wrapper for a `TableXYDataset` that creates a read-only implementation of Swing's `TableModel` interface.

### 54.33.2 Constructors

The default constructor creates an empty table model:

```
➔ public XYDatasetTableModel();
Creates an empty table model. If you use this constructor, you can use the setModel() method
to add a dataset later.
```

To create a new table model:

```
➔ XYDatasetTableModel(TableXYDataset dataset);
Creates a new table model for the specified dataset (null permitted).
```

---

<sup>8</sup>I could not think of a use-case that required `null` x-values.

### 54.33.3 Usage

If you look in the source code for this class, there is a `main()` method (commented out) that shows the usage for this class.

### 54.33.4 Methods

To set the dataset to be presented as a `TableModel`:

```
► public void setModel(TableXYDataset dataset);
```

Sets the underlying dataset for the table model (`null` permitted). This class will register itself as a listener for the supplied dataset, so that changes to the dataset can be passed on as corresponding table model change events.

The following method receives notification of changes to the underlying dataset, allowing the `TableModel` to forward appropriate change events:

```
► public void datasetChanged(DataSetChangeEvent datasetChangeEvent);
```

This method will be called by the underlying dataset whenever it is changed—you shouldn't need to call this method directly.

### 54.33.5 TableModel Methods

The following methods are implemented in support of the `TableModel` interface: To get the row count:

```
► public int getRowCount();
```

Returns the row count. This has been implemented as the number of items in the first series, even though other series may have a different number of items.

To get the column count:

```
► public int getColumnCount();
```

Returns the column count, which is equal to the number of series in the dataset plus 1 (the first column is used to display x-values, the remaining columns the y-values for each series).

To get the column name:

```
► public String getColumnName(int column);
```

Returns the name of a column.

To get a value for the table:

```
► public Object getValueAt(int row, int column);
```

Returns the value.

The table model is “read only”:

```
► public boolean isCellEditable(int row, int column);
```

Returns `false`.

You cannot update the dataset via the `TableModel` interface:

```
► public void setValueAt(Object value, int row, int column);
```

Does nothing, since there is no general way to update the underlying dataset.

## 54.34 XYInterval

### 54.34.1 Overview

A data record used by the `XYIntervalDataItem` class to record an x-interval, plus the associated y-value and y-interval. You should not need to use this class directly. *This class was first introduced in JFreeChart version 1.0.3.*

### 54.34.2 Constructors

To create a new instance:

```
► public XYInterval(double xLow, double xHigh, double y, double yLow, double yHigh); [1.0.3]
Creates a new instance with the specified values. No validation is performed on the interval
values.
```

### 54.34.3 Methods

The following accessor methods are provided (you should never need to use these directly):

```
► public double getXLow(); [1.0.3]
Returns the lower bound of the x-interval, as supplied to the constructor.

► public double getXHigh(); [1.0.3]
Returns the upper bound of the x-interval, as supplied to the constructor.

► public double getY(); [1.0.3]
Returns the y-value, as supplied to the constructor.

► public double getYLow(); [1.0.3]
Returns the lower bound of the y-interval, as supplied to the constructor.

► public double getYHigh(); [1.0.3]
Returns the upper bound of the y-interval, as supplied to the constructor.
```

### 54.34.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
Tests this record for equality with an arbitrary object. This methods returns true if and only
if:


- obj is not null;
- obj is an instance of XYInterval;
- obj has the same data values as this instance.

```

Instances of this class are `Serializable`. Cloning is unnecessary, because instances of this class are immutable.

### 54.34.5 Notes

This class is intended for internal use by the `XYIntervalDataItem` class.

## 54.35 XYIntervalDataItem

### 54.35.1 Overview

A data item used by the `XYIntervalSeries` class. In general, you won't need to use this class directly. *This class was first introduced in JFreeChart 1.0.3.*

### 54.35.2 Constructors

To create a new instance:

```
► public XYIntervalDataItem(double x, double xLow, double xHigh, double y, double yLow, double
yHigh); [1.0.3]
Creates a new instance with the specified attributes defining an (x, y) point plus intervals
around the x and y values. No validation is performed on the supplied values.
```

In general, you don't need to create instances of this class directly, the `XYIntervalSeries` class will take care of that.

### 54.35.3 Methods

- `public Double getX(); [1.0.3]`  
Returns the x-value, which is stored internally as a `Double`.
- `public double getXLowValue(); [1.0.3]`  
Returns the lower bound of the x-interval, as supplied to the constructor.
- `public double getXHighValue(); [1.0.3]`  
Returns the upper bound of the x-interval, as supplied to the constructor.
- `public double getYValue(); [1.0.3]`  
Returns the y-value, as supplied to the constructor.
- `public double getYLowValue(); [1.0.3]`  
Returns the lower bound of the y-interval, as supplied to the constructor.
- `public double getYHighValue(); [1.0.3]`  
Returns the upper bound of the y-interval, as supplied to the constructor.

### 54.35.4 Equals, Cloning and Serialization

The inherited `equals()` implementation can distinguish between instances of this class.

Instances of this class are `Cloneable` and `Serializable`.

### 54.35.5 Notes

It shouldn't be necessary to use this class directly—instead, rely on the API provided by the `XYIntervalSeries` class.

## 54.36 XYIntervalSeries

### 54.36.1 Overview

A series containing zero, one or many data items in the form (`x`, `xLow`, `xHigh`, `y`, `yLow`, `yHigh`). Typically you will add one or more series instances to an `XYIntervalSeriesCollection` to use as a dataset (for charts that require an `IntervalXYDataset`).

*This class was first introduced in JFreeChart version 1.0.3.*

### 54.36.2 Constructors

Two constructors are defined:

- `public XYIntervalSeries(Comparable key); [1.0.3]`  
Equivalent to `XYIntervalSeries(key, true, true)`—see the next constructor.
- `public XYIntervalSeries(Comparable key, boolean autoSort, boolean allowDuplicateXValues); [1.0.3]`  
Creates a new series, initially empty. The `key` should be non-null, immutable and `Serializable` (so that series instances can be cloned and/or serialized correctly). The `autoSort` flag determines whether or not the items added to the series are automatically sorted (by ascending x-value). The `allowDuplicateXValues` flag controls whether or not two (or more) items in the series are permitted to have the same x-value.

### 54.36.3 Methods

This class inherits many methods from the `ComparableObjectSeries` class. In addition, the following methods are defined:

- `public Number getX(int index); [1.0.3]`  
Returns the x-value for an item in the series.

```
► public double getXLowValue(int index); [1.0.5]
    Returns the lower bound of the x-interval for an item in the series.

► public double getXHighValue(int index); [1.0.5]
    Returns the upper bound of the x-interval for an item in the series.

► public double getYValue(int index); [1.0.3]
    Returns the y-value for an item in the series.

► public double getYLowValue(int index); [1.0.5]
    Returns the lower bound of the y-interval for an item in the series.

► public double getYHighValue(int index); [1.0.5]
    Returns the upper bound of the y-interval for an item in the series.
```

To add a new item to the series:

```
► public void add(double x, double xLow, double xHigh, double y, double yLow,
    double yHigh); [1.0.3]
    Adds a data item to the series.
```

You can remove items using the inherited `remove()` method.

The following method is used by JFreeChart to access items from the series:

```
► public ComparableObjectItem getDataItem(int index); [1.0.3]
    Returns a data item from the series. This method will return an instance of YIntervalDataItem,
    and is intended for internal use only.
```

#### 54.36.4 Notes

Some missing methods were added in version 1.0.5.

**See Also:**

[XYIntervalSeriesCollection](#).

### 54.37 XYIntervalSeriesCollection

#### 54.37.1 Overview

A dataset that implements the [IntervalXYDataset](#) interface. *This class was first introduced in JFreeChart version 1.0.3.*

#### 54.37.2 Constructors

To create a new empty dataset:

```
► public XYIntervalSeriesCollection(); [1.0.3]
    Creates a new dataset, initially empty.
```

#### 54.37.3 Methods

To fetch a series from the collection:

```
► public XYIntervalSeries getSeries(int series); [1.0.3]
    Returns a series from the dataset. If series is not in the range from 0 to getSeriesCount(), this
    method throws an IllegalArgumentException.
```

To add a series to the collection:

```
► public void addSeries(XYIntervalSeries series); [1.0.3]
    Adds a series to the dataset and sends a DatasetChangeEvent to all registered listeners.
```

To find out how many series are stored in the collection:

► `public int getSeriesCount(); [1.0.3]`  
 Returns the number of series in this dataset.

To get the key (unique identifier) for a series:

► `public Comparable getSeriesKey(int series); [1.0.3]`  
 Returns the key that identifies a series in the collection.

► `public int getItemCount(int series); [1.0.3]`  
 Returns the number of items in a given series.

To access the data values for an item in a series:

► `public Number getX(int series, int item); [1.0.3]`  
 Returns the x-value for an item within a series.

► `public double getStartXValue(int series, int item); [1.0.3]`  
 Returns the lower bound of the x-interval for an item within a series.

► `public double getEndXValue(int series, int item); [1.0.3]`  
 Returns the upper bound of the x-interval for an item within a series.

► `public double getYValue(int series, int item); [1.0.3]`  
 Returns the y-value for an item within a series.

► `public double getStartYValue(int series, int item); [1.0.3]`  
 Returns the lower bound of the y-interval for an item within a series.

► `public double getEndYValue(int series, int item); [1.0.3]`  
 Returns the upper bound of the y-interval for an item within a series.

The following methods return the data values as `Number` objects, but you should avoid calling them because they allocate a new object every time they are called:

► `public Number getY(int series, int item); [1.0.3]`  
 Returns the y-value for an item within a series.

► `public Number getStartX(int series, int item); [1.0.3]`  
 Returns the lower bound of the x-interval for an item within a series.

► `public Number getEndX(int series, int item); [1.0.3]`  
 Returns the upper bound of the x-interval for an item within a series.

► `public Number getStartY(int series, int item); [1.0.3]`  
 Returns the lower bound of the y-interval for an item within a series.

► `public Number getEndY(int series, int item); [1.0.3]`  
 Returns the upper bound of the y-interval for an item within a series.

#### 54.37.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

► `public boolean equals(Object obj); [1.0.3]`  
 Tests this dataset for equality with an arbitrary object. This method returns `true` if and only if:

- `obj` is not `null`;
- `obj` is an instance of `XYIntervalSeriesCollection`;
- `obj` contains series that exactly match the series in this dataset.

Instances of this class are `Cloneable`<sup>9</sup> and `Serializable`.

---

<sup>9</sup>Cloning is broken in version 1.0.4, but should be working from version 1.0.5 onwards.

### 54.37.5 Notes

Some points to note:

- see the `IntervalXYDataset` documentation for a list of renderers that can use this dataset;
- a demo (`XYErrorRendererDemo1.java`) is included in the JFreeChart demo collection.

## 54.38 XYSeries

### 54.38.1 Overview

A series containing zero, one or many  $(x, y)$  data items (extends `Series`). Each item is represented by an instance of `XYDataItem` and stored in a list (sorted in ascending order of x-values, by default). `XYSeries` will allow duplicate x-values, unless a flag is set in the constructor to prevent duplicates.

You can create a dataset (`XYDataset`) from one or more series objects by adding them to an `XYSeriesCollection` class.

### 54.38.2 Usage

In the following example, two series are created, populated and added to a collection that can be used as the dataset for a chart:

```
XYSeries series1 = new XYSeries("Series 1");
series1.add(1.0, 3.3);
series1.add(2.0, 4.4);
series1.add(3.0, 1.7);
XYSeries series2 = new XYSeries("Series 2");
series2.add(1.0, 7.3);
series2.add(2.0, 6.8);
series2.add(3.0, 9.6);
series2.add(4.0, 5.6);
XYSeriesCollection dataset = new XYSeriesCollection();
dataset.addSeries(series1);
dataset.addSeries(series2);
```

### 54.38.3 Constructors

To construct a new `XYSeries`, use one of the following constructors:

► `public XYSeries(Comparable key);`

Creates a new (empty) series with the specified name. By default, the data items will be sorted in ascending order of x-values as they are added to the series, and duplicate x-values will be permitted.

To construct a series with control over sorting and whether or not duplicate x-values are permitted:

► `public XYSeries(Comparable key, boolean autoSort);`

Creates a new (empty) series with the specified name. The `autoSort` flag controls whether or not data items will be sorted by ascending x-value as they are added to the series. Duplicate x-values will be permitted.

► `public XYSeries(Comparable key, boolean autoSort, boolean allowDuplicateXValues);`

Creates a new series (initially empty) with the specified name. Flags are set that determine whether the data items are sorted by x-value, and whether duplicate x-values will be allowed or disallowed, as specified.

The series `key` is used to identify the series—it can be any instance of `Comparable`, but is typically a `String`. For all of these constructors, if `key` is `null`, an `IllegalArgumentException` is thrown.

#### 54.38.4 Flags

The `autoSort` and `allowDuplicateXValues` flags can only be set via the constructors. There are no methods to set these flags after a series is created, but you can use the following methods to find out the flag settings:

► `public boolean getAutoSort();`

Returns a flag that indicates whether or not the items in the series are sorted (into ascending order by x-value) automatically.

► `public boolean getAllowDuplicateXValues();`

Returns a flag that indicates whether or not duplicate x-values are permitted within the series.

#### 54.38.5 Adding and Removing Items

A range of methods are provided for adding and removing data items. In most cases, a `SeriesChangeEvent` will be sent to all registered listeners, although some methods provide a `notify` flag that allows you to control this:

► `public void add(double x, double y);`

Adds a new data item to the series and sends a change event to all registered listeners.

► `public void add(double x, double y, boolean notify);`

Adds a new data item to the series and, if requested, sends a change event to all registered listeners.

► `public void add(Number x, Number y);`

Adds a new data item to the series and sends a change event to all registered listeners.

► `public void add(Number x, Number y, boolean notify);`

Adds a new data item to the series and, if requested, sends a change event to all registered listeners.

In the following two methods, an odd combination of parameters is used. This is to support the addition of `null` y-values in a sequence of calls to the previous two methods:

► `public void add(double x, Number y);`

Adds a new data item to the series and sends a change event to all registered listeners.

► `public void add(double x, Number y, boolean notify);`

Adds a new data item to the series and, if requested, sends a change event to all registered listeners.

Two further methods allow you to add the item as a single object:

► `public void add(XYDataItem item);`

Adds an item to the series and sends a change event to all registered listeners.

► `public void add(XYDataItem item, boolean notify);`

Adds an item to the series and, if requested, sends a change event to all registered listeners.

To remove an item:

► `public XYDataItem remove(int index);`

Removes an item and sends a `SeriesChangeEvent` to all registered listeners.

► `public XYDataItem remove(Number x);`

Removes an item and sends a `SeriesChangeEvent` to all registered listeners.

To delete a range of values:

► `public void delete(int start, int end);`

Deletes a range of values from the series and sends a change event to all registered listeners.

To clear all values from the series:

► `public void clear();`

Clears all values from the series and sends a change event to all registered listeners.

### 54.38.6 The Maximum Item Count

In rare circumstances, you might wish to limit the number of items that can be retained within a series. You can set a limit, and when the item limit is reached, adding a new item to the series will cause the FIRST item in the series to be removed:

► `public int getMaximumItemCount();`

Returns the maximum number of items that will be retained within the series.

► `public void setMaximumItemCount(int maximum);`

Sets the maximum number of items that will be retained within the series. When you add a new item, if it would cause the series to exceed the maximum number of items then the FIRST item in the series is removed.

### 54.38.7 Other Methods

To find out how many items are contained in a series:

► `public int getItemCount();`

Returns the number of items in the series.

To obtain a list of the items in the dataset:

► `public List getItems();`

Returns an unmodifiable list of the items in the series. Note that the list is unmodifiable, but you can still change the y-values for the individual data items in the list—this is not the recommended way to change data in the series, because no notification of the change occurs.

To update an existing data value:

► `public void update(int item, Number y);`

Changes the value of one item in the series. The `item` is a zero-based index.

► `public void update(Number x, Number y);`

Updates the y-value that is associated with `x` (which must already exist in the series, otherwise a `SeriesException` is thrown).

► `public void addOrUpdate(Number x, Number y);`

Adds a new item or updates an existing item (depending on whether or not there is already an item in the series with the given `x`-value). Note that `null` is allowed for `y`, but not for `x`.

To access a data item:

► `public XYDataItem getDataItem(int index);`

Returns an item from the series.

► `public Number getX(int index);`

Returns the x-value for an item.

► `public Number getY(int index);`

Returns the y-value for an item.

► `public int indexOf(Number x);`

Returns the index of an item that has the specified x-value.

A utility method is provided to copy the series data into an array in the format required by the `addSeries()` method in the `DefaultXYDataset` class:

► `public double[][] toArray(); [1.0.4]`

Returns an array structure containing the data values from the series. If `result` is the return value from this method, then:

- `result[0]` is an array containing the x-values for the series, so that `result[0][index]` returns an x-value from the series;
- `result[1]` is an array containing the y-values for the series, so that `result[1][index]` returns a y-value from the series.

The length of the two sub-arrays is equal to the number of items in this series.

### 54.38.8 Equality, Cloning and Serialization

This class overrides the `equals()` method:

```
► public boolean equals(Object obj);
```

Tests this series for equality with `obj`. An object is equal to this series if and only if:

- it is an instance of `XYSeries`;
- it has the same attributes as this series;
- it contains the same data items as this series.

Instances of this class are `Cloneable` and `Serializable`.

### 54.38.9 Notes

Some points to note:

- this class extends `Series`, so you can register change listeners with the series;

#### See Also:

[XYSeriesCollection](#).

## 54.39 XYSeriesCollection

### 54.39.1 Overview

A collection of `XYSeries` objects. This class implements both the `XYDataset` and `IntervalXYDataset` interfaces, so can be used as the dataset for a wide range of charts.

### 54.39.2 Constructors

To construct a series collection:

```
► public XYSeriesCollection();
Creates a new empty collection.
```

```
► public XYSeriesCollection(XYSeries series);
Creates a new collection containing a single series. If series is null, it is ignored. Additional series can be added later, if required.
```

### 54.39.3 Usage

Several demos (`XYSeriesDemo1-3.java`) are included in the JFreeChart demo collection.

### 54.39.4 Adding and Removing Series

To add a series to the collection:

```
► public void addSeries(XYSeries series);
Adds a series to the collection and sends a DatasetChangeEvent to all registered listeners.
```

To remove a series from the collection:

```
► public void removeSeries(int series);
Removes the specified series from the collection and sends a DatasetChangeEvent to all registered listeners.

► public void removeSeries(XYSeries series);
Removes the specified series from the collection and sends a DatasetChangeEvent to all registered listeners.
```

To remove all series from the collection:

```
► public void removeAllSeries();
Removes all series from the collection.
```

### 54.39.5 Using as an IntervalXYDataset

This class implements the `IntervalXYDataset` interface, which means you can (for example) use the collection as a dataset to create a bar chart (using the `XYPlot` and `XYBarRenderer` classes). The underlying data items are just points, so it is necessary to “manufacture” an x-interval for each item. The width of this interval defaults to 1.0, but can be specified with the following method:

► `public void setIntervalWidth(double width);`

Sets the width of the x-interval and sends a `DatasetChangeEvent` to all registered listeners.

Given a data item at `(2.0, 3.75)`, the default x-interval will be extend from 1.5 to 2.5 (that is, an interval of width 1.0 centered about the x-value of 2.0). You might want to change where the interval falls about the actual x-value—you can use the following method:

► `public void setIntervalPositionFactor(double factor);`

Sets the interval position factor, a value between 0.0 and 1.0 (the default is 0.5, which centers the interval about the x-value).

### 54.39.6 Other Methods

To find out how many series are held in the collection:

► `public int getSeriesCount();`

Returns the number of series in the collection.

To get a list of all series in the collection:

► `public List getSeries();`

Returns an unmodifiable list of the series in the collection.

To access a particular series:

► `public XYSeries getSeries(int series);`

Returns a series from the collection. The `series` argument is a zero-based index.

► `public XYSeries getSeries(Comparable key); [1.0.9]`

Returns a series from the collection.

To get the key for a series:

► `public Comparable getSeriesKey(int series);`

Returns the name of the specified series. The `series` argument should be in the range 0 to `getSeriesCount()` - 1, otherwise an `IllegalArgumentException` is thrown.

To get the number of items in a series:

► `public int getItemCount(int series);`

Returns the number of items in the specified series. The `series` argument should be in the range 0 to `getSeriesCount()` - 1, otherwise an `IllegalArgumentException` is thrown.

To get the x-value for an item within a series:

► `public Number getX(int series, int item);`

Returns the value of the specified item.

To get the starting value of the x-interval for an item within a series:

► `public Number getStartX(int series, int item);`

Returns the starting value of the x-interval for the specified item.

To get the ending value of the x-interval for an item within a series:

► `public Number getEndX(int series, int item);`

Returns the ending value of the x-interval for the specified item.

To get the y-value for an item within a series:

► `public Number getY(int series, int item);`

Returns the value of the specified item.

### 54.39.7 Notes

Some points to note:

- if the x-values in your dataset are time or date based, consider using the `TimeSeriesCollection` class instead;
- in JFreeChart 1.0.2, a new `DefaultXYZDataset` class has been introduced which stores data in `double[]` arrays. This class allows more compact data storage, but is not as flexible as `XYSeriesCollection`.

## 54.40 XYZDataset

### 54.40.1 Overview

An interface that defines a collection of data items in the form of (x, y, z) values. This is a natural extension of the `XYDataset` interface.

### 54.40.2 Methods

This interface adds two methods for accessing the z-value:

- ▶ `public Number getZ(int series, int item);`  
Returns the z-value, which may be `null`. Some datasets (not all) will create a new `Number` object each time this method is called—if you want to avoid this, use the `getZValue()` method instead.
- ▶ `public double getZValue(int series, int item);`  
Returns the z-value. A return value of `Double.NaN` indicates (a) a missing or unknown value, or (b) a value that is “not a number”. If you want to distinguish between these cases, you need to call the `getZ()` method and look at the result.

### 54.40.3 Notes

Some points to note:

- the `DefaultXYZDataset` class implements this interface;
- JFreeChart doesn’t have support for three dimensional charts yet, but this interface is still used by the `XYBubbleRenderer` and `XYBlockRenderer` classes.

## 54.41 YInterval

### 54.41.1 Overview

A data record used by the `YIntervalDataItem` class to record a y-value and y-interval. You should not need to use this class directly. *This class was first introduced in JFreeChart version 1.0.3.*

### 54.41.2 Constructors

To create a new instance:

- ▶ `public YInterval(double y, double yLow, double yHigh); [1.0.3]`  
Creates a new instance with the specified values. No validation is performed on the interval values.

### 54.41.3 Methods

The following accessor methods are provided (you should never need to use these directly):

- `public double getY(); [1.0.3]`  
Returns the y-value, as supplied to the constructor.
- `public double getYLow(); [1.0.3]`  
Returns the lower bound of the y-interval, as supplied to the constructor.
- `public double getYHigh(); [1.0.3]`  
Returns the upper bound of the y-interval, as supplied to the constructor.

### 54.41.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj);`  
Tests this record for equality with an arbitrary object. This methods returns `true` if and only if:
  - `obj` is not `null`;
  - `obj` is an instance of `YInterval`;
  - `obj` has the same data values as this instance.

Instances of this class are `Serializable`. Cloning is unnecessary, because instances of this class are immutable.

### 54.41.5 Notes

This class is intended for internal use by the `YIntervalDataItem` class.

## 54.42 YIntervalDataItem

### 54.42.1 Overview

A data item used by the `YIntervalSeries` class. In general, you won't need to use this class directly.  
*This class was first introduced in JFreeChart 1.0.3.*

### 54.42.2 Constructors

To create a new instance:

- `public YIntervalDataItem(double x, double y, double yLow, double yHigh); [1.0.3]`  
Creates a new instance with the specified attributes defining an (x, y) point plus intervals around the x and y values. No validation is performed on the supplied values.

In general, you don't need to create instances of this class directly, the `YIntervalSeries` class will take care of that.

### 54.42.3 Methods

The following accessor methods are defined:

- `public Double getX(); [1.0.3]`  
Returns the x-value, which is stored internally as a `Double`.
- `public double getYValue(); [1.0.3]`  
Returns the y-value, as supplied to the constructor.
- `public double getYLowValue(); [1.0.3]`  
Returns the lower bound of the y-interval, as supplied to the constructor.
- `public double getYHighValue(); [1.0.3]`  
Returns the upper bound of the y-interval, as supplied to the constructor.

#### 54.42.4 Equals, Cloning and Serialization

The inherited `equals()` implementation can distinguish between instances of this class.

Instances of this class are `Cloneable` and `Serializable`.

#### 54.42.5 Notes

It shouldn't be necessary to use this class directly—instead, rely on the API provided by the `YIntervalSeries` class.

### 54.43 YIntervalSeries

#### 54.43.1 Overview

A series containing zero, one or many data items in the form (`x`, `y`, `yLow`, `yHigh`)—that is, an interval is defined for the `y`-values, but not the `x`-values. Typically you will add one or more series instances to an `YIntervalSeriesCollection` to use as a dataset (for charts that require an `IntervalXYDataset`). *This class was first introduced in JFreeChart version 1.0.3.*

#### 54.43.2 Constructors

Two constructors are defined:

- `public YIntervalSeries(Comparable key); [1.0.3]`  
Equivalent to `YIntervalSeries(key, true, true)`—see the next constructor.
- `public YIntervalSeries(Comparable key, boolean autoSort, boolean allowDuplicateXValues); [1.0.3]`  
Creates a new series, initially empty. The `key` should be non-null, immutable and `Serializable` (so that series instances can be cloned and/or serialized correctly). The `autoSort` flag determines whether or not the items added to the series are automatically sorted (by ascending `x`-value). The `allowDuplicateXValues` flag controls whether or not two (or more) items in the series are permitted to have the same `x`-value.

#### 54.43.3 Methods

This class inherits many methods from the `ComparableObjectSeries` class. In addition, the following methods are defined:

- `public Number getX(int index); [1.0.3]`  
Returns the `x`-value for an item in the series.
- `public double getYValue(int index); [1.0.3]`  
Returns the `y`-value for an item in the series.
- `public double getYLowValue(int index); [1.0.5]`  
Returns the lower bound of the `y`-interval for an item in the series.
- `public double getYHighValue(int index); [1.0.5]`  
Returns the upper bound of the `y`-interval for an item in the series.

To add a new item to the series:

- `public void add(double x, double y, double yLow, double yHigh); [1.0.3]`  
Adds a data item to the series.

You can remove items using the inherited `remove()` method.

The following method is used by JFreeChart to access items from the series:

- `public ComparableObjectItem getDataItem(int index); [1.0.3]`  
Returns a data item from the series. This method will return an instance of `YIntervalDataItem`, and is intended for internal use only.

#### 54.43.4 Notes

Some points to note:

- some missing methods were added in version 1.0.5;
- there are some demos (`DeviationRendererDemo1.java` and `YIntervalChartDemo1.java`) that use this class included in the JFreeChart demo collection.

#### See Also:

[YIntervalSeriesCollection](#).

### 54.44 YIntervalSeriesCollection

#### 54.44.1 Overview

A dataset that implements the `IntervalXYDataset` interface, with an interval for the y-values but not the x-values. *This class was first introduced in JFreeChart version 1.0.3.*

#### 54.44.2 Constructors

To create a new empty dataset:

► `public YIntervalSeriesCollection(); [1.0.3]`  
Creates a new dataset, initially empty.

#### 54.44.3 Methods

To fetch a series from the collection:

► `public YIntervalSeries getSeries(int series); [1.0.3]`  
Returns a series from the dataset. If `series` is not in the range from 0 to `getSeriesCount()`, this method throws an `IllegalArgumentException`.

To add a series to the collection:

► `public void addSeries(YIntervalSeries series); [1.0.3]`  
Adds a series to the dataset and sends a `DatasetChangeEvent` to all registered listeners.

To find out how many series are stored in the collection:

► `public int getSeriesCount(); [1.0.3]`  
Returns the number of series in this dataset.

To get the key (unique identifier) for a series:

► `public Comparable getSeriesKey(int series); [1.0.3]`  
Returns the key that identifies a series in the collection.  
► `public int getItemCount(int series); [1.0.3]`  
Returns the number of items in a given series.

To access the data values for an item in a series:

► `public Number getX(int series, int item); [1.0.3]`  
Returns the x-value for an item within a series.  
► `public double getYValue(int series, int item); [1.0.3]`  
Returns the y-value for an item within a series.  
► `public double getStartYValue(int series, int item); [1.0.3]`  
Returns the lower bound of the y-interval for an item within a series.  
► `public double getEndYValue(int series, int item); [1.0.3]`  
Returns the upper bound of the y-interval for an item within a series.

The following methods return the data values as `Number` objects, but you should avoid calling them because they allocate a new object every time they are called:

- `public Number getY(int series, int item); [1.0.3]`  
Returns the y-value for an item within a series.
- `public Number getStartX(int series, int item);`  
Returns the same as `getX(series, item)`, as this dataset does not record an interval about the x-values.
- `public Number getEndX(int series, int item);`  
Returns the same as `getX(series, item)`, as this dataset does not record an interval about the x-values.
- `public Number getStartY(int series, int item); [1.0.3]`  
Returns the lower bound of the y-interval for an item within a series.
- `public Number getEndY(int series, int item); [1.0.3]`  
Returns the upper bound of the y-interval for an item within a series.

#### 54.44.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

- `public boolean equals(Object obj); [1.0.3]`  
Tests this dataset for equality with an arbitrary object. This method returns `true` if and only if:
  - `obj` is not `null`;
  - `obj` is an instance of `YIntervalSeriesCollection`;
  - `obj` contains series that exactly match the series in this dataset.

Instances of this class are `Cloneable`<sup>10</sup> and `Serializable`.

#### 54.44.5 Notes

Some points to note:

- see the `IntervalXYDataset` documentation for a list of renderers that can use this dataset;
- a demo (`XYErrorRendererDemo1.java`) is included in the JFreeChart demo collection.

### 54.45 YisSymbolic

#### 54.45.1 Overview

An interface that can be implemented by an `XYDataset` in order to link the (integer) y-values with symbols.

#### 54.45.2 Methods

The following methods are defined by the interface:

- `public String[] getYSymbolicValues();`  
Returns an array of symbols to associate with (integral) data values.
- `public String getYSymbolicValue(int series, int item);`  
Returns the symbolic y-value for an item within a series.
- `public String getYSymbolicValue(Integer val);`  
Returns the symbolic y-value associated with a specific integer value.

---

<sup>10</sup>Cloning is broken in version 1.0.4, but should be working from version 1.0.5 onwards.

### 54.45.3 Notes

None of the standard datasets implement this interface.

# Appendix A

## Migration

### A.1 Introduction

This section includes notes on migrating to JFreeChart 1.0.9 from earlier versions of the library. In principle, all releases in the version 1.0.x series are backwards compatible with earlier releases in the series. If you experience any trouble migrating up through versions in the 1.0.x series, please report the problems so that they can be fixed and/or documented in this section.

### A.2 1.0.8 to 1.0.9

#### Overview

This release contains an important security fix in the code that generates HTML image maps—if you are using HTML image maps, you are strongly encouraged to upgrade to this release. For more information about the security issue, please refer to the following report:

<http://www.rapid7.com/advisories/R7-0031.jsp>

In other respects, this release should be a straight-forward upgrade from JFreeChart 1.0.8.

#### New Methods

New methods in this release:

- `HashUtilities` – new methods for computing hash codes for `BooleanList`, `PaintList` and `StrokeList`;
- `ImageMapUtilities` – added `htmlEscape()` method;
- `IntervalMarker` – added a new constructor;
- `Range` – added `intersects(Range)` and `scale()` methods;
- `XYDataItem` – added `getXValue()` and `getYValue()` methods;
- `XYPlot` – added `setFixedDomainAxisSpace()` and `setFixedRangeAxisSpace()` methods;
- `XYSeriesCollection` – added `getSeries(Comparable)` method.

## A.3 1.0.7 to 1.0.8

### Overview

This release should be a straight-forward upgrade from 1.0.7. Some changes have been made to the `PiePlot` class to fix bugs in the layout of the labels...if your application displays pie charts, you should check them. In particular, if you set custom values for the `interiorGap`, `labelGap`, `labelLinkMargin` or `maximumLabelWidth` attributes, you may find that these result in more white space around the chart than previously. The reason is that bugs in the labelling code resulted in the specified dimensions being halved.

### New Methods

New methods in this release:

- `DialPointer.Pointer` – added `fillPaint` and `outlinePaint` attributes;
- `StandardDialScale` – added accessor methods for bounds and minor tick stroke attributes;
- `StatisticalBarRenderer` – added `get/setErrorIndicatorStroke()` methods;

### Deprecations

In this release, the following deprecations have been made:

- `StandardXYItemRenderer` – deprecated the `shapesFilled` override attribute. You should avoid using this override flag, which will be removed in a future release, and simply rely on the per-series and default settings;

## A.4 1.0.6 to 1.0.7

### Overview

This release should be a straight-forward upgrade from 1.0.6, with the majority of changes being additions to the API. There is, however, one non-compatible API change to the `Zoomable` interface which now has several new methods.

### New Classes

New classes in this release include:

- a new package `org.jfree.chart.plot.dial.*`, containing the `DialPlot` class and related classes that were previously in the experimental source tree;
- a new `LogAxis` class (previously in experimental) that provides an alternative to the existing `LogarithmicAxis` class;
- a new `PlotUtilities` class;
- the `TickType` class defines tokens to represent major and minor ticks;

### New Methods

New methods include:

- `CategoryAxis` – added `getCategorySeriesMiddle()` method which is used to offset items in different series within a category;
- `CategoryPlot` – added `getRangeAxisIndex()`, `zoomDomainAxes()` and `zoomRangeAxes()` methods;

- `ChartPanel` – added a `defaultSaveAsDirectory` attribute with accessor methods;
- `FastScatterPlot` – added new zooming methods;
- `LineAndShapeRenderer` – added `useSeriesOffset` and `itemMargin` attributes;
- `NumberTick` – a new constructor that allows the tick type to be specified;
- `NumberTickUnit` – added a new constructor to set the `minorTickCount` attribute;
- `PiePlot` – new attributes/methods to support simple labelling as an alternative;
- `PolarPlot` – added new zooming methods;
- `StandardPieSectionLabelGenerator` – added new constructors that accept a `Locale` argument;
- `StandardPieToolTipGenerator` – added new constructors that accept a `Locale` argument;
- `SymbolAxis` – added a new `alternateGridBandPaint` attribute with accessor methods;
- `TickUnit` – added a `minorTickCount` attribute and a new constructor to set the value of this attribute;
- `ThermometerPlot` – added new zooming methods;
- `XYPlot` – added new zooming methods;

### Deprecations

The following deprecations have been made:

- `LineAndShapeRenderer` – the `linesVisible`, `shapesVisible` and `shapesFilled` override attributes have been deprecated. You should just use the per-series and base (default) settings;
- `ThermometerPlot` – various constants for the thermometer dimensions have been deprecated and replaced by variable attributes;
- `Week` – deprecated a constructor;
- `XYLineAndShapeRenderer` – the `linesVisible`, `shapesVisible` and `shapesFilled` override attributes have been deprecated. You should just use the per-series and base (default) settings;

## A.5 1.0.5 to 1.0.6

In version 1.0.6, some method additions in the `DrawingSupplier` interface (related to the fill paint attribute) may cause trouble for developers that use their own custom drawing supplier implementation. These methods have been added to the `DefaultDrawingSupplier` implementation.

A large number of deprecations have been made in the `CategoryItemRenderer` and `XYItemRenderer` interfaces, and the `AbstractRenderer` class. The override flags for most renderer attributes have been deprecated as they are essentially redundant.

The following classes have new methods:

- `AbstractRenderer` – new “lookup” methods for core attributes, that refer to new “auto-populate” flags to determine whether or not to use the current drawing supplier to auto-populate the per-series settings;
- `CategoryItemEntity` – various new methods for addition of `rowKey` and `columnKey` attributes;
- `DefaultKeyedValues` – new `insertValues()` methods;

- `DefaultPieDataset` – new `insertValues()` methods;
- `LegendItemEntity` – added `get/setDataset()` and `get/setSeriesKey()` methods;
- `LookupPaintScale` – a new `add()` method that takes a double primitive;
- `PiePlot` – new methods to get/set the pie label distributor;
- `Plot` – new `drawBackground()` method to handle `GradientPaint`, and new `outlineVisible` flag with associated accessor methods;
- `QuarterDateFormat` – added new `GREEK_QUARTERS` field;
- `SimpleHistogramDataset` – new `clearObservations()` and `removeAllBins()` methods;
- `TimeSeriesCollection` – new `indexOf()` method;
- `XYSeriesCollection` – new `indexOf()` method.

New classes in this release:

- `AbstractPieLabelDistributor` – a new base class for a plugin object used by the `PiePlot` class to distribute labels;
- `HexNumberFormat` – a new custom number formatter;
- `URLUtilities` – a utility class that provides access (via reflection) to the `URLEncoder.encode(String, String)` method introduced in JDK1.4, while maintaining JFreeChart's support for JDK1.3.1.

The `VectorRenderer` and associated dataset classes have been moved from the experimental tree into the main JFreeChart API.

## A.6 1.0.4 to 1.0.5

New classes in this release:

- `BlockFrame` – a new interface for borders that can be assigned to any `AbstractBlock`;
- `DeviationRenderer` – a new renderer for the `XYPlot` class.
- `LineBorder` – a new border class.

The following classes have new methods:

- `AbstractBlock` – added `get/setFrame()` methods;
- `AbstractCategoryItemRenderer` – added `createState()` protected method;
- `CategoryPlot` – added `setDomainAxisLocation()` variant;
- `CandlestickRenderer` – added `getDrawVolume()`;
- `ColorBlock` – added `getPaint()`;
- `DatasetUtilities` – added `calculateStackTotal()`;
- `StackedXYBarRenderer` – added `get/setRenderAsPercentages()` methods;
- `ValueAxis` – added `get/setDefaultAutoRange()` methods;
- `XYIntervalSeries` – added `getXLowValue()`, `getXHighValue()`, `getYLowValue()` and `getYHighValue()`;
- `XYPlot` – added `setDomainAxisLocation()` and `setRangeAxisLocation()` variants;

- [YIntervalSeries](#) – added `getYLowValue()` and `getYHighValue()` methods.

Other changes include:

- [BlockBorder](#) implements the new [BlockFrame](#) interface;

Deprecations include:

- [AbstractBlock](#) – deprecated the `get/setBorder()` methods, use the new `get/setFrame()` methods instead. This change has been made to allow for new border types without having to subclass [BlockBorder](#).

## A.7 1.0.3 to 1.0.4

There are some significant additions to the API in this release, as well as a number of deprecations. However, it is not expected that these changes will break existing code, and the upgrade from 1.0.3 to 1.0.4 should be straight forward.

New classes in this release:

- [GrayPaintScale](#) – a [PaintScale](#) implementation that returns shades of gray for values in a range;
- [LookupPaintScale](#) – a [PaintScale](#) implementation based on a lookup table;
- [OHLC](#) – a data record for an open-high-low-close item;
- [OHLCItem](#) – represents one item in a [OHLCSeries](#);
- [OHLCSeries](#) – a series of open-high-low-close observations;
- [OHLCSeriesCollection](#) – a new dataset implementation for creating open-high-low-close style charts;
- [PaintScale](#) – an interface that defines the API for converting values in a range into a corresponding [Paint](#) instance;
- [PaintScaleLegend](#) – a chart legend that displays the range of colors for a [PaintScale](#);
- [XYBlockRenderer](#) – a new renderer for the [XYPlot](#) class;

Other additions to the API include:

- [AbstractXYItemLabelGenerator](#): added a new constructor;
- [ChartFactory](#): added `createBoxAndWhiskerChart()` method;
- [DateAxis](#): added `get/setTimeZone()` methods;
- [LegendItem](#): added `fillPaintTransformer` attribute, plus accessor methods;
- [PiePlot](#): added `legendLabelURLGenerator` attribute, plus accessor methods;
- [StandardXYItemLabelGenerator](#): added a new constructor;
- [StandardXYToolTipGenerator](#): added a new constructor;
- [XYBarDataset](#): added `getUnderlyingDataset()` method, and `get/setBarWidth()` methods;
- [XYDifferenceRenderer](#): added `roundXCoordinates` attribute, plus accessor methods;
- [XYImageAnnotation](#): added anchor attribute and several new methods;

- `XYSeries`: added `toArray()` method.

Deprecations include:

- all the classes relating to the `ContourPlot` class have been deprecated in this release—for this type of chart, use the `XYPlot` class and `XYBlockRenderer` instead. The deprecated classes/interfaces are:

- `ClipPath`;
- `ColorBar`;
- `ColorPalette`;
- `ContourDataset`;
- `ContourPlot`;
- `ContourPlotUtilities`;
- `ContourValuePlot`;
- `DefaultContourDataset`;
- `GreyPalette`;
- `NonGridContourDataset`;
- `PaletteChooserPanel`;
- `PaletteSample`;
- `RainbowPalette`.

## A.8 1.0.2 to 1.0.3

The following new classes have been added in this release:

- `DefaultIntervalXYDataset` – a new dataset implementation;
- `MarkerChangeEvent` – a new event class;
- `MarkerChangeListener` – a new listener interface;
- `XIntervalSeriesCollection` – a new dataset implementation;
- `XYErrorRenderer` – a new renderer;
- `XYIntervalSeriesCollection` – a new dataset implementation;
- `YIntervalSeriesCollection` – a new dataset implementation;

The following classes have new methods:

- `AreaRenderer` – new `equals()` override;
- `ChartPanel` – added new public method `doEditChartProperties()`;
- `CrosshairState` – several new methods have been added;
- `LegendItemBlockContainer` – added support for tooltips and URLs;
- `RingPlot` – added methods to get/set the depth of the ring;
- `StackedAreaRenderer` – added a flag to allow the renderer to display values as percentages;

The following have been deprecated:

- **JDBCXYDataset** – the methods `getLegendItemCount()` and `getLegendItemLabels()` have been deprecated;
- **PiePlot** – attributes that relate to pie sections are now stored with the section key rather than the section index, to allow for reordering of the underlying dataset. Most methods that specify a section index are now replaced by corresponding methods that specify a section key.

Other notes:

- in previous versions of JFreeChart, the rendering of domain and range markers was incomplete—the alpha transparency of the marker was ignored, as were the outline paint and stroke. This has been fixed, and may change the appearance of charts that use markers.

## A.9 1.0.1 to 1.0.2

This version should be a drop-in replacement for 1.0.1. The following is a summary of the API additions and deprecations:

- **DrawableLegendItem** – this class has been deprecated as it is not used anywhere by JFreeChart;
- **CategoryToPieDataset** – methods have been added for accessing the underlying dataset;
- **DefaultXYDataset** – a new implementation of the **XYDataset** interface;
- **DefaultXYZDataset** – a new implementation of the **XYZDataset** interface;
- **LegendItemBlockContainer** – a new class used internally by JFreeChart;
- **MultiplePiePlot** – new fields `aggregatedItemsKey` and `aggregatedItemsPaint`, plus accessor methods;
- **SpiderWebPlot** – added new fields `toolTipGenerator` and `urlGenerator`, plus accessor methods;
- **StackedBarRenderer3D** – added a new flag (`renderAsPercentages`), plus accessor methods, that controls whether the data items are displayed as values or percentages. Two new constructors are also added.
- **XYPolygonAnnotation** – added new accessor methods:
  - `getPolygonCoordinates()`;
  - `getFillPaint()`;
  - `getOutlineStroke()`;
  - `getOutlinePaint()`.

Other changes include:

- in the **PiePlot** class, the default section label format has changed to simply display the section key (and not the value any longer). If you want to change the format of the labels, see section [33.27.11](#).

## A.10 1.0.0 to 1.0.1

Some minor adjustments have been made to the API:

- **BarRenderer** – introduced a new flag (`includeBaseInRange`), with corresponding accessor methods, to control whether or not the base value (typically zero, but user-definable) is included in the value range calculated by the renderer;

- `LevelRenderer` – for consistency with method names in other renderers, deprecated the `getMaxItemWidth()` method and added a new method `getMaximumItemWidth()`. Likewise for `setMaxItemWidth()` and `setMaximumItemWidth()`;
- `Range` – added a new method `expandToInclude(Range, double)` for convenience;
- `TaskSeriesCollection` – added new methods `getSeries(int)` and `getSeries(Comparable)`. Without these, it is not possible to retrieve a `TaskSeries` that has been added to the collection.
- `TimeSeriesCollection` – the `domainIsPointsInTime` flag has been deprecated, because it is redundant. If you get a deprecation warning for code that sets this flag, you should be able to simply remove the code;
- `XYSeries` – the `update(int, Number)` method has been deprecated and replaced by the otherwise equivalent method `updateByIndex(int, Number)`. This is to avoid confusion with the other `update()` method in this class;

## A.11 0.9.x to 1.0.0

Prior to version 1.0.0 being released (on 2-Dec-2005), the API was clearly marked as being subject to change. The changes up to version 1.0.0 are not documented. Try searching the forum at [jfree.org](http://jfree.org) for hints.

# Appendix B

## JCommon

### B.1 Introduction

JFreeChart makes use of classes in the JCommon class library. The JCommon runtime jar file is included in the JFreeChart distribution. If you require the source code and/or documentation, you can download these from:

<http://www.jfree.org/jcommon/>

Selected JCommon classes are documented here because they are used extensively within JFreeChart. For further information, refer to the JCommon Javadoc API documentation.

### B.2 Align

#### B.2.1 Overview

This class is used to align a rectangle with another rectangle (the “reference frame”). Alignment codes are defined that control how the alignment is performed.

Code:	Description:
Align.CENTER	Centers the rectangle within (or over) the reference frame.
Align.TOP	Aligns the top edge of the rectangle with the top edge of the reference frame.
Align.BOTTOM	Aligns the bottom edge of the rectangle with the bottom edge of the reference frame.
Align.LEFT	Aligns the left edge of the rectangle with the left edge of the reference frame.
Align.RIGHT	Aligns the right edge of the rectangle with the right edge of the reference frame.

Table B.1: Alignment codes

#### B.2.2 Methods

This class defines a single (static) method:

```
► public static void align(Rectangle2D rect, Rectangle2D frame, int align);  
Aligns the rect with the frame according to the specified alignment code. An exception will be  
thrown if either rect or frame is null.
```

## B.3 GradientPaintTransformer

### B.3.1 Overview

An interface that provides a method for transforming the fixed points of a `GradientPaint` instance to match some arbitrary shape. For example, in a bar chart, you will typically want the gradient paint to cover the length of each bar. JFreeChart classes that make use of this interface include:

- `BarRenderer` – see section [36.4.6](#);
- `XYBarRenderer` – see section [37.16.4](#);
- `IntervalMarker` – see section [33.20.5](#).

### B.3.2 Interface Methods

This interface defines a single method:

```
→ public GradientPaint transform(GradientPaint paint, Shape target);
```

Returns a `GradientPaint` that has been transformed in some way to match the target shape. Classes that implement this method will typically return a new `GradientPaint` instance each time this method is called. Callers should ensure that both `paint` and `target` are not `null`.

### B.3.3 Notes

JCommon provides the `StandardGradientPaintTransformer` class, which implements this interface.

## B.4 GradientPaintTransformType

### B.4.1 Overview

An enumeration used by the `StandardGradientPaintTransformer` class:

- `HORIZONTAL` – a horizontal gradient with color 1 on the left and color 2 on the right;
- `VERTICAL` – a vertical gradient with color 1 at the top and color 2 at the bottom;
- `CENTER_HORIZONTAL` – a cyclic gradient with color 1 in the middle and color 0 at the left and right edges;
- `CENTER_VERTICAL` – a cyclic gradient with color 1 in the middle and color 0 at the top and bottom edges.

### B.4.2 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
→ public boolean equals(Object obj);
```

Tests this instance for equality with an arbitrary object.

Instances of this class are `Serializable` but not `Cloneable` (the predefined instances are immutable).

## B.5 PublicCloneable

### B.5.1 Overview

An interface for objects with a `clone()` method. This is used in JFreeChart to “look behind” an interface to see if the class implementing the interface can be cloned.

### B.5.2 Methods

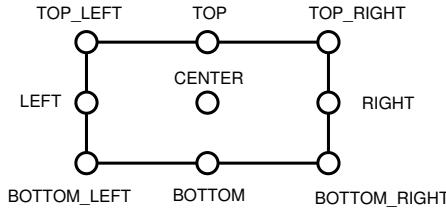
This interface declares a single method:

```
↳ public Object clone() throws CloneNotSupportedException;
Creates a clone of the object.
```

## B.6 RectangleAnchor

### B.6.1 Overview

This class defines an enumeration of nine common anchor points within a rectangle—see figure B.1.



*Figure B.1: Rectangle anchor points*

The tokens defined to represent these points are listed in table B.2.

ID:	Description:
RectangleAnchor.TOP	The midpoint of the rectangle's top edge.
RectangleAnchor.BOTTOM	The midpoint of the rectangle's bottom edge.
RectangleAnchor.LEFT	The midpoint of the rectangle's left edge.
RectangleAnchor.RIGHT	The midpoint of the rectangle's right edge.
RectangleAnchor.TOP_LEFT	The top-left corner of the rectangle.
RectangleAnchor.TOP_RIGHT	The top-right corner of the rectangle.
RectangleAnchor.BOTTOM_LEFT	The bottom-left corner of the rectangle.
RectangleAnchor.BOTTOM_RIGHT	The bottom-right corner of the rectangle.
RectangleAnchor.CENTER	The center of the rectangle.

*Table B.2: Constants defined by RectangleAnchor*

## B.7 RectangleEdge

### B.7.1 Overview

This class defines an enumeration of the four edges of a rectangle—the elements are listed in table B.3. It is used to specify the location of objects (for example, axes in a plot) relative to a rectangle.

## B.8 RectangleInsets

### B.8.1 Overview

This class is used to specify left, right, top and bottom insets relative to an arbitrary rectangle. The space can be specified in absolute terms (points, or 1/72 inch) or relative terms (a percentage of the height or width of the rectangle).

ID:	Description:
RectangleEdge.TOP	The top edge.
RectangleEdge.BOTTOM	The bottom edge.
RectangleEdge.LEFT	The left edge.
RectangleEdge.RIGHT	The right edge.

Table B.3: Constants defined by *RectangleEdge*

### B.8.2 Constructor

To create a new instance:

- `public RectangleInsets(double top, double left, double bottom, double right);`  
Creates a new instance with the given insets as absolute units.
- `public RectangleInsets(UnitType unitType, double top, double left, double bottom, double right);`  
Creates a new instance with the given insets. The values are interpreted as points (1/72 inch) for absolute spacing, or percentages for relative spacing.

### B.8.3 Accessor Methods

The following methods provide access to the attributes of an instance:

- `public UnitType getUnitType();`  
Returns the unit type (relative or absolute) for the insets.
- `public double getTop();`  
Returns the top insets value—this may be a relative or absolute value, depending on the unit type (see `getUnitType()`).
- `public double getBottom();`  
Returns the bottom insets value—this may be a relative or absolute value, depending on the unit type (see `getUnitType()`).
- `public double getLeft();`  
Returns the left insets value—this may be a relative or absolute value, depending on the unit type (see `getUnitType()`).
- `public double getRight();`  
Returns the right insets value—this may be a relative or absolute value, depending on the unit type (see `getUnitType()`).

### B.8.4 Calculation Methods

These methods are used to apply the insets to areas in various ways:

- `public Rectangle2D createAdjustedRectangle(Rectangle2D base, LengthAdjustmentType horizontal, LengthAdjustmentType vertical);`  
A general method that contracts or expands the width and height of the `base` area, as requested.
- `public Rectangle2D createInsetRectangle(Rectangle2D base);`  
Applies the insets to `base` and returns a (smaller) rectangle.
- `public Rectangle2D createInsetRectangle(Rectangle2D base, boolean horizontal, boolean vertical);`  
Applies the insets (as requested) to `base` and returns a (smaller) rectangle.
- `public Rectangle2D createOutsetRectangle(Rectangle2D base);`  
Applies the insets to `base` and returns a (larger) rectangle. This method works as the inverse to `createInsetRectangle()`.
- `public Rectangle2D createOutsetRectangle(Rectangle2D base, boolean horizontal, boolean vertical);`  
Applies the insets (as requested) to `base` and returns a (smaller) rectangle.

```

→ public double calculateTopInset(final double height);
Returns the top “inset” amount calculated relative to the given height.

→ public double calculateTopOutset(final double height);
Returns the top “outset” amount calculated relative to the given height.

→ public double calculateBottomInset(final double height);
Returns the bottom “inset” amount calculated relative to the given height.

→ public double calculateBottomOutset(final double height);
Returns the bottom “outset” amount calculated relative to the given height.

→ public double calculateLeftInset(final double width);
Returns the left “inset” amount calculated relative to the given width.

→ public double calculateLeftOutset(final double width);
Returns the left “outset” amount calculated relative to the given width.

→ public double calculateRightInset(final double width);
Returns the right “inset” amount calculated relative to the given width.

→ public double calculateRightOutset(final double width);
Returns the right “outset” amount calculated relative to the given width.

→ public double trimWidth(double width);
Returns width minus the left and right insets.

→ public double trimHeight(double height);
Returns height minus the top and bottom insets.

→ public void trim(Rectangle2D area);
Trims the insets from the given area. Note that this overwrites the contents of area.

→ public double extendWidth(double width);
Returns width plus the left and right “outsets”. This method provides the inverse operation to trimWidth().

→ public double extendHeight(double height);
Returns the height plus the top and bottom “outsets”. This method provides the inverse operation to trimHeight().

```

### B.8.5 Equality, Cloning and Serialization

This class overrides `equals()`:

```

→ public boolean equals(Object obj);
Tests this instance for equality with an arbitrary object. Returns true if and only if:


- the obj is an instance of RectangleInsets;
- both objects have the same insets and unit types.

```

To get a hash code for an instance:

```

→ public int hashCode();
Returns a hash code for this instance.

```

This class is cloneable and serializable.

## B.9 StandardGradientPaintTransformer

### B.9.1 Overview

A standard transformer used to fit a `GradientPaint` (with fixed anchor points) to some arbitrary shape (in fact, the transformation is relative to the bounding rectangle of the specified shape). The transformation in fact returns a new `GradientPaint` instance with modified anchor points.

A demo (`GradientPaintTransformerDemo1.java`) shows the effects that can be achieved with this transformer—see figure B.2.

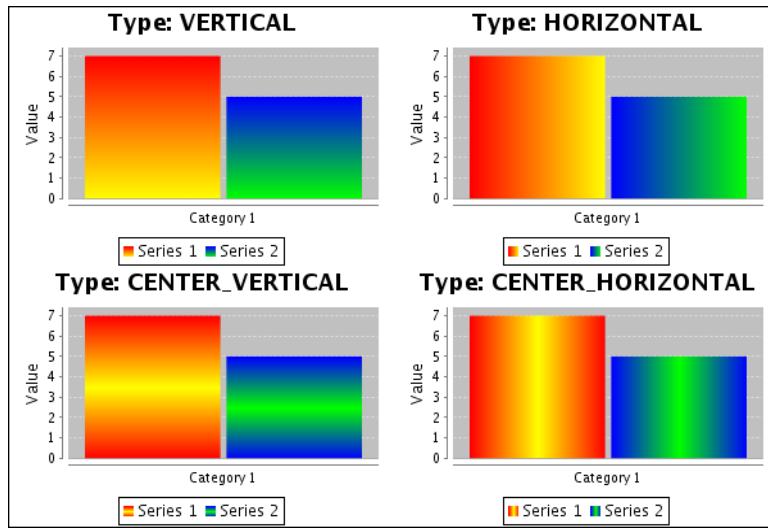


Figure B.2: Output from `GradientPaintTransformerDemo1.java`

### B.9.2 Constructors

This class defines two constructors:

```
→ public StandardGradientPaintTransformer();
Equivalent to StandardGradientPaintTransformer(GradientPaintTransformType.VERTICAL)—see the
next constructor.
```

```
→ public StandardGradientPaintTransformer(GradientPaintTransformType type);
Creates a transformer with the specified type. If type is null, this constructor throws an
IllegalArgumentException.
```

### B.9.3 Methods

To determine the transform type:

```
→ public GradientPaintTransformType getType(); [1.0.10]
Returns the transform type (never null) which is specified via the constructors (instances of
this class are immutable, so there is no method to change the type).
```

To transform a gradient paint instance to match an arbitrary shape:

```
→ public GradientPaint transform(GradientPaint paint, Shape target);
Returns a new GradientPaint instance that fits the bounding rectangle of the specified target,
according to the transform type;
```

- HORIZONTAL – the transformed gradient positions color 1 at the left and color 2 at the right;
- VERTICAL – a vertical gradient with color 1 at the top of the bounding box and color 2 at the bottom;
- CENTER\_HORIZONTAL – a cyclic horizontal gradient with color 1 in the middle of the bounding
 box and color 2 at the left and right edges;
- CENTER\_VERTICAL – a cyclic vertical gradient with color 1 in the middle of the bounding
 box and color 2 at the top and bottom edges.

If `paint` or `target` is null, this method will throw a `NullPointerException`.

### B.9.4 Equals, Cloning and Serialization

This class overrides the `equals()` method:

```
➔ public boolean equals(Object obj);
    Tests this transformer for equality with an arbitrary object (which may be null).
```

Instances of this class are `Cloneable`<sup>1</sup> and `Serializable`.

#### See Also

[GradientPaintTransformer](#).

## B.10 TextAnchor

### B.10.1 Overview

This class defines an enumeration of the anchor points relative to the bounds of a text string (see table B.4). It is used to specify an anchor point for text alignment and rotation.

ID:	Description:
<code>TextAnchor.TOP_LEFT</code>	The top left corner.
<code>TextAnchor.TOP_CENTER</code>	The center point on the top edge.
<code>TextAnchor.TOP_RIGHT</code>	The top right corner.
<code>TextAnchor.CENTER_LEFT</code>	The center point on the left edge.
<code>TextAnchor.CENTER</code>	The center point of the text.
<code>TextAnchor.CENTER_RIGHT</code>	The center point on the right edge.
<code>TextAnchor.HALF_ASCENT_LEFT</code>	The half ascent point on the left edge.
<code>TextAnchor.HALF_ASCENT_CENTER</code>	The center point along the half ascent line.
<code>TextAnchor.HALF_ASCENT_RIGHT</code>	The half ascent point on the right edge.
<code>TextAnchor.BASELINE_LEFT</code>	The baseline point on the left edge.
<code>TextAnchor.BASELINE_CENTER</code>	The center point along the half ascent line.
<code>TextAnchor.BASELINE_RIGHT</code>	The baseline point on the right edge.
<code>TextAnchor.BOTTOM_LEFT</code>	The bottom left corner.
<code>TextAnchor.BOTTOM_CENTER</code>	The center point on the bottom edge.
<code>TextAnchor.BOTTOM_RIGHT</code>	The bottom right corner.

Table B.4: Constants defined by `TextAnchor`

### B.10.2 Notes

To see how these anchor values affect the alignment of text, try running the demo application included with JCommon:

```
org.jfree.demo.DrawStringDemo
```

## B.11 UnitType

### B.11.1 Overview

This class defines tokens to indicate “relative” or “absolute” measurement units—see table B.5. These tokens are used by the `RectangleInsets` class.

---

<sup>1</sup>This is not strictly necessary, since instances of this class are immutable.

ID:	Description:
UnitType.ABSOLUTE	Absolute units.
UnitType.RELATIVE	Relative units.

Table B.5: Constants defined by *UnitType*

## Appendix C

# Configuring IDEs for JFreeChart

### C.1 Introduction

There are a number of IDEs (integrated development environments) that developers use when working on Java programs. In this section, I describe how to configure some popular IDEs to use JFreeChart.<sup>1</sup> Specifically, I'll cover:

- Eclipse (version 3.2);
- NetBeans (version 5.5);

In the future I'll add configuration descriptions for other IDEs.<sup>2</sup>

### C.2 Eclipse

#### C.2.1 Overview

Eclipse is a free IDE originally developed by IBM, but now managed by the Eclipse Foundation:

<http://www.eclipse.org/>

In Eclipse, third party libraries are configured as “user libraries”. In this section, I'll describe how to set up JFreeChart and JCommon as user libraries in Eclipse 3.2. This makes it straightforward to include JFreeChart and JCommon as dependencies in your application(s), with Eclipse automatically handling features like code-completion, Javadoc popups, stepping through the JFreeChart/JCommon sources during debugging, and more.

#### C.2.2 Configuration Steps

To begin with, you need to download the JFreeChart and JCommon distributions, unpack them on your local machine, and generate the API documentation. The following steps are necessary:

1. Download the latest version of the JCommon class library:

<http://www.jfree.org/jcommon/>

---

<sup>1</sup>Notes that this section is concerned with *using* JFreeChart as a library. If you intend to *modify* the JFreeChart sources, you'll want to configure JFreeChart as a project within your IDE.

<sup>2</sup>At least those I can get access to.

...and unpack it to a directory on your computer (almost anywhere is fine).

2. From the `ant` subdirectory of the just-unpacked `JCommon`, run `ant javadoc` to generate the Javadocs locally. If you are unfamiliar with Ant, you can skip this step, but then Eclipse won't be able to show you the Javadoc popups for `JCommon`.
3. Download the latest version of the `JFreeChart` class library:

<http://www.jfree.org/jfreechart/>

...and unpack it to a directory on your computer (again, almost anywhere is fine).

4. From the `ant` subdirectory of the just-unpacked `JFreeChart`, run `ant javadoc` to generate the Javadocs locally. As with step 2, you can skip this step, but then you'll be missing the API documentation.

Now, launch Eclipse, and carry out the following steps to configure `JFreeChart` and `JCommon` as user libraries:

5. In Eclipse, select `Preferences...` from the `Window` menu, then choose the `Java -> Build Path -> User Libraries` node in the tree—you should see the dialog shown in figure C.1.

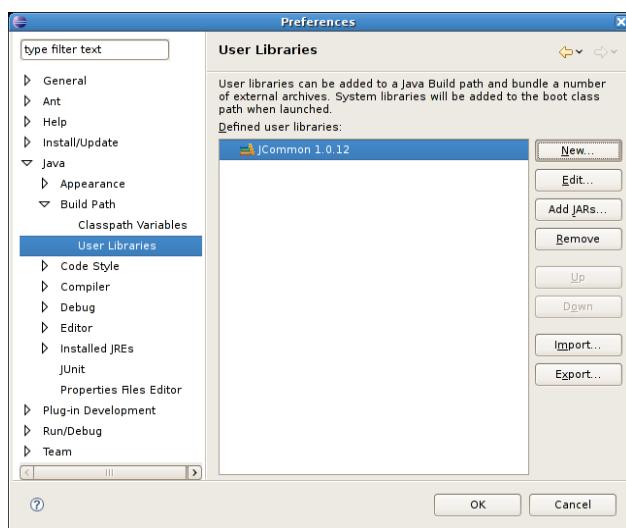


Figure C.1: Eclipse User Libraries Dialog.

6. Click on the `New...` button and enter `JCommon 1.0.12` as the name for a new user library.
7. Ensure that the `JCommon 1.0.12` item is selected in the list, then click the `Add JARs...` button and select the `jcommon-1.0.12.jar` file from the `JCommon` directory created back in step 1.
8. Double-click the item that says “Source attachment: (None)”, then click the `External folder...` button, then select the `source` directory for `JCommon`.
9. Double-click the item that says “Javadoc location: (None)”, then click the `Browse...` button, then select the `javadoc` directory from `JCommon` (see step 2).
10. Click on the `New...` button and enter `JFreeChart 1.0.7` as the name for a new user library.

11. Ensure that the `JFreeChart 1.0.7` item is selected in the list, then click on the `Add JARs...` button and select the `jfreechart-1.0.7.jar` file from the JFreeChart directory (see step 3).
12. Double-click the item that says “`Source attachment: (None)`”, then click the `External folder...` button, then select the `source` directory for JFreeChart.
13. Double-click the item that says “`Javadoc location: (None)`”, then click the `Browse...` button, then select the `javadoc` directory from JFreeChart (see step 4).

At this point, you have completed the configuration of the user libraries—you should have something that looks like figure C.2.

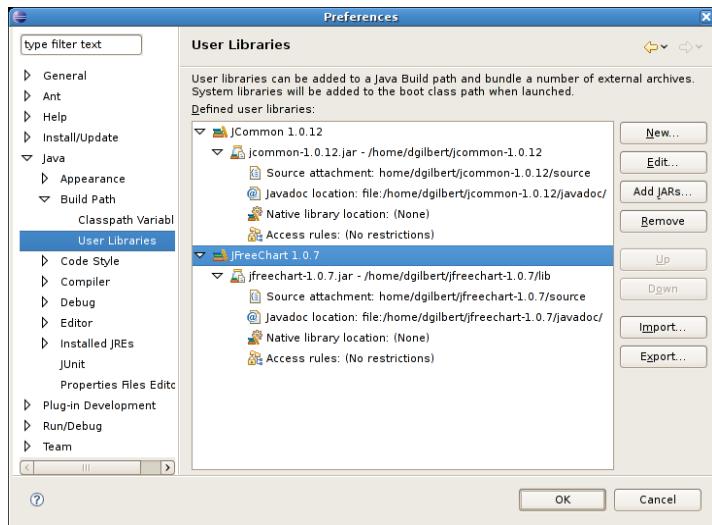


Figure C.2: The Configured User Libraries.

The next section shows how to create a new project in Eclipse that depends on these libraries.

### C.2.3 Creating an Eclipse Project that uses JFreeChart

Now that JFreeChart and JCommon are configured as user libraries, it is straightforward to develop an application that uses these libraries:

1. In Eclipse, select `New -> Project...` from the `File` menu, select `Java Project` from the list and click the `Next` button.
2. Enter `MyAppThatUsesJFreeChart` as the project name and click the `Finish` button.
3. Right-click on the project in the `Package Explorer` then select `Properties` from the pop-up menu. In the properties window—see figure C.3—click on the `Add Library...` button and select both the JCommon and JFreeChart libraries. Click `OK`.
4. Create a new source file (`First.java`) in the project, and copy and paste the following small application:

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

/**
 * This class creates a pie chart showing the results of a survey.
 */
public class First {
    public static void main(String[] args) {
        // Create the data for the chart
        DefaultPieDataset dataset = new DefaultPieDataset();
        dataset.setValue("A", 30);
        dataset.setValue("B", 20);
        dataset.setValue("C", 15);
        dataset.setValue("D", 10);
        dataset.setValue("E", 15);

        // Create the chart
        JFreeChart chart = ChartFactory.createPieChart("Survey Results", dataset, true, false, false);

        // Create the frame
        ChartFrame frame = new ChartFrame("Survey Results", chart);
        frame.pack();
        frame.setVisible(true);
    }
}
```

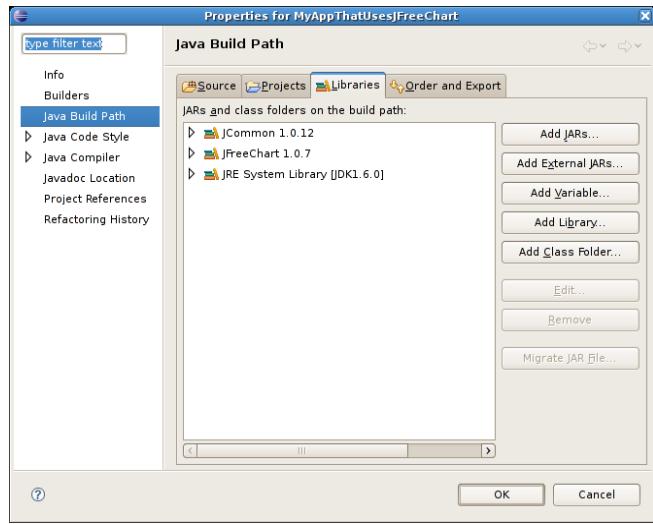


Figure C.3: The completed libraries.

```

/*
 * A simple introduction to using JFreeChart. This demo is described in the
 * JFreeChart Developer Guide.
 */
public class First {

    /**
     * The starting point for the demo.
     *
     * @param args ignored.
     */
    public static void main(String[] args) {

        // create a dataset...
        DefaultPieDataset data = new DefaultPieDataset();
        data.setValue("Category 1", 43.2);
        data.setValue("Category 2", 27.9);
        data.setValue("Category 3", 79.5);

        // create a chart...
        JFreeChart chart = ChartFactory.createPieChart(
            "Sample Pie Chart",
            data,
            true,      // legend?
            true,      // tooltips?
            false     // URLs?
        );

        // create and display a frame...
        ChartFrame frame = new ChartFrame("First", chart);
        frame.pack();
        frame.setVisible(true);

    }
}

```

5. Compile and run the application. Notice how you can browse the JFreeChart/JCommon source files and step through the code while debugging.

That's all there is to it!

## C.3 NetBeans

### C.3.1 Overview

NetBeans is a free IDE developed by Sun Microsystems:

<http://www.netbeans.org/>

In NetBeans, third party libraries are configured using the “Library Manager”. In this section, I’ll describe how to set up JFreeChart and JCommon within the Library Manager in NetBeans version 5.5. This makes it straightforward to include JFreeChart and JCommon as dependencies in your application(s), with NetBeans automatically handling features like code completion, Javadoc popups, stepping through the JFreeChart/JCommon sources during debugging, and more.

### C.3.2 Configuration Steps

To begin with, you need to download the JFreeChart and JCommon distributions, unpack them on your local machine, and generate the API documentation. The following steps are necessary:

1. Download the latest version of the JCommon class library:

<http://www.jfree.org/jcommon/>

...and unpack it to a directory on your computer (almost anywhere is fine).

2. From the `ant` subdirectory of the just-unpacked JCommon, run `ant javadoc` to generate the Javadocs locally. If you are unfamiliar with Ant, you can skip this step, but then NetBeans won’t be able to show you the Javadoc popups for JCommon.

3. Download the latest version of the JFreeChart class library:

<http://www.jfree.org/jfreechart/>

...and unpack it to a directory on your computer (again, almost anywhere is fine).

4. From the `ant` subdirectory of the just-unpacked JFreeChart, run `ant javadoc` to generate the Javadocs locally. As with step 2, you can skip this step, but then you’ll be missing the API documentation.

Now, launch NetBeans, and carry out the following steps to configure JFreeChart and JCommon as user libraries:

5. In NetBeans, select the `Library Manager` item from the `Tools` menu—you should see the dialog shown in figure C.4.
6. Click on the `New Library...` button and enter `JCommon-1.0.12` as the library name.
7. With the `Classpath` tab selected, click on the `Add JAR/Folder...` button and select the `jcommon-1.0.12.jar` file from the JCommon directory created back in step 1.
8. With the `Sources` tab selected, click on the `Add JAR/Folder...` button and select the `source` directory for JCommon.
9. With the `Javadoc` tab selected, click on the `Add ZIP/Folder...` button and select the `javadoc` directory for JCommon (refer to step 2).
10. Click on the `New Library...` button and enter `JFreeChart-1.0.7` as the library name.
11. With the `Classpath` tab selected, click on the `Add JAR/Folder...` button and select the `jfreechart-1.0.7.jar` file from the JFreeChart directory created back in step 3.

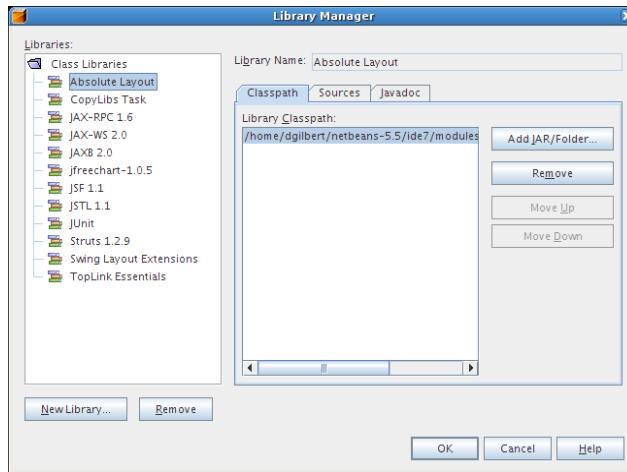


Figure C.4: The Library Manager.

12. With the Sources tab selected, click on the Add JAR/Folder... button and select the source directory for JFreeChart.

13. With the Javadoc tab selected, click on the Add ZIP/Folder... button and select the javadoc directory for JFreeChart (refer to step 4).

At this point, you have complete the configuration of the libraries. The next section shows how to create a new project in NetBeans that depends on these libraries.

### C.3.3 Creating a NetBeans Project that uses JFreeChart

Now that JFreeChart and JCommon are configured as libraries in NetBeans, it is straightforward to develop an application that uses these libraries:

1. In NetBeans, select New Project... from the File menu, select General/Java Application, and click the Next button.
2. Enter MyAppThatUsesJFreeChart as the project name, and click the Finish button.
3. In the Projects pane, you'll see a Libraries node in the project. Right-click on this node, select Add Library... and select the JFreeChart and JCommon libraries.
4. NetBeans has already created a Main.java source file—copy and paste the following code into the main method of this source file:

```
public static void main(String[] args) {

    // create a dataset...
    DefaultPieDataset data = new DefaultPieDataset();
    data.setValue("Category 1", 43.2);
    data.setValue("Category 2", 27.9);
    data.setValue("Category 3", 79.5);

    // create a chart...
    JFreeChart chart = ChartFactory.createPieChart(
        "Sample Pie Chart",
        data,
        true,      // legend?
        true,      // tooltips?
        false     // URLs?
    );
}
```

```
// create and display a frame...
ChartFrame frame = new ChartFrame("First", chart);
frame.pack();
frame.setVisible(true);
}
```

5. Select **Fix Imports** from the **Source** menu, then compile and run the application. Notice how you can browse the JFreeChart/JCommon source files and step through the code while debugging.

That's all there is to it!

## Appendix D

# The GNU Lesser General Public Licence

### D.1 Introduction

JFreeChart is licensed under the terms of the GNU Lesser General Public Licence (LGPL). The full text of this licence is reproduced in this appendix. You should read and understand this licence before using JFreeChart in your own projects.

If you are not familiar with the idea of *free software*, you can find out more at the Free Software Foundation's web site:

<http://www.fsf.org>

Please send e-mail to [david.gilbert@object-refinery.com](mailto:david.gilbert@object-refinery.com) if you have any questions about the licensing of JFreeChart (but please read section [D.3](#) first).

### D.2 The Licence

The following licence has been used for the distribution of the JFreeChart class library:

#### GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

#### Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

#### TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

\* a) The modified work must itself be a software library.

\* b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

\* c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

\* d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

\* a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable “work that uses the Library”, as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

\* b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user’s computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

\* c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

\* d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

\* e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy. For an executable, the required form of the “work that uses the Library” must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

\* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

\* b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work. 8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients’ exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest

validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

#### **NO WARRANTY**

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### **END OF TERMS AND CONDITIONS**

##### **How to Apply These Terms to Your New Libraries**

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
This library is free software; you can redistribute it and/or modify it under the terms of
the GNU Lesser General Public License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.
```

```
This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.
```

```
You should have received a copy of the GNU Lesser General Public License along with this library;
if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
02111-1307 USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library ‘Frob’ (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990  
Ty Coon, President of Vice

That's all there is to it!

## D.3 Frequently Asked Questions

### D.3.1 Introduction

Some of the most frequently asked questions about JFreeChart concern the licence. I've published this FAQ to help developers understand my choice of licence for JFreeChart. If anything is unclear, or technically incorrect, please e-mail me ([david.gilbert@object-refinery.com](mailto:david.gilbert@object-refinery.com)) and I will try to improve the text.

### D.3.2 Questions and Answers

*1. “Can I incorporate JFreeChart into a proprietary (closed-source) application?”*

Yes, the GNU Lesser General Public Licence (LGPL) is specifically designed to allow this.

*2. “Do I have to pay a licence fee to use JFreeChart?”*

No, JFreeChart is free software. You are not required to pay a fee to use JFreeChart. All that we ask is that you comply with the terms of the licence, which (for most developers) is not very difficult.

If you want to make a financial contribution to the JFreeChart project, you can buy a copy of the JFreeChart Developer Guide from Object Refinery Limited. This is appreciated, but not required.

*3. “If I use JFreeChart, do I have to release the source code for my application under the terms of the LGPL?”*

No, you can choose whatever licence you wish for your software. But when you distribute your application, you must include the complete source code for JFreeChart—including any changes you make to it—under the terms of the LGPL. Your users end up with the same rights in relation to JFreeChart as you have been granted under the LGPL.

*4. “My users will never look at the source code, and if they did, they wouldn’t know what to do with it...why do I have to give it to them?”*

The important point is that your users have access to the source code—whether or not they choose to use it is up to them. Bear in mind that non-technical users *can* make use of the source code by hiring someone else to work on it for them.

*5. “What are the steps I must follow to release software that incorporates JFreeChart?”*

The steps are listed in the licence (see section 6 especially). The most important things are:

- include a notice in your software that it uses the JFreeChart class library, and that the library is covered by the LGPL;
- include a copy of the LGPL so your users understand that JFreeChart is distributed WITHOUT WARRANTY, and the rights that they have under the licence;
- include the complete source code for the version of the library that you are distributing (or a written offer to supply it on demand);

*6. “I want to display the JFreeChart copyright notice, what form should it take?”*

Try this:

*This software incorporates JFreeChart, (C)opyright 2000-2007 by Object Refinery Limited and Contributors.*

*7. “The LGPL is unnecessarily complicated!”*

OK, that's not a question, but the point has been raised by a few developers.

Yes, the LGPL is complicated, but only out of necessity. The complexity is mostly related to the difficulty of defining (in precise legal terms) the relationship between a free software library and a proprietary application that uses the library.

A useful first step towards understanding the LGPL is to read the GNU General Public Licence (GPL). It is a much simpler licence, because it does not allow free software to be combined with non-free (or proprietary) software. The LGPL is a superset of the GPL (you are free to switch from the LGPL to the GPL at any time), but slightly more “relaxed” in that it allows you to combine free and non-free software.

A final note, some of the terminology in the LGPL is easier to understand if you keep in mind that the licence was originally developed with statically-linked C programs in mind. Ensuring that it is possible to relink a modified free library with a non-free application, adds significant complexity to the licence. For Java libraries, where code is dynamically linked, modifying and rebuilding a free library for use with a non-free application needn’t be such a big issue, particularly if the free library resides in its own jar file.

*8. “Who developed the licence?”*

The licence was developed by the Free Software Foundation and has been adopted by many thousands of free software projects. You can find out more information at the Free Software Foundation website:

<http://www.fsf.org>

The Free Software Foundation performs important work, please consider supporting them financially.

*9. “Have you considered releasing JFreeChart under a different licence, such as an “Apache-style” licence?”*

Yes, a range of licences was considered for JFreeChart, but now that the choice has been made there are no plans to change the licence in the future.

A publication by Bruce Perens was especially helpful in comparing the available licences:

<http://www.oreilly.com/catalog/opensources/book/perens.html>

In the end, the LGPL was chosen because it is the closest fit in terms of my goals for JFreeChart. It is not a perfect licence, but there is nothing else that comes close (except the GPL) in terms of protecting the freedom of JFreeChart for everyone to use. Also, the LGPL is very widely used, and many developers are already familiar with its requirements.

Some other open source licences (for example the Apache Software Licence) allow open source software to be packaged and redistributed without source code. These licences offer more convenience to developers (especially in large companies) than the LGPL, but they allow a path from open source software to closed source software, which is not something I want to allow for JFreeChart.

# Index

`AbstractBlock`, 231  
`AbstractCategoryItemLabelGenerator`, 274  
`AbstractCategoryItemRenderer`, 447  
`AbstractDataset`, 628  
`AbstractDialLayer`, 395  
`AbstractIntervalXYDataset`, 716  
`AbstractPieItemLabelGenerator`, 275  
`AbstractRenderer`, 421  
`AbstractSeriesDataset`, 629  
`AbstractXYAnnotation`, 155  
`AbstractXYDataset`, 716  
`AbstractXYItemLabelGenerator`, 276  
`AbstractXYItemRenderer`, 500  
`AbstractXYZDataset`, 717  
acknowledgements, 18  
Acrobat PDF, 103  
adding chart change listeners, 149  
`Align`, 775  
annotations, 155  
    `XYPlot`, 391  
anti-aliasing, 69  
applets, 114  
`ArcDialFrame`, 397  
area charts  
    `StackedXYAreaRenderer`, 513  
    `XYAreaRenderer`, 523  
`AreaRenderer`, 451  
`AreaRendererEndType`, 440  
`Arrangement`, 233  
`Axis`, 177  
axis  
    display integers only, 209  
Axis label rotation, 179  
axis labels  
    as percentages, 209  
`AxisChangeEvent`, 263  
`AxisChangeListener`, 263  
`AxisCollection`, 182  
`AxisLocation`, 182  
`AxisSpace`, 183  
`AxisState`, 183  
background image, 68  
bar  
    outline, 454, 526  
bar charts  
    `CategoryPlot`, 452  
    `XYBarRenderer`, 525  
    `XYPlot`, 525  
    `BarRenderer`, 452  
    `BarRenderer3D`, 457  
    `Batik`, 111  
    `Block`, 234  
    `BlockBorder`, 234  
    `BlockContainer`, 235  
    `BlockFrame`, 237  
    `BlockParams`, 237  
    `BlockResult`, 237  
border, 67  
    for plots, 357  
`BorderArrangement`, 238  
box and whisker chart  
    dataset, 653  
`BoxAndWhiskerCalculator`, 649  
`BoxAndWhiskerCategoryDataset`, 650  
`BoxAndWhiskerItem`, 651  
`BoxAndWhiskerRenderer`, 459  
`BoxAndWhiskerToolTipGenerator`, 278  
`BoxAndWhiskerXYDataset`, 652  
`BoxAndWhiskerXYToolTipGenerator`, 278  
bubble charts, 530  
candle-stick chart  
    `CandlestickRenderer`, 505  
`CandlestickRenderer`, 505  
`catcode.com`, 142  
category axis  
    margins, 185  
    multi-line category labels, 186  
category labels, 186  
    rotate 90 degrees, 186  
`CategoryAnchor`, 184  
`CategoryAnnotation`, 157  
`CategoryAxis`, 184  
`CategoryAxis3D`, 189  
`CategoryDataset`, 609  
`CategoryItemEntity`, 256  
`CategoryItemLabelGenerator`, 278  
`CategoryItemRenderer`, 460  
`CategoryItemRendererState`, 470  
`CategoryLabelPosition`, 190  
`CategoryLabelPositions`, 191

CategoryLabelWidthType, 191  
CategoryLineAnnotation, 157  
CategoryMarker, 304  
CategoryPlot, 306  
CategoryPlot  
    item rendering order, 308  
CategoryPointerAnnotation, 159  
CategorySeriesLabelGenerator, 279  
CategoryStepRenderer, 471  
CategoryTableXYDataset, 717  
CategoryTextAnnotation, 161  
CategoryTick, 192  
CategoryToolTipGenerator, 279  
CategoryToPieDataset, 609  
CenterArrangement, 238  
Cewolf, 129  
chart  
    background color, 68  
    background image, 68  
    border, 67  
    subtitles, 68  
    title, 67  
chart border, 149  
chart change listeners, 149  
chart entities, 256  
ChartChangeEvent, 264  
ChartChangeEvent Type, 264  
ChartChangeListener, 265  
ChartColor, 131  
ChartDelete, 560  
ChartEntity, 258  
ChartFactory, 131  
ChartFrame, 134  
ChartMouseEvent, 135  
ChartMouseListener, 136  
ChartPanel, 136  
ChartProgressEvent, 265  
ChartProgressListener, 266  
ChartRenderingInfo, 141  
ChartUtilities, 142  
ClipPath, 144  
ClusteredXYBarRenderer, 508  
ColorBar, 192  
ColorBlock, 238  
ColumnArrangement, 239  
CombinationDataset, 630  
combined charts, 94  
CombinedDataset, 630  
CombinedDomainCategoryPlot, 312  
CombinedDomainXYPlot, 313  
CombinedRangeCategoryPlot, 315  
CombinedRangeXYPlot, 316  
comments and suggestions, 18  
ComparableObjectItem, 588  
ComparableObjectSeries, 589  
CompassFormat, 192  
CompassPlot, 317  
compiling JFreeChart, 36  
CompositeTitle, 562  
ContourDataset, 617  
ContourEntity, 258  
ContourPlot, 320  
ContourPlotUtilities, 321  
ContourToolTipGenerator, 280  
ContourValuePlot, 321  
contributors, 18  
CrosshairState, 321  
CSV, 645  
CustomXYToolTipGenerator, 280  
CyclicNumberAxis, 193  
CyclicXYItemRenderer, 509  
Dataset, 630  
dataset rendering order  
    XYPlot, 385  
DatasetChangeEvent, 631  
DatasetChangeListener, 631  
DatasetGroup, 631  
DatasetRenderingOrder, 321  
DatasetUtilities, 632  
DataUtilities, 590  
DateAxis, 194  
    rotate tick labels, 229  
DateRange, 674  
DateTick, 198  
DateTickMarkPosition, 198  
DateTickUnit, 199  
DateTitle, 563  
Day, 674  
DefaultBoxAndWhiskerCategoryDataset, 653  
DefaultBoxAndWhiskerXYDataset, 656  
DefaultCategoryDataset, 611  
DefaultCategoryItemRenderer, 473  
DefaultContourDataset, 618  
DefaultDrawingSupplier, 322  
DefaultHighLowDataset, 719  
DefaultIntervalCategoryDataset, 613  
DefaultIntervalXYDataset, 720  
DefaultKeyedValue, 591  
DefaultKeyedValueDataset, 636  
DefaultKeyedValues, 592  
DefaultKeyedValues2D, 594  
DefaultKeyedValues2DDataset, 636  
DefaultKeyedValuesDataset, 636  
DefaultMultiValueCategoryDataset, 658  
DefaultOHCDDataset, 723  
DefaultPieDataset, 636  
DefaultPolarItemRenderer, 440  
DefaultStatisticalCategoryDataset, 660

DefaultTableXYDataset, 724  
DefaultValueDataset, 638  
DefaultWindDataset, 726  
DefaultXYDataset, 728  
DefaultXYItemRenderer, 510  
demo  
    running, 35  
DeviationRenderer, 510  
DialBackground, 398  
DialCap, 399  
DialFrame, 401  
DialLayer, 401  
DialLayerChangeEvent, 402  
DialLayerChangeListener, 403  
DialPlot, 403  
DialPointer, 406  
DialPointer.Pin, 407  
DialPointer.Pointer, 408  
DialScale, 409  
DialShape, 323  
DialTextAnnotation, 410  
DialValueIndicator, 411  
disabling chart change events, 149  
DisplayChart, 560  
distribution  
    contents, 35  
domain axis, 177  
DomainInfo, 596  
DomainOrder, 596  
download, 34  
DrawableLegendItem, 144  
DrawingSupplier, 324  
dynamic charts, 72  
DynamicTimeSeriesCollection, 676  
Eclipse, 783  
Effect3D, 144  
EmptyBlock, 240  
EncoderUtil, 251  
entities, 256  
EntityBlockParams, 240  
EntityBlockResult, 240  
EntityCollection, 259  
events, 263  
exporting charts  
    to JPEG, 143  
    to PDF, 103  
    to PNG, 142  
    to SVG, 111  
ExtendedCategoryAxis, 200  
FastScatterPlot, 325  
features, 16  
FixedMillisecond, 678  
FlowArrangement, 240  
Free Software Foundation, 790  
Function2D, 619  
GanttCategoryDataset, 622  
GanttRenderer, 473  
GradientPaintTransformer, 776  
GradientPaintTransformType, 776  
GrayPaintScale, 441  
GridArrangement, 241  
gridlines  
    XYPlot, 387  
GroupedStackedBarRenderer, 474  
HashUtilities, 145  
headless Java, 129  
HighLowItemLabelGenerator, 280  
HighLowRenderer, 512  
HistogramBin, 662  
HistogramDataset, 663  
HistogramType, 665  
home page, 17  
Hour, 678  
HTML image map, 143  
IDE configuration, 36  
image loading, 129  
image maps, 143, 270  
ImageEncoder, 253  
ImageEncoderFactory, 252  
ImageFormat, 253  
ImageMapUtilities, 270  
images  
    JPEG format, 143  
    PNG format, 142  
ImageTitle, 564  
IntervalBarRenderer, 475  
IntervalCategoryDataset, 616  
IntervalCategoryItemLabelGenerator, 281  
IntervalCategoryToolTipGenerator, 282  
IntervalMarker, 328  
IntervalXYDataset, 732  
IntervalXYDelegate, 733  
IntervalXYZDataset, 734  
item labels, 79  
    XYItemRenderer, 542  
ItemLabelAnchor, 282  
ItemLabelPosition, 283  
iText, 103  
    PDF, 103  
Javadoc, 36  
JDDBCCategoryDataset, 646  
JDBCPieDataset, 647  
JDBCXYDataset, 647  
JFreeChart, 145

JFreeChart  
    applets, 114  
    license, 790  
    overview and features, 16  
    sample charts, 19  
    servlets, 118  
JPEG  
    exporting charts to, 143  
JSP, 129  
  
KeyedObject, 597  
KeyedObjects, 597  
KeyedObjects2D, 599  
KeyValue, 601  
KeyValueComparator, 601  
KeyValueComparatorType, 601  
KeyValueDataset, 638  
KeyedValues, 602  
KeyedValues2D, 602  
KeyValue2DDataset, 639  
KeyedValuesDataset, 639  
KeyPointPNGEncoderAdapter, 253  
KeyToGroupMap, 603  
  
LabelBlock, 241  
LayeredBarRenderer, 476  
legend  
    font, 569  
    positioning, 568  
LegendGraphic, 564  
LegendItem, 150  
LegendItemBlockContainer, 566  
LegendItemCollection, 152  
LegendItemEntity, 259  
LegendItemSource, 153  
LegendRenderingOrder, 154  
LegendTitle, 567  
LengthConstraintType, 243  
LevelRenderer, 477  
LGPL, 790  
license, 790  
    frequently asked questions, 796  
line chart  
    LineAndShapeRenderer, 479  
    LineRenderer3D, 483  
    XYLineAndShapeRenderer, 548  
LineAndShapeRenderer, 479  
linear regression, 667  
LineBorder, 243  
LineFunction2D, 619  
LineRenderer3D, 483  
logarithmic scale  
    LogAxis, 200  
LogarithmicAxis, 203  
LogAxis, 200  
LookupPaintScale, 442  
mapping datasets to axes  
    XYPlot, 387  
Marker, 330  
MarkerAxisBand, 204  
MarkerChangeEvent, 266  
MarkerChangeListener, 267  
MatrixSeries, 734  
MatrixSeriesCollection, 735  
MeanAndStandardDeviation, 666  
MeterInterval, 334  
MeterPlot, 335  
migration, 767  
Millisecond, 679  
MinMaxCategoryRenderer, 485  
Minute, 680  
ModuloAxis, 205  
Month, 681  
MonthDateFormat, 206  
MovingAverage, 682  
MultiplePiePlot, 339  
MultipleXYSeriesLabelGenerator, 284  
MultiValueCategoryDataset, 666  
  
NetBeans, 787  
NonGridContourDataset, 618  
NormalDistributionFunction2D, 620  
NormalizedMatrixSeries, 736  
NotOutlierException, 443  
NumberAxis, 207  
    display integers only, 209  
    rotate tick labels, 229  
NumberAxis3D, 211  
NumberTick, 212  
NumberTickUnit, 213  
  
OHLC, 707  
OHLCDataItem, 737  
OHLCDataSet, 738  
OHLCItem, 708  
OHLCSeries, 709  
OHLCSeriesCollection, 709  
Outlier, 443  
OutlierList, 444  
OutlierListCollection, 444  
overriding the tick label format, 209  
  
PaintScale, 444  
PaintScaleLegend, 570  
PDF, 103  
PeriodAxis, 214  
PeriodAxisLabelInfo, 216  
pie chart  
    DefaultPieDataset, 636

PiePlot3D, 354  
PiePlot, 342  
dataset interface, 639  
exploded sections, 348  
section labels, 348  
shadow effect, 347  
PieDataset, 639  
PieLabelDistributor, 342  
PieLabelRecord, 342  
PiePlot, 342  
border, 345  
PiePlot3D, 354  
PiePlotState, 356  
PieSectionEntity, 260  
PieSectionLabelGenerator, 285  
PieToolTipGenerator, 286  
Plot, 356  
background, 358  
background image, 358  
border, 357  
drawing supplier, 359  
legend items, 360  
PlotChangeEvent, 267  
PlotChangeListener, 268  
PlotOrientation, 361  
PlotRenderingInfo, 361  
PlotState, 363  
PlotUtilities, 363  
PNG, 142  
PolarChartPanel, 154  
PolarItemRenderer, 445  
PolarPlot, 363  
power regression, 667  
PowerFunction2D, 621  
PublicCloneable, 776  
Quarter, 683  
QuarterDateFormat, 218  
Range, 604  
range axis, 177  
RangeInfo, 606  
RangeType, 606  
real time charts, 73  
RectangleAnchor, 777  
RectangleConstraint, 244  
RectangleEdge, 777  
RectangleInsets, 777  
Regression, 667  
RegularTimePeriod, 684  
RelativeDateFormat, 585  
renderer  
shape attributes, 541  
RendererChangeEvent, 268  
RendererChangeListener, 268  
RendererState, 446  
rendering hints, 69  
rendering order  
CategoryPlot, 308  
RingPlot, 368  
sample charts, 19  
scatter plot  
XYDotRenderer, 534  
ScatterRenderer, 487  
Second, 686  
SegmentedTimeline, 218  
Series, 640  
SeriesChangeEvent, 642  
SeriesChangeListener, 642  
SeriesDataset, 642  
SeriesException, 643  
SeriesRenderingOrder, 370  
servlets, 118  
deploying, 127  
ServletUtilities, 560  
SimpleDialFrame, 414  
SimpleHistogramBin, 668  
SimpleHistogramDataset, 669  
SimpleTimePeriod, 687  
SpiderWebPlot, 370  
spline curves, 554  
stacked bars  
showing percentages, 491  
StackedAreaRenderer, 490  
StackedBarRenderer, 490  
StackedBarRenderer3D, 492  
StackedXYAreaRenderer, 513  
StackedXYAreaRenderer2, 515  
StackedXYBarRenderer, 516  
StandardCategoryItemLabelGenerator, 286  
StandardCategorySeriesLabelGenerator, 287  
StandardCategoryToolTipGenerator, 288  
StandardContourToolTipGenerator, 289  
StandardDialRange, 415  
StandardDialScale, 417  
StandardEntityCollection, 260  
StandardGradientPaintTransformer, 779  
StandardPieSectionLabelGenerator, 289  
StandardPieToolTipGenerator, 291  
StandardTickUnitSource, 219  
StandardXYItemLabelGenerator, 292  
StandardXYItemRenderer, 518  
StandardXYSeriesLabelGenerator, 293  
StandardXYZToolTipGenerator, 294  
StandardXYZToolTipGenerator, 295  
StatisticalBarRenderer, 494  
StatisticalCategoryDataset, 671  
StatisticalLineAndShapeRenderer, 495  
Statistics, 671

step charts, 556  
SubCategoryAxis, 220  
subtitles, 68  
SunJPEGEncoderAdapter, 254  
SunPNGEncoderAdapter, 255  
suppressing chart change events, 149  
SVG, 111  
SymbolAxis, 221  
SymbolicXYItemLabelGenerator, 296  
  
TableXYDataset, 739  
Task, 623  
TaskSeries, 624  
TaskSeriesCollection, 625  
TextAnchor, 781  
TextAnnotation, 162  
TextTitle, 572  
thermometer plot  
    dataset, 376  
    general attributes, 376  
    subranges, 379  
    value label, 377  
ThermometerPlot, 375  
Tick, 223  
TickLabelEntity, 261  
TickType, 223  
TickUnit, 223  
TickUnits, 224  
TickUnitSource, 225  
time series  
    tool tips, 294  
Timeline, 225  
TimePeriod, 687  
TimePeriodAnchor, 688  
TimePeriodFormatException, 688  
TimePeriodValue, 688  
TimePeriodValues, 689  
TimePeriodValuesCollection, 691  
TimeSeries, 693  
TimeSeriesCollection, 698  
TimeSeriesDataItem, 701  
TimeSeriesTableModel, 702  
TimeTableXYDataset, 702  
Title, 574  
title, 67  
TitleChangeEvent, 269  
TitleChangeListener, 269  
tool tips  
    time series, 294  
tooltips, 77  
    pie charts, 77  
  
Unicode, 108  
UnitType, 781  
UnknownKeyException, 607  
  
unpacking the JFreeChart distribution, 34  
upgrading versions, 767  
  
Value, 607  
ValueAxis, 226  
    rotate tick labels, 229  
ValueAxisPlot, 382  
ValueDataset, 643  
ValueMarker, 382  
Values, 607  
Values2D, 608  
ValueTick, 229  
Vector, 739  
VectorDataItem, 740  
VectorRenderer, 521  
VectorSeries, 741  
VectorSeriesCollection, 742  
VectorXYDataset, 744  
  
WaferMapDataset, 644  
WaferMapPlot, 383  
WaferMapRenderer, 446  
WaterfallBarRenderer, 497  
Week, 704  
WindDataset, 744  
WindItemRenderer, 523  
  
X11, 129  
XisSymbolic, 745  
XYAnnotation, 164  
XYAnnotationEntity, 262  
XYAreaRenderer, 523  
XYBarDataset, 730, 745  
XYBarRenderer, 525  
XYBlockRenderer, 527  
XYBoxAndWhiskerRenderer, 529  
XYBoxAnnotation, 165  
XYBubbleRenderer, 530  
XYCoordinate, 747  
XYDataItem, 748  
XYDataset, 748  
XYDatasetTableModel, 750  
XYDifferenceRenderer, 532  
XYDotRenderer, 534  
XYDrawableAnnotation, 166  
XYErrorRenderer, 535  
XYImageAnnotation, 167  
XYInterval, 751  
XYIntervalDataItem, 752  
XYIntervalSeries, 753  
XYIntervalSeriesCollection, 754  
XYItemEntity, 262  
XYItemLabelGenerator, 296  
XYItemRenderer, 537  
XYItemRendererState, 547

XYLineAndShapeRenderer, 548  
XYLineAnnotation, 168  
XYPlot, 383  
    zero base line, 389  
XYPointerAnnotation, 169  
XYPolygonAnnotation, 171  
XYSeries, 756  
XYSeriesCollection, 759  
XYSeriesLabelGenerator, 296  
XYShapeAnnotation, 173  
XYSplineRenderer, 554  
XYStepAreaRenderer, 557  
XYStepRenderer, 556  
XYTextAnnotation, 174  
XYToolTipGenerator, 297  
XYZDataset, 761  
XYZToolTipGenerator, 297  
  
Year, 705  
YInterval, 761  
YIntervalDataItem, 762  
YIntervalRenderer, 558  
YIntervalSeries, 763  
YIntervalSeriesCollection, 764  
YisSymbolic, 765  
  
Zoomable, 393