

# The JFreeChart Class Library

Version 1.0.0-pre2

## Developer Guide

Written by David Gilbert

March 10, 2005

© 2000-2005, Object Refinery Limited. All rights reserved.

### **IMPORTANT NOTICE:**

**If you choose to use this document you do so entirely at your own risk.**

# Contents

<b>1</b>	<b>Introduction</b>	<b>15</b>
1.1	What is JFreeChart?	15
1.2	This Document	17
1.3	Acknowledgements	17
1.4	Comments and Suggestions	17
<b>2</b>	<b>Sample Charts</b>	<b>18</b>
2.1	Introduction	18
2.2	Pie Charts	18
2.3	Bar Charts	20
2.4	Line Chart	22
2.5	XY Plots	23
2.6	Time Series Charts	24
2.7	Histograms	25
2.8	Area Charts	26
2.9	Difference Chart	26
2.10	Step Chart	28
2.11	Gantt Chart	29
2.12	Multiple Axis Charts	30
2.13	Combined and Overlaid Charts	31
2.14	Future Development	32
<b>3</b>	<b>Downloading and Installing JFreeChart</b>	<b>33</b>
3.1	Introduction	33
3.2	Download	33
3.3	Unpacking the Files	34
3.4	Running the Demonstration Applications	35
3.5	Compiling the Source	35
3.6	Generating the Javadoc Documentation	35
<b>4</b>	<b>Using JFreeChart</b>	<b>36</b>
4.1	Overview	36
4.2	Creating Your First Chart	36
<b>5</b>	<b>Bar Charts</b>	<b>39</b>
5.1	Introduction	39
5.2	A Bar Chart	39
5.3	Customising Bar Charts	44

<b>6 Line Charts</b>	<b>46</b>
6.1 Introduction . . . . .	46
6.2 A Line Chart Based On A Category Dataset . . . . .	46
<b>7 Time Series Charts</b>	<b>58</b>
7.1 Introduction . . . . .	58
7.2 Time Series Charts . . . . .	58
<b>8 Customising Charts</b>	<b>64</b>
8.1 Introduction . . . . .	64
8.2 Chart Attributes . . . . .	64
8.3 Plot Attributes . . . . .	66
8.4 Axis Attributes . . . . .	67
<b>9 Dynamic Charts</b>	<b>70</b>
9.1 Overview . . . . .	70
9.2 Background . . . . .	70
9.3 The Demo Application . . . . .	71
<b>10 Tooltips</b>	<b>76</b>
10.1 Overview . . . . .	76
10.2 Generating Tool Tips . . . . .	76
10.3 Collecting Tool Tips . . . . .	77
10.4 Displaying Tool Tips . . . . .	77
10.5 Disabling Tool Tips . . . . .	78
10.6 Customising Tool Tips . . . . .	78
<b>11 Item Labels</b>	<b>79</b>
11.1 Introduction . . . . .	79
11.2 Displaying Item Labels . . . . .	80
11.3 Item Label Appearance . . . . .	82
11.4 Item Label Positioning . . . . .	83
11.5 Customising the Item Label Text . . . . .	84
11.6 Example 1 - Values Above a Threshold . . . . .	84
11.7 Example 2 - Displaying Percentages . . . . .	88
<b>12 Multiple Axes and Datasets</b>	<b>93</b>
12.1 Introduction . . . . .	93
12.2 An Example . . . . .	94
12.3 Hints and Tips . . . . .	95
<b>13 Combined Charts</b>	<b>96</b>
13.1 Introduction . . . . .	96
13.2 Combined Domain Category Plot . . . . .	96
13.3 Combined Range Category Plot . . . . .	97
13.4 Combined Domain XY Plot . . . . .	99
13.5 Combined Range XY Plot . . . . .	100

<b>14 Datasets and JDBC</b>	<b>102</b>
14.1 Introduction . . . . .	102
14.2 About JDBC . . . . .	102
14.3 Sample Data . . . . .	102
14.4 PostgreSQL . . . . .	103
14.5 The JDBC Driver . . . . .	105
14.6 The Demo Applications . . . . .	105
<b>15 Exporting Charts to Acrobat PDF</b>	<b>107</b>
15.1 Introduction . . . . .	107
15.2 What is Acrobat PDF? . . . . .	107
15.3 iText . . . . .	107
15.4 Graphics2D . . . . .	108
15.5 Getting Started . . . . .	108
15.6 The Application . . . . .	109
15.7 Viewing the PDF File . . . . .	113
15.8 Unicode Characters . . . . .	113
<b>16 Exporting Charts to SVG Format</b>	<b>117</b>
16.1 Introduction . . . . .	117
16.2 Background . . . . .	117
16.3 A Sample Application . . . . .	117
<b>17 Applets</b>	<b>121</b>
17.1 Introduction . . . . .	121
17.2 Issues . . . . .	121
17.3 A Sample Applet . . . . .	123
<b>18 Servlets</b>	<b>126</b>
18.1 Introduction . . . . .	126
18.2 A Simple Servlet . . . . .	126
18.3 Compiling the Servlet . . . . .	129
18.4 Deploying the Servlet . . . . .	129
18.5 Embedding Charts in HTML Pages . . . . .	130
18.6 Supporting Files . . . . .	135
18.7 Deploying Servlets . . . . .	136
<b>19 Miscellaneous</b>	<b>138</b>
19.1 Introduction . . . . .	138
19.2 X11 / Headless Java . . . . .	138
19.3 Java Server Pages . . . . .	138
19.4 Loading Images . . . . .	138
<b>20 Packages</b>	<b>140</b>
20.1 Overview . . . . .	140

<b>21 Package: org.jfree.chart</b>	<b>141</b>
21.1 Overview . . . . .	141
21.2 ChartColor . . . . .	141
21.3 ChartFactory . . . . .	141
21.4 ChartFrame . . . . .	144
21.5 ChartMouseEvent . . . . .	144
21.6 ChartMouseListener . . . . .	145
21.7 ChartPanel . . . . .	145
21.8 ChartRenderingInfo . . . . .	149
21.9 ChartUtilities . . . . .	149
21.10 ClipPath . . . . .	151
21.11 DrawableLegendItem . . . . .	151
21.12 Effect3D . . . . .	152
21.13 JFreeChart . . . . .	152
21.14 Legend . . . . .	157
21.15 LegendItem . . . . .	157
21.16 LegendItemCollection . . . . .	158
21.17 LegendItemSource . . . . .	159
21.18 LegendRenderingOrder . . . . .	159
21.19 MeterLegend . . . . .	159
21.20 PolarChartPanel . . . . .	159
21.21 StandardLegend . . . . .	160
<b>22 Package: org.jfree.chart.annotations</b>	<b>162</b>
22.1 Overview . . . . .	162
22.2 CategoryAnnotation . . . . .	162
22.3 CategoryTextAnnotation . . . . .	162
22.4 TextAnnotation . . . . .	163
22.5 XYAnnotation . . . . .	163
22.6 XYDrawableAnnotation . . . . .	164
22.7 XYImageAnnotation . . . . .	164
22.8 XYLineAnnotation . . . . .	165
22.9 XYPointerAnnotation . . . . .	166
22.10 XYPolygonAnnotation . . . . .	168
22.11 XYShapeAnnotation . . . . .	169
22.12 XYTextAnnotation . . . . .	169
<b>23 Package: org.jfree.chart.axis</b>	<b>171</b>
23.1 Overview . . . . .	171
23.2 Axis . . . . .	171
23.3 AxisCollection . . . . .	175
23.4 AxisLocation . . . . .	176
23.5 AxisSpace . . . . .	176
23.6 AxisState . . . . .	177
23.7 CategoryAnchor . . . . .	177
23.8 CategoryAxis . . . . .	178
23.9 CategoryAxis3D . . . . .	182
23.10 CategoryLabelPosition . . . . .	182
23.11 CategoryLabelPositions . . . . .	183
23.12 CategoryLabelWidthType . . . . .	184

23.13CategoryTick . . . . .	184
23.14ColorBar . . . . .	184
23.15CompassFormat . . . . .	185
23.16CyclicNumberAxis . . . . .	185
23.17DateAxis . . . . .	186
23.18DateTickMarkPosition . . . . .	191
23.19DateTick . . . . .	191
23.20DateTickUnit . . . . .	191
23.21ExtendedCategoryAxis . . . . .	193
23.22LogarithmicAxis . . . . .	193
23.23MarkerAxisBand . . . . .	193
23.24ModuloAxis . . . . .	193
23.25NumberAxis . . . . .	195
23.26NumberAxis3D . . . . .	198
23.27NumberTick . . . . .	198
23.28NumberTickUnit . . . . .	198
23.29PeriodAxis . . . . .	199
23.30PeriodAxisLabelInfo . . . . .	202
23.31SegmentedTimeline . . . . .	204
23.32StandardTickUnitSource . . . . .	204
23.33SubCategoryAxis . . . . .	204
23.34SymbolicAxis . . . . .	204
23.35SymbolicTickUnit . . . . .	204
23.36Tick . . . . .	204
23.37TickUnit . . . . .	205
23.38TickUnits . . . . .	205
23.39TickUnitSource . . . . .	206
23.40Timeline . . . . .	206
23.41ValueAxis . . . . .	207
23.42ValueTick . . . . .	210
<b>24 Package: org.jfree.chart.block</b>	<b>211</b>
24.1 Introduction . . . . .	211
24.2 AbstractBlock . . . . .	211
24.3 Arrangement . . . . .	213
24.4 Block . . . . .	213
24.5 BlockBorder . . . . .	214
24.6 BlockContainer . . . . .	214
24.7 BorderArrangement . . . . .	215
24.8 ColorBlock . . . . .	216
24.9 ColumnArrangement . . . . .	216
24.10EmptyBlock . . . . .	217
24.11FlowArrangement . . . . .	217
24.12GridArrangement . . . . .	217
24.13LabelBlock . . . . .	218
24.14LengthConstraintType . . . . .	219
24.15RectangleConstraint . . . . .	219

<b>25 Package: org.jfree.chart.entity</b>	<b>221</b>
25.1 Introduction . . . . .	221
25.2 Background . . . . .	221
25.3 CategoryItemEntity . . . . .	221
25.4 ChartEntity . . . . .	222
25.5 ContourEntity . . . . .	223
25.6 EntityCollection . . . . .	223
25.7 LegendItemEntity . . . . .	224
25.8 PieSectionEntity . . . . .	224
25.9 StandardEntityCollection . . . . .	224
25.10 TickLabelEntity . . . . .	225
25.11 XYItemEntity . . . . .	225
<b>26 Package: org.jfree.chart.event</b>	<b>226</b>
26.1 Introduction . . . . .	226
26.2 AxisChangeEvent . . . . .	226
26.3 AxisChangeListener . . . . .	226
26.4 ChartChangeEvent . . . . .	227
26.5 ChartChangeEvent-Type . . . . .	228
26.6 ChartChangeListener . . . . .	228
26.7 ChartProgressEvent . . . . .	228
26.8 ChartProgressListener . . . . .	229
26.9 LegendChangeEvent . . . . .	229
26.10 LegendChangeListener . . . . .	229
26.11 PlotChangeEvent . . . . .	229
26.12 PlotChangeListener . . . . .	230
26.13 RendererChangeEvent . . . . .	230
26.14 RendererChangeListener . . . . .	230
26.15 TitleChangeEvent . . . . .	231
26.16 TitleChangeListener . . . . .	231
<b>27 Package: org.jfree.chart.imagemap</b>	<b>232</b>
27.1 Overview . . . . .	232
27.2 DynamicDriveToolTipTagFragmentGenerator . . . . .	232
27.3 ImageMapUtilities . . . . .	232
27.4 OverLIBToolTipTagFragmentGenerator . . . . .	233
27.5 StandardToolTipTagFragmentGenerator . . . . .	233
27.6 StandardURLTagFragmentGenerator . . . . .	233
27.7 ToolTipTagFragmentGenerator . . . . .	233
27.8 URLTagFragmentGenerator . . . . .	234
<b>28 Package: org.jfree.chart.labels</b>	<b>235</b>
28.1 Introduction . . . . .	235
28.2 AbstractCategoryItemLabelGenerator . . . . .	235
28.3 AbstractXYItemLabelGenerator . . . . .	236
28.4 BoxAndWhiskerToolTipGenerator . . . . .	238
28.5 BoxAndWhiskerXYToolTipGenerator . . . . .	238
28.6 CategoryLabelGenerator . . . . .	238
28.7 CategoryToolTipGenerator . . . . .	239
28.8 ContourToolTipGenerator . . . . .	239

28.9 CustomXYToolTipGenerator . . . . .	239
28.10 HighLowItemLabelGenerator . . . . .	240
28.11 IntervalCategoryLabelGenerator . . . . .	241
28.12 IntervalCategoryToolTipGenerator . . . . .	241
28.13 ItemLabelAnchor . . . . .	242
28.14 ItemLabelPosition . . . . .	243
28.15 PieSectionLabelGenerator . . . . .	243
28.16 PieToolTipGenerator . . . . .	244
28.17 StandardCategoryLabelGenerator . . . . .	244
28.18 StandardCategoryToolTipGenerator . . . . .	246
28.19 StandardContourToolTipGenerator . . . . .	247
28.20 StandardPieItemLabelGenerator . . . . .	247
28.21 StandardXYLabelGenerator . . . . .	248
28.22 StandardXYZToolTipGenerator . . . . .	249
28.23 StandardXYZToolTipGenerator . . . . .	250
28.24 SymbolicXYItemLabelGenerator . . . . .	251
28.25 XYLabelGenerator . . . . .	251
28.26 XYZToolTipGenerator . . . . .	251
28.27 XYZToolTipGenerator . . . . .	252
<b>29 Package: org.jfree.chart.needle</b>	<b>253</b>
29.1 Overview . . . . .	253
29.2 ArrowNeedle . . . . .	254
29.3 LineNeedle . . . . .	254
29.4 LongNeedle . . . . .	255
29.5 MeterNeedle . . . . .	255
29.6 PinNeedle . . . . .	256
29.7 PlumNeedle . . . . .	256
29.8 PointerNeedle . . . . .	257
29.9 ShipNeedle . . . . .	257
29.10 WindNeedle . . . . .	258
<b>30 Package: org.jfree.chart.plot</b>	<b>259</b>
30.1 Overview . . . . .	259
30.2 CategoryPlot . . . . .	259
30.3 CombinedDomainCategoryPlot . . . . .	264
30.4 CombinedDomainXYPlot . . . . .	265
30.5 CombinedRangeCategoryPlot . . . . .	267
30.6 CombinedRangeXYPlot . . . . .	268
30.7 CompassPlot . . . . .	270
30.8 ContourPlot . . . . .	270
30.9 ContourPlotUtilities . . . . .	270
30.10 ContourValuePlot . . . . .	270
30.11 CrosshairState . . . . .	270
30.12 DatasetRenderingOrder . . . . .	271
30.13 DefaultDrawingSupplier . . . . .	272
30.14 DialShape . . . . .	272
30.15 DrawingSupplier . . . . .	272
30.16 FastScatterPlot . . . . .	273
30.17 IntervalMarker . . . . .	274

30.18Marker . . . . .	275
30.19MeterPlot . . . . .	275
30.20MultiplePiePlot . . . . .	277
30.21PieLabelDistributor . . . . .	278
30.22PieLabelRecord . . . . .	278
30.23PiePlot . . . . .	278
30.24PiePlot3D . . . . .	283
30.25PiePlotState . . . . .	284
30.26Plot . . . . .	284
30.27PlotOrientation . . . . .	287
30.28PlotRenderingInfo . . . . .	287
30.29PlotState . . . . .	288
30.30PolarPlot . . . . .	288
30.31RingPlot . . . . .	288
30.32ThermometerPlot . . . . .	289
30.33ValueAxisPlot . . . . .	292
30.34ValueMarker . . . . .	292
30.35WaferMapPlot . . . . .	293
30.36XYPlot . . . . .	293
<b>31 Package: org.jfree.chart.renderer</b>	<b>299</b>
31.1 Overview . . . . .	299
31.2 AbstractRenderer . . . . .	299
31.3 AreaRendererEndType . . . . .	302
31.4 DefaultPolarItemRenderer . . . . .	302
31.5 NotOutlierException . . . . .	302
31.6 Outlier . . . . .	303
31.7 OutlierList . . . . .	303
31.8 OutlierListCollection . . . . .	303
31.9 PolarItemRenderer . . . . .	303
31.10 RendererState . . . . .	303
31.11 WaferMapRenderer . . . . .	304
<b>32 Package: org.jfree.chart.renderer.category</b>	<b>305</b>
32.1 Overview . . . . .	305
32.2 AbstractCategoryItemRenderer . . . . .	305
32.3 AreaRenderer . . . . .	308
32.4 BarRenderer . . . . .	308
32.5 BarRenderer3D . . . . .	313
32.6 BoxAndWhiskerRenderer . . . . .	314
32.7 CategoryItemRenderer . . . . .	315
32.8 CategoryItemRendererState . . . . .	318
32.9 CategoryStepRenderer . . . . .	318
32.10 DefaultCategoryItemRenderer . . . . .	319
32.11 GanttRenderer . . . . .	319
32.12 GroupedStackedBarRenderer . . . . .	320
32.13 IntervalBarRenderer . . . . .	322
32.14 LayeredBarRenderer . . . . .	322
32.15 LevelRenderer . . . . .	322
32.16 LineAndShapeRenderer . . . . .	323

32.17MinMaxCategoryRenderer . . . . .	325
32.18StackedAreaRenderer . . . . .	325
32.19StackedBarRenderer . . . . .	326
32.20StackedBarRenderer3D . . . . .	326
32.21StatisticalBarRenderer . . . . .	327
32.22WaterfallBarRenderer . . . . .	328
<b>33 Package: org.jfree.chart.renderer.xy</b>	<b>329</b>
33.1 Overview . . . . .	329
33.2 AbstractXYItemRenderer . . . . .	329
33.3 CandlestickRenderer . . . . .	331
33.4 ClusteredXYBarRenderer . . . . .	332
33.5 CyclicXYItemRenderer . . . . .	333
33.6 DefaultXYItemRenderer . . . . .	333
33.7 HighLow . . . . .	333
33.8 HighLowRenderer . . . . .	333
33.9 StackedXYAreaRenderer . . . . .	334
33.10StackedXYBarRenderer . . . . .	334
33.11StandardXYItemRenderer . . . . .	336
33.12WindItemRenderer . . . . .	337
33.13XYAreaRenderer . . . . .	337
33.14XYBarRenderer . . . . .	339
33.15XYBoxAndWhiskerRenderer . . . . .	341
33.16XYBubbleRenderer . . . . .	342
33.17XYDifferenceRenderer . . . . .	342
33.18XYDotRenderer . . . . .	343
33.19XYItemRenderer . . . . .	344
33.20XYItemRendererState . . . . .	346
33.21XYLineAndShapeRenderer . . . . .	346
33.22XYStepRenderer . . . . .	348
33.23XYStepAreaRenderer . . . . .	349
33.24YIntervalRenderer . . . . .	349
<b>34 Package: org.jfree.chart.servlet</b>	<b>351</b>
34.1 Overview . . . . .	351
34.2 ChartDeleter . . . . .	351
34.3 DisplayChart . . . . .	351
34.4 ServletUtilities . . . . .	351
<b>35 Package: org.jfree.chart.title</b>	<b>353</b>
35.1 Overview . . . . .	353
35.2 Events . . . . .	353
35.3 DateTitle . . . . .	353
35.4 ImageTitle . . . . .	354
35.5 LegendTitle . . . . .	354
35.6 TextTitle . . . . .	354
35.7 Title . . . . .	355

CONTENTS	10
----------	----

<b>36 Package: org.jfree.chart.ui</b>	<b>358</b>
36.1 Introduction . . . . .	358
36.2 AxisPropertyEditPanel . . . . .	358
36.3 ChartPropertyEditPanel . . . . .	359
36.4 ColorBarPropertyEditPanel . . . . .	359
36.5 ColorPalette . . . . .	359
36.6 GreyPalette . . . . .	359
36.7 LegendPropertyEditPanel . . . . .	359
36.8 NumberAxisPropertyEditPanel . . . . .	360
36.9 PaletteChooserPanel . . . . .	360
36.10PaletteSample . . . . .	360
36.11PlotPropertyEditPanel . . . . .	360
36.12RainbowPalette . . . . .	361
36.13TitlePropertyEditPanel . . . . .	361
<b>37 Package: org.jfree.chart.urls</b>	<b>362</b>
37.1 Overview . . . . .	362
37.2 CategoryURLGenerator . . . . .	362
37.3 CustomPieURLGenerator . . . . .	363
37.4 CustomXYURLGenerator . . . . .	363
37.5 PieURLGenerator . . . . .	363
37.6 StandardCategoryURLGenerator . . . . .	364
37.7 StandardPieURLGenerator . . . . .	365
37.8 StandardXYURLGenerator . . . . .	365
37.9 StandardXYZURLGenerator . . . . .	365
37.10TimeSeriesURLGenerator . . . . .	365
37.11XYURLGenerator . . . . .	365
37.12XYZURLGenerator . . . . .	366
<b>38 Package: org.jfree.data</b>	<b>367</b>
38.1 Introduction . . . . .	367
38.2 DefaultKeyedValue . . . . .	367
38.3 DefaultKeyedValues . . . . .	368
38.4 DefaultKeyedValues2D . . . . .	368
38.5 DomainInfo . . . . .	369
38.6 DomainOrder . . . . .	369
38.7 KeyedObject . . . . .	370
38.8 KeyedObjects . . . . .	370
38.9 KeyedObjects2D . . . . .	370
38.10KeyValue . . . . .	370
38.11KeyValueComparator . . . . .	370
38.12KeyValueComparatorType . . . . .	370
38.13KeyedValues . . . . .	371
38.14KeyedValues2D . . . . .	371
38.15KeyToGroupMap . . . . .	372
38.16Range . . . . .	373
38.17RangeInfo . . . . .	375
38.18Value . . . . .	376
38.19Values . . . . .	376
38.20Values2D . . . . .	377

<b>39 Package: org.jfree.data.category</b>	<b>378</b>
39.1 Introduction . . . . .	378
39.2 CategoryDataset . . . . .	378
39.3 CategoryToPieDataset . . . . .	379
39.4 DefaultCategoryDataset . . . . .	379
39.5 DefaultIntervalCategoryDataset . . . . .	380
39.6 IntervalCategoryDataset . . . . .	380
<b>40 Package: org.jfree.data.contour</b>	<b>382</b>
40.1 Introduction . . . . .	382
40.2 ContourDataset . . . . .	382
40.3 DefaultContourDataset . . . . .	383
40.4 NonGridContourDataset . . . . .	383
<b>41 Package: org.jfree.data.function</b>	<b>384</b>
41.1 Introduction . . . . .	384
41.2 Function2D . . . . .	384
41.3 LineFunction2D . . . . .	385
41.4 NormalDistributionFunction2D . . . . .	385
41.5 PowerFunction2D . . . . .	385
<b>42 Package: org.jfree.data.gantt</b>	<b>387</b>
42.1 Introduction . . . . .	387
42.2 GanttCategoryDataset . . . . .	387
42.3 Task . . . . .	388
42.4 TaskSeries . . . . .	390
42.5 TaskSeriesCollection . . . . .	391
<b>43 Package: org.jfree.data.general</b>	<b>394</b>
43.1 Introduction . . . . .	394
43.2 AbstractDataset . . . . .	394
43.3 AbstractSeriesDataset . . . . .	395
43.4 CombinationDataset . . . . .	396
43.5 CombinedDataset . . . . .	396
43.6 Dataset . . . . .	397
43.7 DatasetChangeEvent . . . . .	397
43.8 DatasetChangeListener . . . . .	398
43.9 DatasetGroup . . . . .	398
43.10DatasetUtilities . . . . .	398
43.11DataUtilities . . . . .	403
43.12DefaultKeyedValueDataset . . . . .	403
43.13DefaultKeyedValuesDataset . . . . .	403
43.14DefaultKeyedValues2DDataset . . . . .	403
43.15DefaultPieDataset . . . . .	404
43.16DefaultValueDataset . . . . .	405
43.17KeyedValueDataset . . . . .	406
43.18KeyedValuesDataset . . . . .	406
43.19KeyedValues2DDataset . . . . .	407
43.20PieDataset . . . . .	407
43.21Series . . . . .	407

43.22SeriesChangeEvent . . . . .	408
43.23SeriesChangeListener . . . . .	408
43.24SeriesDataset . . . . .	409
43.25SeriesException . . . . .	409
43.26SubSeriesDataset . . . . .	409
43.27ValueDataset . . . . .	409
43.28WaferMapDataset . . . . .	410
<b>44 Package: org.jfree.data.jdbc</b>	<b>411</b>
44.1 Introduction . . . . .	411
44.2 JDBCCategoryDataset . . . . .	411
44.3 JDBCPieDataset . . . . .	412
44.4 JDBCXYDataset . . . . .	413
<b>45 Package: org.jfree.data.statistics</b>	<b>415</b>
45.1 Introduction . . . . .	415
45.2 BoxAndWhiskerCalculator . . . . .	415
45.3 BoxAndWhiskerCategoryDataset . . . . .	416
45.4 BoxAndWhiskerItem . . . . .	417
45.5 BoxAndWhiskerXYDataset . . . . .	418
45.6 DefaultBoxAndWhiskerCategoryDataset . . . . .	419
45.7 DefaultBoxAndWhiskerXYDataset . . . . .	419
45.8 DefaultStatisticalCategoryDataset . . . . .	419
45.9 HistogramBin . . . . .	421
45.10HistogramDataset . . . . .	421
45.11HistogramType . . . . .	422
45.12MeanAndStandardDeviation . . . . .	422
45.13Regression . . . . .	423
45.14StatisticalCategoryDataset . . . . .	424
45.15Statistics . . . . .	424
<b>46 Package: org.jfree.data.time</b>	<b>426</b>
46.1 Introduction . . . . .	426
46.2 DateRange . . . . .	426
46.3 Day . . . . .	427
46.4 DynamicTimeSeriesCollection . . . . .	428
46.5 FixedMillisecond . . . . .	431
46.6 Hour . . . . .	432
46.7 Millisecond . . . . .	433
46.8 Minute . . . . .	434
46.9 Month . . . . .	435
46.10MovingAverage . . . . .	436
46.11Quarter . . . . .	437
46.12RegularTimePeriod . . . . .	439
46.13Second . . . . .	441
46.14SimpleTimePeriod . . . . .	442
46.15TimePeriod . . . . .	442
46.16TimePeriodAnchor . . . . .	443
46.17TimePeriodFormatException . . . . .	443
46.18TimePeriodValue . . . . .	443

46.19TimePeriodValues . . . . .	444
46.20TimePeriodValuesCollection . . . . .	444
46.21TimeSeries . . . . .	445
46.22TimeSeriesCollection . . . . .	447
46.23TimeSeriesDataItem . . . . .	450
46.24TimeSeriesTableModel . . . . .	451
46.25TimeTableXYDataset . . . . .	451
46.26Week . . . . .	453
46.27Year . . . . .	455
<b>47 Package: org.jfree.data.xml</b>	<b>457</b>
47.1 Introduction . . . . .	457
47.2 Usage . . . . .	457
47.3 CategoryDatasetHandler . . . . .	457
47.4 CategorySeriesHandler . . . . .	458
47.5 DatasetReader . . . . .	459
47.6 DatasetTags . . . . .	459
47.7 ItemHandler . . . . .	459
47.8 KeyHandler . . . . .	459
47.9 PieDatasetHandler . . . . .	460
47.10RootHandler . . . . .	461
47.11ValueHandler . . . . .	461
<b>48 Package: org.jfree.data.xy</b>	<b>462</b>
48.1 Introduction . . . . .	462
48.2 AbstractIntervalXYDataset . . . . .	462
48.3 AbstractXYZDataset . . . . .	462
48.4 AbstractXYZDataset . . . . .	463
48.5 CategoryTableXYDataset . . . . .	463
48.6 DefaultHighLowDataset . . . . .	465
48.7 DefaultOHLCDataset . . . . .	465
48.8 DefaultTableXYDataset . . . . .	467
48.9 DefaultWindDataset . . . . .	469
48.10IntervalXYDataset . . . . .	470
48.11IntervalXYDelegate . . . . .	470
48.12IntervalXYZDataset . . . . .	472
48.13MatrixSeries . . . . .	473
48.14MatrixSeriesCollection . . . . .	473
48.15NormalizedMatrixSeries . . . . .	473
48.16OHLCDataItem . . . . .	473
48.17OHLCDataset . . . . .	474
48.18SignalsDataset . . . . .	475
48.19TableXYDataset . . . . .	475
48.20WindDataset . . . . .	475
48.21XisSymbolic . . . . .	476
48.22XYBarDataset . . . . .	476
48.23XYDataItem . . . . .	476
48.24XYDataset . . . . .	477
48.25XYDatasetTableModel . . . . .	478
48.26XYSeries . . . . .	479

<i>CONTENTS</i>	14
48.27XYSeriesCollection . . . . .	482
48.28XYZDataset . . . . .	484
48.29YisSymbolic . . . . .	485
<b>A JCommon</b>	<b>486</b>
A.1 Introduction . . . . .	486
A.2 Align . . . . .	486
A.3 PublicCloneable . . . . .	487
A.4 RectangleAnchor . . . . .	487
A.5 RectangleEdge . . . . .	487
A.6 RectangleInsets . . . . .	488
A.7 Spacer . . . . .	489
A.8 TextAnchor . . . . .	490
A.9 UnitType . . . . .	491
<b>B The GNU Lesser General Public License</b>	<b>492</b>
B.1 Introduction . . . . .	492
B.2 The License . . . . .	492
B.3 Frequently Asked Questions . . . . .	499

# 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

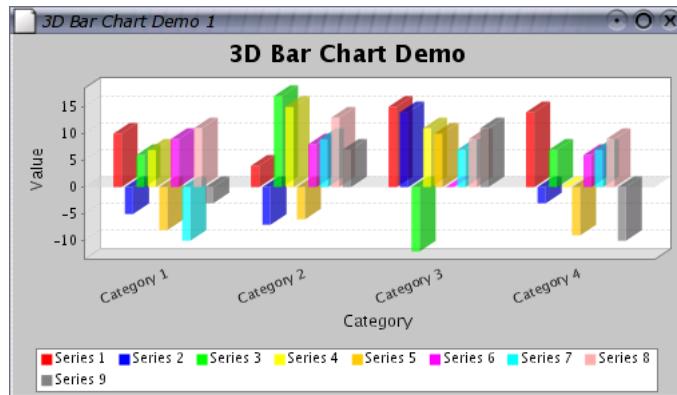


Figure 1.1: A sample chart

complete source code subject to the terms of the GNU Lesser General Public Licence (see Appendix B for details).

#### 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;
- 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;
- 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 1.2.2 or later).

### 1.1.3 Home Page

The JFreeChart home page can be found at:

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

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.

## 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.

### 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:

Richard Atkinson, David Berry, Anthony Boulestreau, Jeremy Bowman, Daniel Bridenbecker, Nicolas Brodu, David Browning, Søren Caspersen, Chuanhao Chiu, Pascal Collet, Martin Cordova, Paolo Cova, Michael Duffy, Jonathan Gabbai, Serge V. Grachov, Hans-Jurgen Greiner, Joao Guilherme Del Valle, Aiman Han, Jon Iles, Wolfgang Irler, Xun Kang, Bill Kelemen, Norbert Kiesel, Gideon Krause, Arnaud Lelievre, David Li, Tin Luu, Craig MacFarlane, Achilleus Mantzios, Thomas Meier, Aaron Metzger, Jim Moore, Jonathan Nash, Barak Naveh, David M. O'Donnell, Krzysztof Paz, Tomer Peretz, Andrzej Porebski, Luke Quinane, Viktor Rajewski, Eduardo Ramalho, Michael Rauch, Cameron Riley, Dan Rivett, Michel Santos, Thierry Saura, Andreas Schneider, Jean-Luc Schwab, Bryan Scott, Roger Studner, Irv Thomae, Eric Thomas, Rich Unger, Daniel van Enckevort, Laurence Vanhelsuwe, Sylvain Vieujoet, Jelai Wang, Mark Watson, Alex Weber, Matthew Wright, 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-1.0.0-pre2-demo.jar
```

The complete source code for the demo application is available to purchasers of the JFreeChart Developer Guide.

### 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.

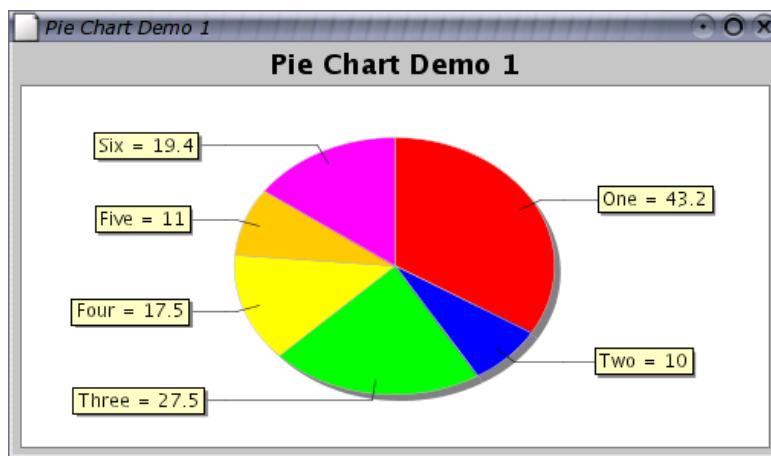


Figure 2.1: A simple pie chart

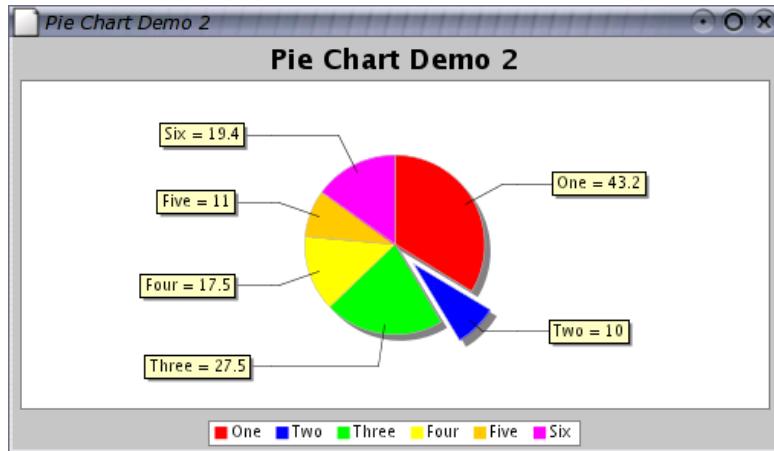


Figure 2.2: A pie chart with an “exploded” section

Individual pie sections can be “exploded”, as shown in figure 2.2. You can also display pie charts with a 3D effect, as shown in figure 2.3.

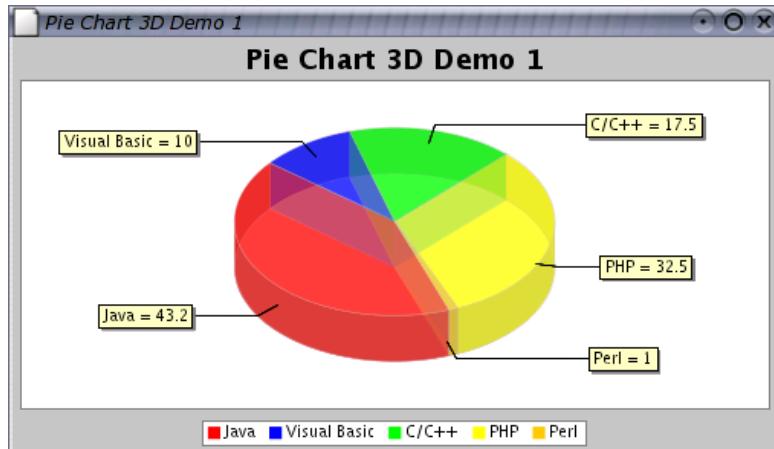


Figure 2.3: A pie chart drawn with a 3D effect

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.

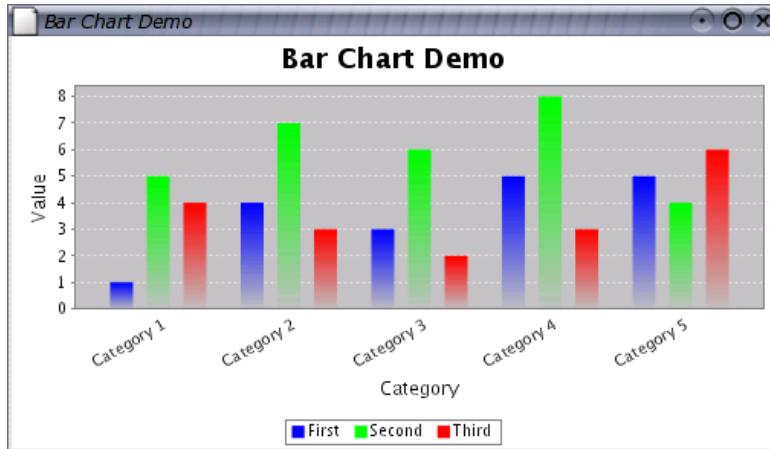


Figure 2.4: A vertical bar chart

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

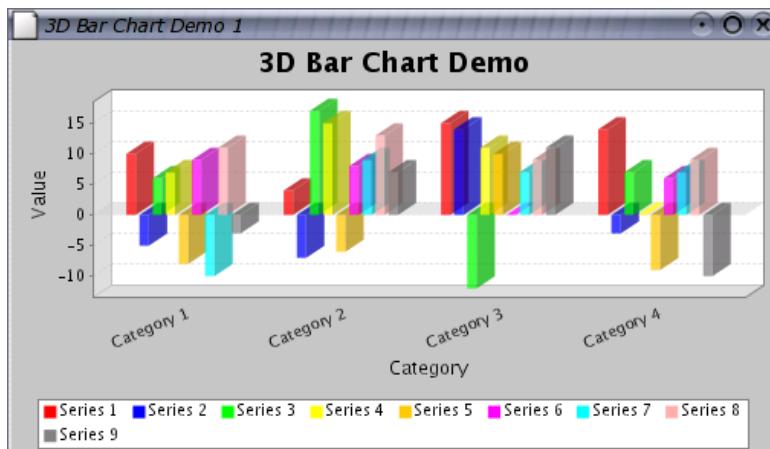


Figure 2.5: A bar chart with 3D effect

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

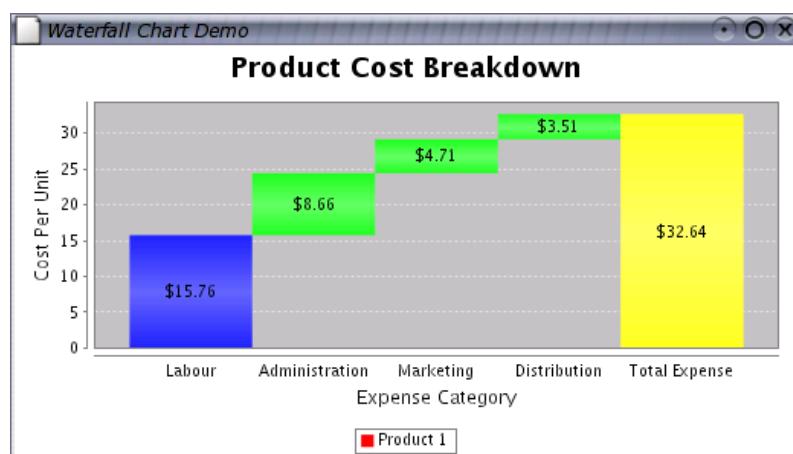


Figure 2.6: A waterfall chart

## 2.4 Line Chart

The *line chart* can be generated using the same `CategoryDataset` that is used for the bar charts—figure 2.7 shows an example.

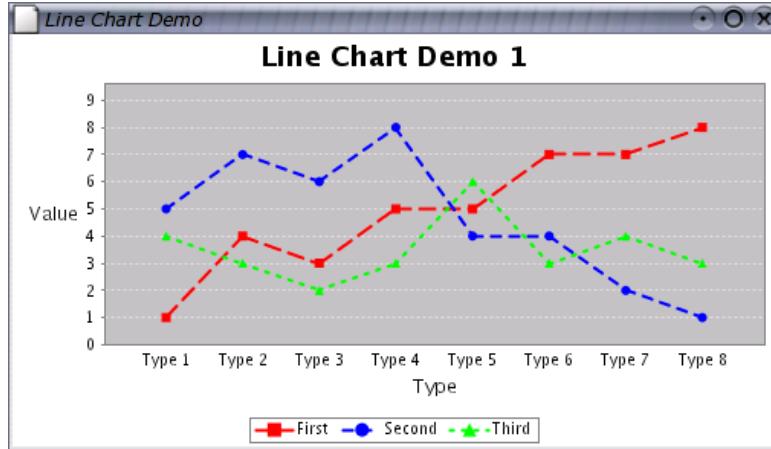


Figure 2.7: A line chart

## 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 2.8.

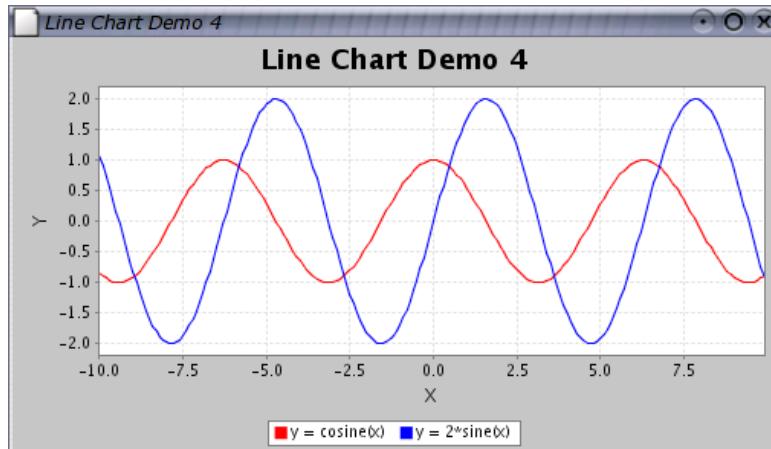


Figure 2.8: A line chart

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 2.9.

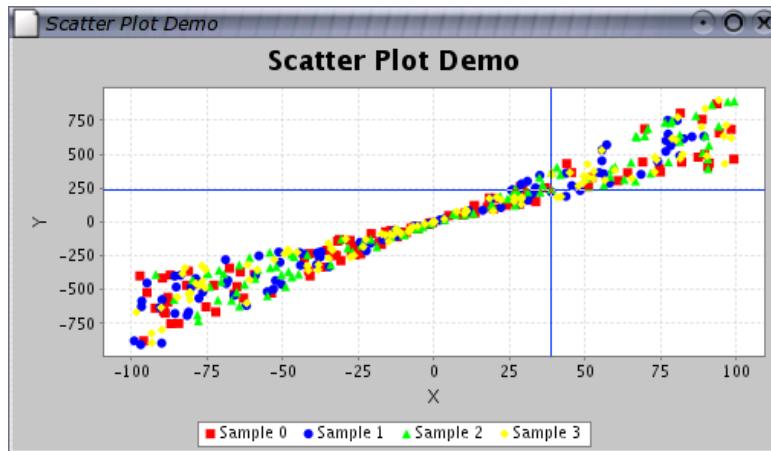


Figure 2.9: A scatter plot

## 2.6 Time Series Charts

JFreeChart supports *time series charts*, as shown in figure 2.10.

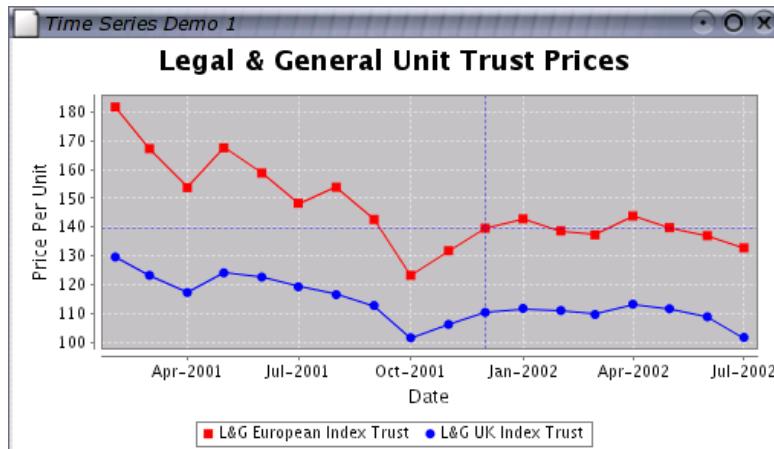


Figure 2.10: A time series chart

It is straightforward to add a moving average line to a time series chart—see figure 2.11 for an example.

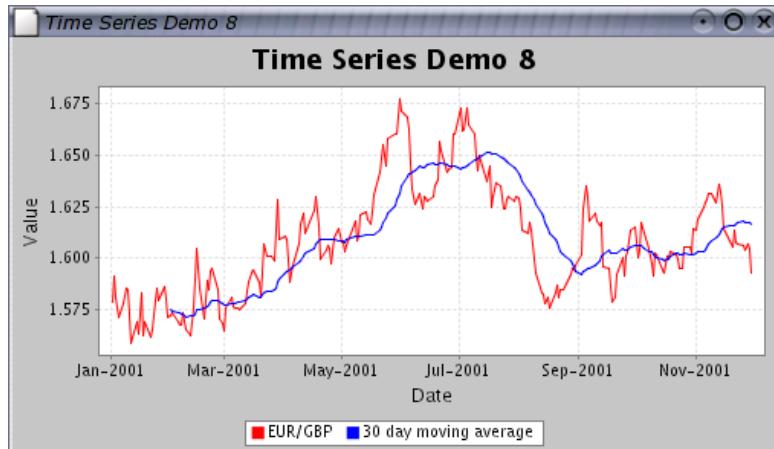


Figure 2.11: A time series chart with a moving average

Using a [HighLowDataset](#) (an extension of [XYDataset](#)) you can display *high-low-open-close* data, see figure 2.12 for an example.

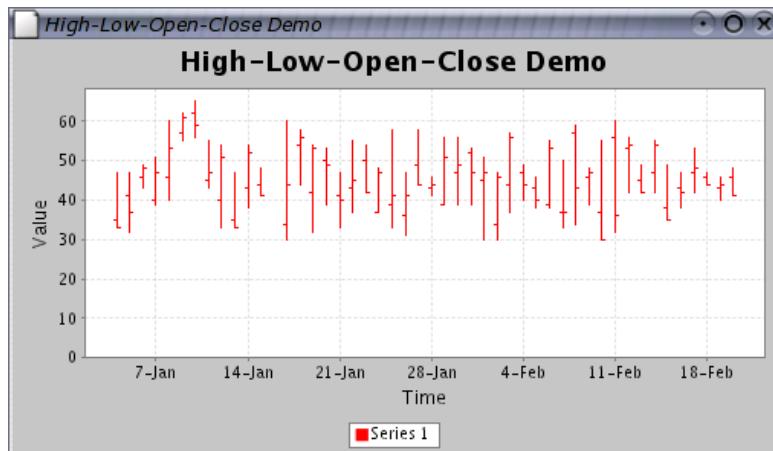


Figure 2.12: A high-low-open-close chart

## 2.7 Histograms

Histograms can be generated using an [IntervalXYDataset](#) (another extension of [XYDataset](#)), see figure 2.13 for an example.

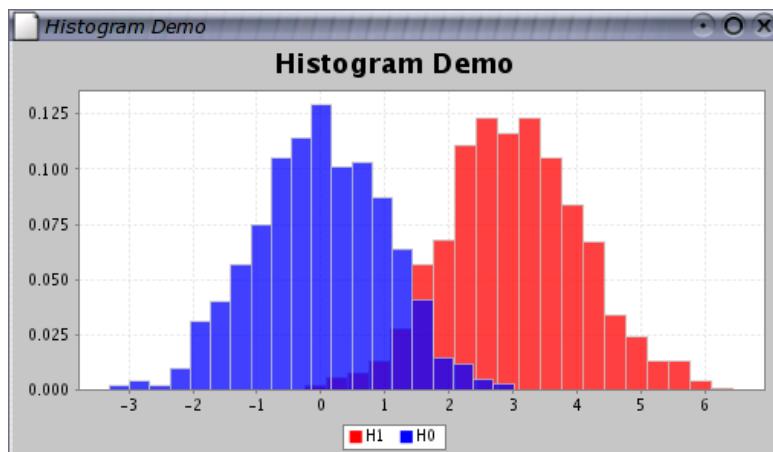


Figure 2.13: A histogram

## 2.8 Area Charts

You can generate an *area chart* for data in a [CategoryDataset](#) or an [XYDataset](#). Figure 2.14 shows an example.

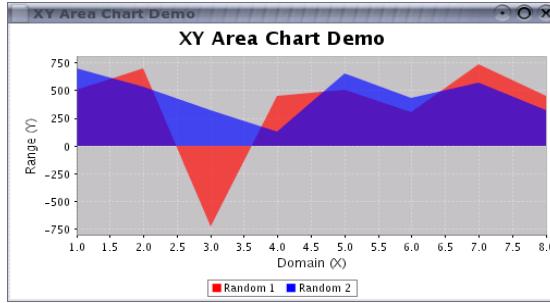


Figure 2.14: An area chart

JFreeChart also supports the creation of *stacked area charts* as shown in figure 2.15.

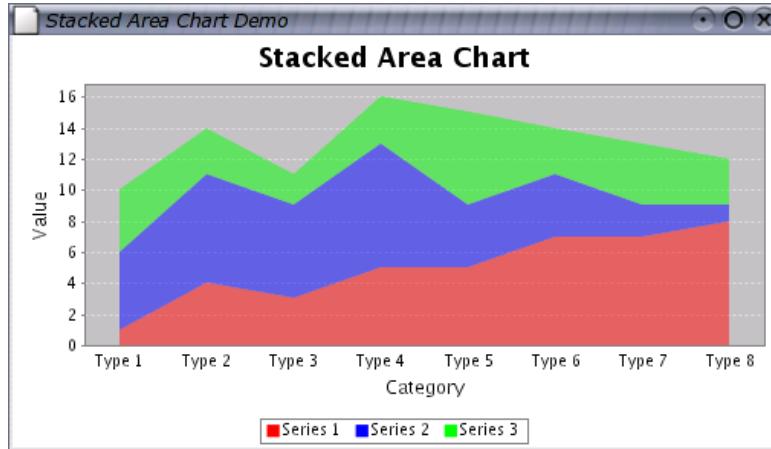


Figure 2.15: A stacked area chart

## 2.9 Difference Chart

A *difference chart* highlights the difference between two series (see figure 2.16). A second example, shown in figure 2.17 shows how a date axis can be used for the range values.

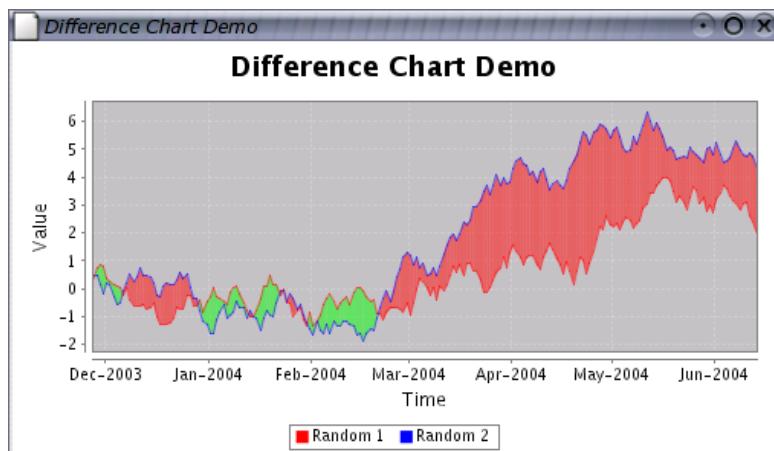


Figure 2.16: A difference chart

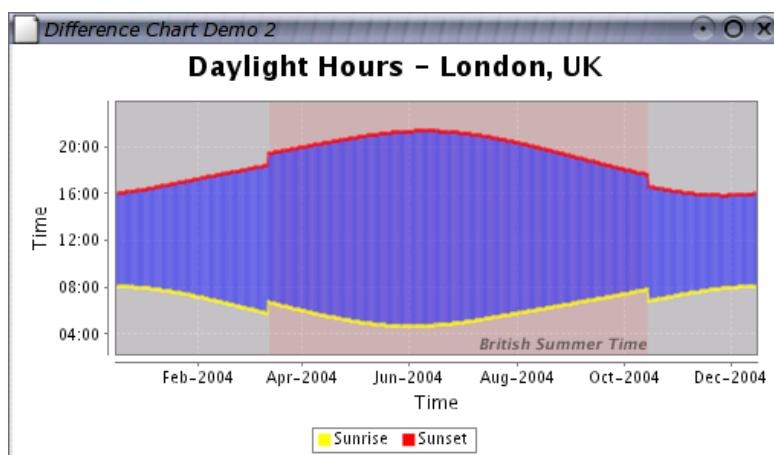


Figure 2.17: A difference chart with times on the range axis

## 2.10 Step Chart

A *step chart* displays numerical data as a sequence of “steps”—an example is shown in figure 2.18.

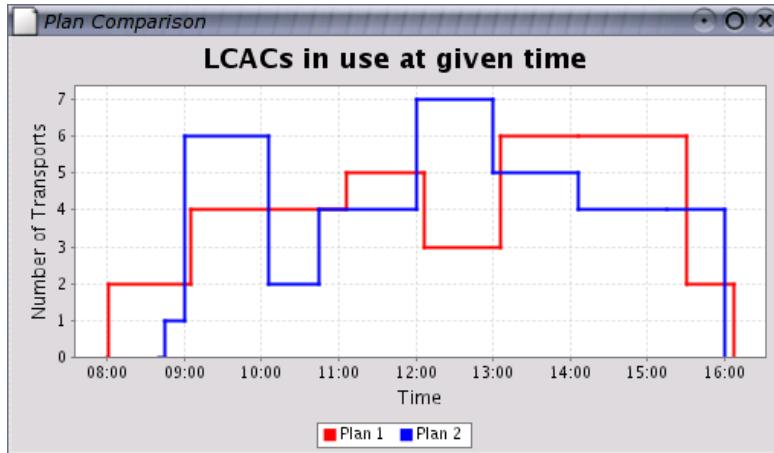


Figure 2.18: A step chart

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 2.19.

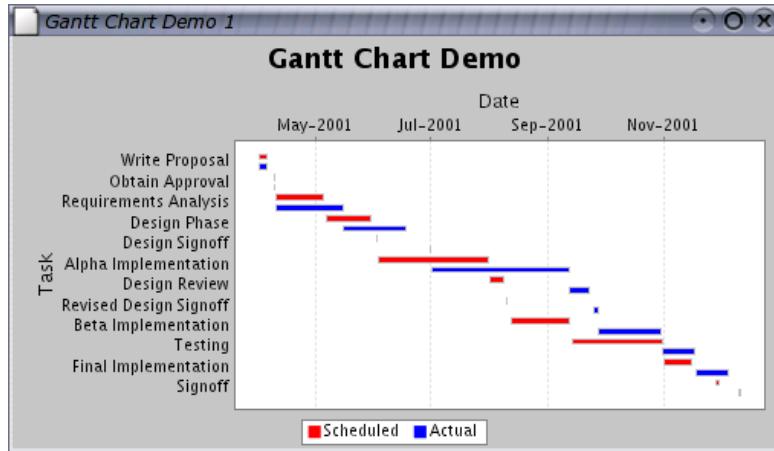


Figure 2.19: A Gantt chart

Another example, showing subtasks and progress indicators, is shown in figure 2.20.

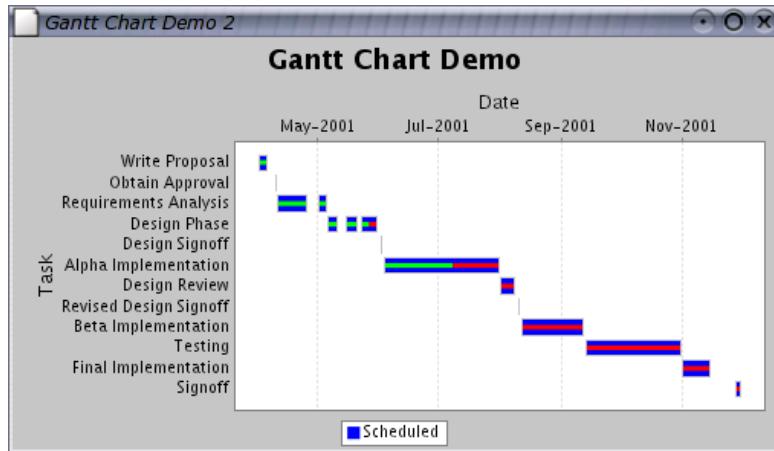


Figure 2.20: A Gantt chart with progress indicators

## 2.12 Multiple Axis Charts

JFreeChart has support for charts with multiple axes. Figure 2.21 shows a *price-volume chart* that demonstrates this feature.



Figure 2.21: A price-volume chart

This feature is supported by the `CategoryPlot` and `XYPlot` classes. Figure 2.22 shows an example with four range axes.

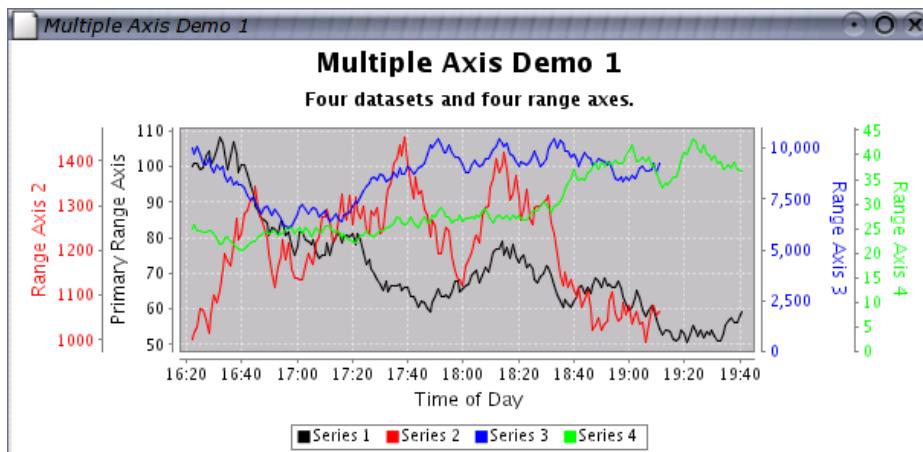


Figure 2.22: A chart with multiple axes

## 2.13 Combined and Overlaid Charts

JFreeChart supports combined and overlaid charts. Figure 2.23 shows a line chart overlaid on top of a bar chart.

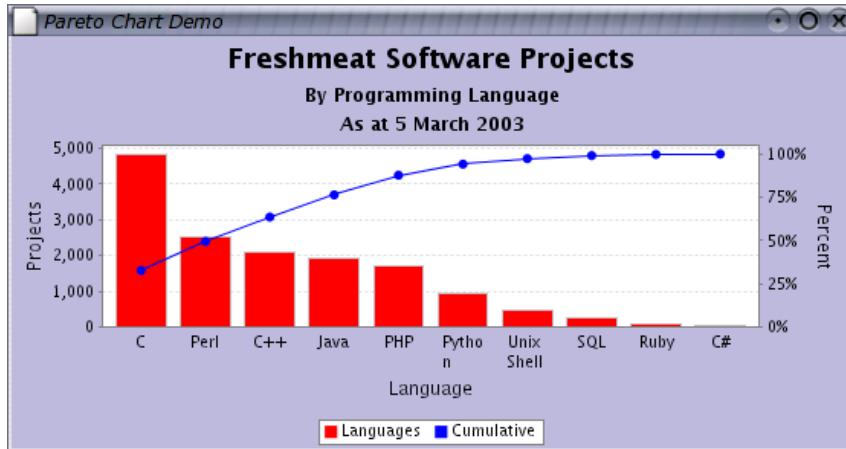


Figure 2.23: An overlaid chart

It is possible to combine several charts that share a common domain axis, as shown in figure 2.24.

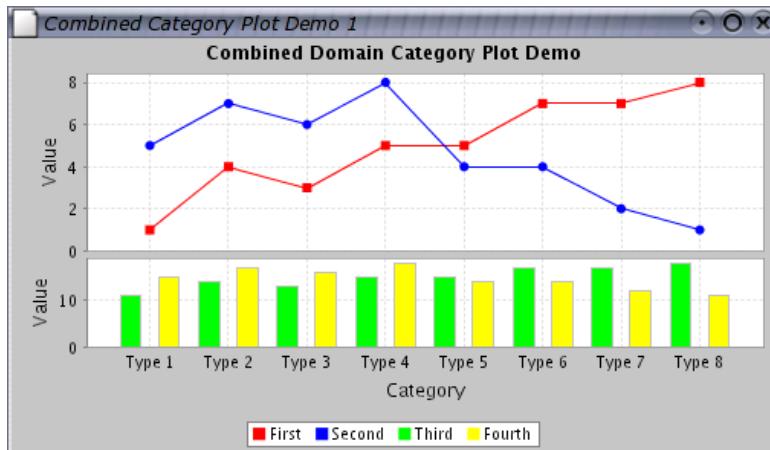


Figure 2.24: A chart with a combined domain

In a similar way, JFreeChart can combine several charts that share a common range axis, see figure 2.25.

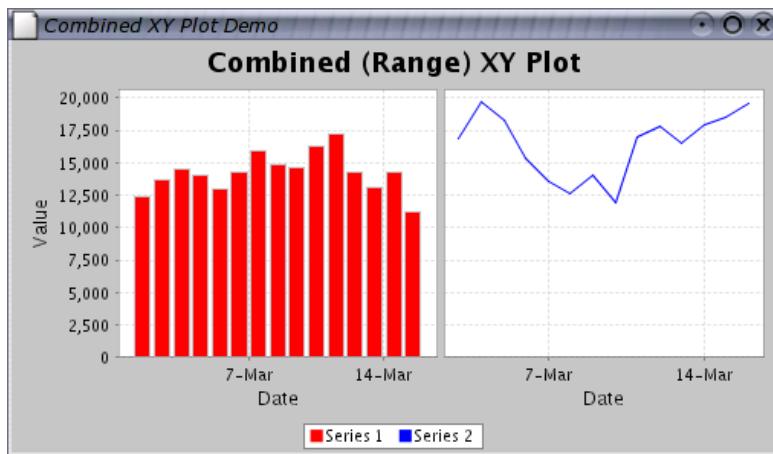


Figure 2.25: A chart with a combined range

## 2.14 Future Development

JFreeChart is *free software*,<sup>1</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/index.html>

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

---

<sup>1</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/index.php>

There are two versions of the JFreeChart download:

File:	Description:
jfreechart-1.0.0-pre2.tar.gz	JFreeChart for Linux/Unix.
jfreechart-1.0.0-pre2.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 `1.0.0-pre2`). 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/index.php>

There is a separate PDF document for JCommon, which includes full instructions for downloading and unpacking the files.

### 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-1.0.0-pre2`) 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-1.0.0-pre2.tar.gz
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-1.0.0-pre2`.

#### 3.3.2 Unpacking on Windows

To extract the files from the download on Windows, enter the following command:

```
jar -xvf jfreechart-1.0.0-pre2.zip
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-1.0.0-pre2`.

#### 3.3.3 The Files

The top-level directory (`jfreechart-1.0.0-pre2`) contains the files and directories listed in the following table:

File/Directory:	Description:
<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>CHANGELOG.txt</code>	A log of changes made to JFreeChart since the previous release.
<code>checkstyle</code>	A directory containing several Checkstyle property files. These define the coding conventions used in the JFreeChart source code.
<code>jfreechart-1.0.0-pre2.jar</code>	The JFreeChart runtime jar file.
<code>jfreechart-1.0.0-pre2-demo.jar</code>	A runnable jar file containing demo applications.
<code>lib</code>	A directory containing libraries used by JFreeChart.
<code>licence-LGPL.txt</code>	The GNU LGPL.
<code>README.txt</code>	Important information - <i>read this first!</i>
<code>source</code>	A directory containing the source code for JFreeChart.

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-1.0.0-pre2-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.

## 3.5 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/>

## 3.6 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-1.0.0-pre2` 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.

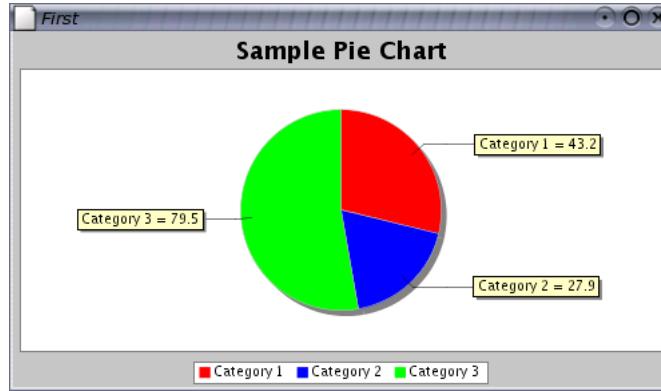
Each of the three steps outlined above is described, along with sample code, in the following sections.

#### 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)



*Figure 4.1: A pie chart created using `First.java`*

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:

---

<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 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) {

        // 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

## Bar Charts

### 5.1 Introduction

This section describes the *bar charts* that can be created with JFreeChart. Most bar charts are created using data obtained via the [CategoryDataset](#) interface (it is also possible to use the [IntervalXYDataset](#) interface, but more on that later).

### 5.2 A Bar Chart

#### 5.2.1 Overview

A *bar chart* is created using data from a [CategoryDataset](#), and represents each data item as a bar where the length of the bar is equal to the data value. This section presents a sample application that generates the chart shown in figure 5.1.

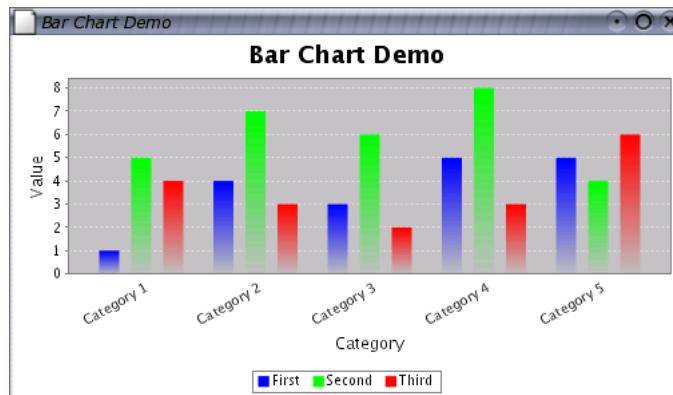


Figure 5.1: A sample bar chart

The full source code ([BarChartDemo1.java](#)) for this demo is available for download from the same URL as the JFreeChart Developer Guide.

### 5.2.2 The Dataset

The first step in generating the chart is to create a dataset. You can use *any class* that implements the [CategoryDataset](#) interface—for the example, we have used the [DefaultCategoryDataset](#) class (included in the JFreeChart distribution):

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

    // row keys...
    String series1 = "First";
    String series2 = "Second";
    String series3 = "Third";

    // column keys...
    String category1 = "Category 1";
    String category2 = "Category 2";
    String category3 = "Category 3";
    String category4 = "Category 4";
    String category5 = "Category 5";

    // create the dataset...
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();

    dataset.addValue(1.0, series1, category1);
    dataset.addValue(4.0, series1, category2);
    dataset.addValue(3.0, series1, category3);
    dataset.addValue(5.0, series1, category4);
    dataset.addValue(5.0, series1, category5);

    dataset.addValue(5.0, series2, category1);
    dataset.addValue(7.0, series2, category2);
    dataset.addValue(6.0, series2, category3);
    dataset.addValue(8.0, series2, category4);
    dataset.addValue(4.0, series2, category5);

    dataset.addValue(4.0, series3, category1);
    dataset.addValue(3.0, series3, category2);
    dataset.addValue(2.0, series3, category3);
    dataset.addValue(3.0, series3, category4);
    dataset.addValue(6.0, series3, category5);

    return dataset;
}
```

Notice that we have used `String` objects as the row and column keys for the data values. You can use *any class* that implements the `Comparable` interface as the keys for your data values.

### 5.2.3 Constructing the Chart

The `createBarChart()` method in the [ChartFactory](#) class provides a convenient way to create the chart:<sup>1</sup>

```
// create the chart...
JFreeChart chart = ChartFactory.createBarChart(
    "Bar Chart Demo",           // chart title
    "Category",                 // domain axis label
    "Value",                    // range axis label
```

---

<sup>1</sup>Take a look at the source code for this method, if you are interested to know how the bar chart is constructed from the components (axes, plots, renderers etc.) in the JFreeChart library.

```

        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, renderer and tooltip generator. The `dataset` is the one created in the previous section.

#### 5.2.4 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, several attributes are modified:

- the chart background color;
- the “auto tick units” on the range axis (so that the tick labels always display integer values);
- gradient paint is used for the series colors;

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

```
// set the background color for the chart...
chart.setBackgroundPaint(new Color(0xBBBBDD));
```

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

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

The range axis is modified so that the tick units are always integers:

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

The bar renderer is modified so that bar outlines are not drawn, and `GradientPaint` instances are used for the series colors:

```

// disable bar outlines...
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setDrawBarOutline(false);

// set up gradient paints for series...
GradientPaint gp0 = new GradientPaint(
    0.0f, 0.0f, Color.blue,
    0.0f, 0.0f, Color.lightGray
);
GradientPaint gp1 = new GradientPaint(
    0.0f, 0.0f, Color.green,
    0.0f, 0.0f, Color.lightGray
);
GradientPaint gp2 = new GradientPaint(
    0.0f, 0.0f, Color.red,
    0.0f, 0.0f, Color.lightGray
);
renderer.setSeriesPaint(0, gp0);
renderer.setSeriesPaint(1, gp1);
renderer.setSeriesPaint(2, gp2);

```

Refer to the source code, Javadoc API documentation or elsewhere in this document for details of the other customisations that you can make to a bar plot.

### 5.2.5 The Complete Program

The code for the demonstration application is presented in full, complete with the import statements. You should find this code included in the JFreeChart Premium Demo distribution.

```

/*
 * BarChartDemo1.java
 *
 * (C) Copyright 2002-2004, by Object Refinery Limited.
 *
 */

package demo;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GradientPaint;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.CategoryAxis;
import org.jfree.chart.axis.CategoryLabelPositions;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.BarRenderer;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;

/**
 * A simple demonstration application showing how to create a bar chart.
 */
public class BarChartDemo1 extends ApplicationFrame {

    /**
     * Creates a new demo instance.
     *
     * @param title the frame title.
     */
    public BarChartDemo1(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() {

        // row keys...
        String series1 = "First";
        String series2 = "Second";
        String series3 = "Third";

        // column keys...
        String category1 = "Category 1";
        String category2 = "Category 2";
        String category3 = "Category 3";
        String category4 = "Category 4";
    }
}

```

```

String category5 = "Category 5";

// create the dataset...
DefaultCategoryDataset dataset = new DefaultCategoryDataset();

dataset.addValue(1.0, series1, category1);
dataset.addValue(4.0, series1, category2);
dataset.addValue(3.0, series1, category3);
dataset.addValue(5.0, series1, category4);
dataset.addValue(5.0, series1, category5);

dataset.addValue(5.0, series2, category1);
dataset.addValue(7.0, series2, category2);
dataset.addValue(6.0, series2, category3);
dataset.addValue(8.0, series2, category4);
dataset.addValue(4.0, series2, category5);

dataset.addValue(4.0, series3, category1);
dataset.addValue(3.0, series3, category2);
dataset.addValue(2.0, series3, category3);
dataset.addValue(3.0, series3, category4);
dataset.addValue(6.0, series3, category5);

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(
        "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?
    );

    // NOW DO SOME OPTIONAL CUSTOMISATION OF THE CHART...

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

    // get a reference to the plot for further customisation...
    CategoryPlot plot = chart.getCategoryPlot();
    plot.setBackgroundPaint(Color.lightGray);
    plot.setDomainGridlinePaint(Color.white);
    plot.setDomainGridlinesVisible(true);
    plot.setRangeGridlinePaint(Color.white);

    // set the range axis to display integers only...
    final NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
    rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

    // disable bar outlines...
    BarRenderer renderer = (BarRenderer) plot.getRenderer();
    renderer.setDrawBarOutline(false);

    // set up gradient paints for series...
    GradientPaint gp0 = new GradientPaint(
        0.0f, 0.0f, Color.blue,
        0.0f, 0.0f, new Color(0, 0, 64)
    );
    GradientPaint gp1 = new GradientPaint(

```

```

        0.0f, 0.0f, Color.green,
        0.0f, 0.0f, new Color(0, 64, 0)
    );
    GradientPaint gp2 = new GradientPaint(
        0.0f, 0.0f, Color.red,
        0.0f, 0.0f, new Color(64, 0, 0)
    );
    renderer.setSeriesPaint(0, gp0);
    renderer.setSeriesPaint(1, gp1);
    renderer.setSeriesPaint(2, gp2);

    CategoryAxis domainAxis = plot.getDomainAxis();
    domainAxis.setCategoryLabelPositions(
        CategoryLabelPositions.createUpRotationLabelPositions(Math.PI / 6.0)
    );
    // 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);
}

/**
 * Returns a description of the demo.
 *
 * @return A description.
 */
public static String getDemoDescription() {
    return "A bar chart.";
}

/**
 * Starting point for the demonstration application.
 *
 * @param args ignored.
 */
public static void main(String[] args) {
    BarChartDemo1 demo = new BarChartDemo1("Bar Chart Demo");
    demo.pack();
    RefineryUtilities.centerFrameOnScreen(demo);
    demo.setVisible(true);
}
}

```

## 5.3 Customising Bar Charts

This section describes some of the methods you can use to customise the appearance of bar charts.

### 5.3.1 Bar Colors

You can customise the colors used in a bar chart in the same way that you would for most other chart types. You need to obtain a reference to the renderer (the object responsible for drawing the bars in the chart) and set the series colors there:

```
CategoryPlot plot = chart.getCategoryPlot();
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setSeriesPaint(0, Color.red);
renderer.setSeriesPaint(1, Color.green);
renderer.setSeriesPaint(2, Color.blue);
```

The `setSeriesPaint()` method is defined in the `AbstractRenderer` class.

### 5.3.2 Bar Spacing

JFreeChart allows you to configure the way that bars are distributed along the category axis. There are settings for:

- the margin before the start of the first category;
- the margin between categories;
- the margin after the end of the last category;
- the gap between bars within a category;

The first three items are configured using the `CategoryAxis`:

```
CategoryPlot plot = chart.getCategoryPlot();
CategoryAxis axis = plot.getDomainAxis();
axis.setLowerMargin(0.02); // two percent
axis.setCategoryMargin(0.10); // ten percent
axis.setUpperMargin(0.02); // two percent
```

All of the margins are specified as a percentage of the length of the category axis, to allow for the fact that JFreeChart can draw charts at varying sizes. Note that the percentage for the category margin specifies the total margin for all the categories—if  $N$  is the number of categories, the margin is allocated over  $N - 1$  gaps between the categories.

The spacing between bars *within a category* is not controlled by the axis—instead, it is dealt with by the `BarRenderer`.

```
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setItemMargin(0.15); // fifteen percent
```

As with the category margin, the item margin is the total margin for all the “intra-category” gaps in the chart. If there are  $M$  series in the chart, and  $N$  categories, then there will be  $N \times (M - 1)$  gaps.

A final point to note—the bar widths are dynamically calculated to fill the remaining space after the various margins have been allocated. It is not possible to specify fixed bar widths in JFreeChart.

# Chapter 6

## Line Charts

### 6.1 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.

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

#### 6.2.1 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 6.1.

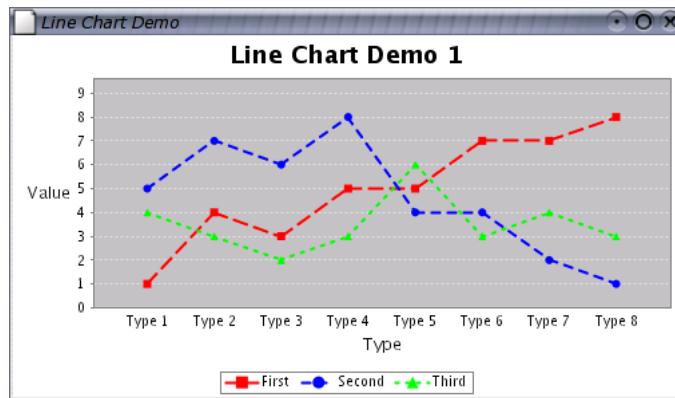


Figure 6.1: A sample line chart

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

### 6.2.2 The Dataset

The first step in generating the chart is, as always, to create a dataset. In the example, the `DefaultCategoryDataset` class is used:

```
/**
 * Creates a sample dataset.
 *
 * @return The dataset.
 */
private CategoryDataset createDataset() {

    // row keys...
    String series1 = "First";
    String series2 = "Second";
    String series3 = "Third";

    // column keys...
    String type1 = "Type 1";
    String type2 = "Type 2";
    String type3 = "Type 3";
    String type4 = "Type 4";
    String type5 = "Type 5";
    String type6 = "Type 6";
    String type7 = "Type 7";
    String type8 = "Type 8";

    // create the dataset...
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();

    dataset.addValue(1.0, series1, type1);
    dataset.addValue(4.0, series1, type2);
    dataset.addValue(3.0, series1, type3);
    dataset.addValue(5.0, series1, type4);
    dataset.addValue(5.0, series1, type5);
    dataset.addValue(7.0, series1, type6);
    dataset.addValue(7.0, series1, type7);
    dataset.addValue(8.0, series1, type8);

    dataset.addValue(5.0, series2, type1);
    dataset.addValue(7.0, series2, type2);
    dataset.addValue(6.0, series2, type3);
    dataset.addValue(8.0, series2, type4);
    dataset.addValue(4.0, series2, type5);
    dataset.addValue(4.0, series2, type6);
    dataset.addValue(2.0, series2, type7);
    dataset.addValue(1.0, series2, type8);

    dataset.addValue(4.0, series3, type1);
    dataset.addValue(3.0, series3, type2);
    dataset.addValue(2.0, series3, type3);
    dataset.addValue(3.0, series3, type4);
    dataset.addValue(6.0, series3, type5);
    dataset.addValue(3.0, series3, type6);
    dataset.addValue(4.0, series3, type7);
    dataset.addValue(3.0, series3, type8);

    return dataset;
}
```

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

### 6.2.3 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...
final JFreeChart chart = ChartFactory.createLineChart(
    "Line Chart Demo 1", // chart title
    "Type", // domain axis label
    "Value", // range axis label
    dataset, // data
    PlotOrientation.VERTICAL, // orientation
    true, // include legend
    true, // tooltips
    false // urls
);
```

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

#### 6.2.4 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, several attributes are modified:

- the chart background color;
- the series stroke;
- the “auto tick units” on the range axis (so that the tick labels always display integer values);

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();
```

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. To draw shapes (as well as lines), customise the line stroke used for each series, and display labels for each data item:

```
LineAndShapeRenderer renderer = (LineAndShapeRenderer) plot.getRenderer();
renderer.setDrawShapes(true);

renderer.setSeriesStroke(
    0, new BasicStroke(
        2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
        1.0f, new float[] {10.0f, 6.0f}, 0.0f
    )
);
renderer.setSeriesStroke(
    1, new BasicStroke(
        2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
        1.0f, new float[] {6.0f, 6.0f}, 0.0f
    )
);
```

```

    renderer.setSeriesStroke(
        2, new BasicStroke(
            2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
            1.0f, new float[] {2.0f, 6.0f}, 0.0f
        )
    );
}

```

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());
rangeAxis.setAutoRangeIncludesZero(true);

```

Refer to the source code, Javadoc API documentation or elsewhere in this document for details of the other customisations that you can make to a line plot.

### 6.2.5 The Complete Program

The code for the demonstration application is presented in full, complete with the import statements. You should find this code included in the JFreeChart Premium Demo distribution.

```

/*
 * LineChartDemo1.java
 *
 * (C) Copyright 2002-2004, by Object Refinery Limited.
 *
 */

package demo;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.StandardLegend;
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.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
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() {

    // row keys...
    String series1 = "First";
    String series2 = "Second";
    String series3 = "Third";

    // column keys...
    String type1 = "Type 1";
    String type2 = "Type 2";
    String type3 = "Type 3";
    String type4 = "Type 4";
    String type5 = "Type 5";
    String type6 = "Type 6";
    String type7 = "Type 7";
    String type8 = "Type 8";

    // create the dataset...
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();

    dataset.addValue(1.0, series1, type1);
    dataset.addValue(4.0, series1, type2);
    dataset.addValue(3.0, series1, type3);
    dataset.addValue(5.0, series1, type4);
    dataset.addValue(5.0, series1, type5);
    dataset.addValue(7.0, series1, type6);
    dataset.addValue(7.0, series1, type7);
    dataset.addValue(8.0, series1, type8);

    dataset.addValue(5.0, series2, type1);
    dataset.addValue(7.0, series2, type2);
    dataset.addValue(6.0, series2, type3);
    dataset.addValue(8.0, series2, type4);
    dataset.addValue(4.0, series2, type5);
    dataset.addValue(4.0, series2, type6);
    dataset.addValue(2.0, series2, type7);
    dataset.addValue(1.0, series2, type8);

    dataset.addValue(4.0, series3, type1);
    dataset.addValue(3.0, series3, type2);
    dataset.addValue(2.0, series3, type3);
    dataset.addValue(3.0, series3, type4);
    dataset.addValue(6.0, series3, type5);
    dataset.addValue(3.0, series3, type6);
    dataset.addValue(4.0, series3, type7);
    dataset.addValue(3.0, series3, type8);

    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(
        "Line Chart Demo 1",           // chart title
        "Type",                      // domain axis label
        "Value",                     // range axis label

```

```

        dataset,                                // data
        PlotOrientation.VERTICAL,    // orientation
        true,                                     // include legend
        true,                                     // tooltips
        false                                     // urls
    );

    // NOW DO SOME OPTIONAL CUSTOMISATION OF THE CHART...
    StandardLegend legend = (StandardLegend) chart.getLegend();
    legend.setDisplaySeriesShapes(true);
    legend.setShapeScaleX(1.5);
    legend.setShapeScaleY(1.5);
    legend.setDisplaySeriesLines(true);

    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());
    rangeAxis.setAutoRangeIncludesZero(true);

    // customise the renderer...
    LineAndShapeRenderer renderer = (LineAndShapeRenderer) plot.getRenderer();
    renderer.setDrawShapes(true);

    renderer.setSeriesStroke(
        0, new BasicStroke(
            2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
            1.0f, new float[] {10.0f, 6.0f}, 0.0f
        )
    );
    renderer.setSeriesStroke(
        1, new BasicStroke(
            2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
            1.0f, new float[] {6.0f, 6.0f}, 0.0f
        )
    );
    renderer.setSeriesStroke(
        2, new BasicStroke(
            2.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
            1.0f, new float[] {2.0f, 6.0f}, 0.0f
        )
    );
    // 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);
}

/**
 * Returns a description of the demo.
 *
 * @return A description.
 */
public static String getDemoDescription() {
    return "A line 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);  
  
}  
}
```

### 6.2.6 A Line Chart Based On An XYDataset

#### 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 6.2.

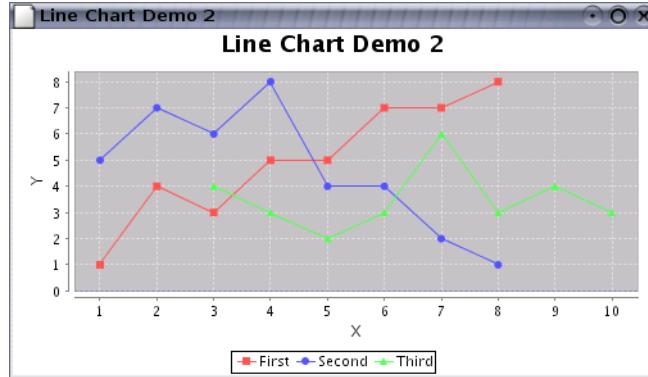


Figure 6.2: A sample line chart using an `XYPlot`

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

#### 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.

### 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).

### 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 grid-lines, 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`.

### 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 7

## Time Series Charts

### 7.1 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.

### 7.2 Time Series Charts

#### 7.2.1 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 7.1.



Figure 7.1: A time series chart

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

### 7.2.2 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.

### 7.2.3 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).

### 7.2.4 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.

### 7.2.5 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.setShapesVisible(true);
    renderer.setShapesFilled(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.

### 7.2.6 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
        );
    }
}
```

```

        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.setDefaultShapesVisible(true);
        renderer.setDefaultShapesFilled(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 8

# Customising Charts

## 8.1 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.

## 8.2 Chart Attributes

### 8.2.1 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.

### 8.2.2 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.

### 8.2.3 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 `Title`. 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`.

### 8.2.4 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.

### 8.2.5 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.

---

<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.

### 8.2.6 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).

### 8.2.7 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.

## 8.3 Plot Attributes

### 8.3.1 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.

### 8.3.2 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.

### 8.3.3 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`.

### 8.3.4 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).

## 8.4 Axis Attributes

### 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.

### 8.4.1 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);
```

#### 8.4.2 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.

#### 8.4.3 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).

#### 8.4.4 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.

#### 8.4.5 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.

#### 8.4.6 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.

#### 8.4.7 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();
TickUnits 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.

#### 8.4.8 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 9

## Dynamic Charts

### 9.1 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.

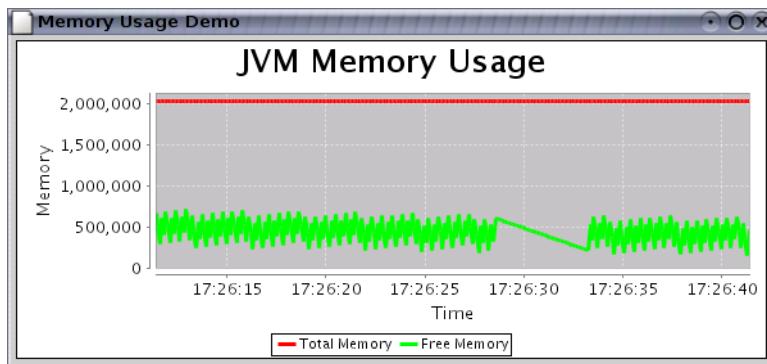


Figure 9.1: A dynamic chart demo

### 9.2 Background

#### 9.2.1 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.

### 9.2.2 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.

## 9.3 The Demo Application

### 9.3.1 Overview

The `MemoryUsageDemo.java` demonstration is included in the “premium demos” download available to purchasers of this document. 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.

### 9.3.2 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.setHistoryCount(30000);
this.free = new TimeSeries("Free", Millisecond.class);
this.free.setHistoryCount(30000);
TimeSeriesCollection dataset = new TimeSeriesCollection();
dataset.addSeries(this.total);
dataset.addSeries(this.free);
```

The *historyCount* 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.

### 9.3.3 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.

### 9.3.4 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.*

### 9.3.5 Source Code

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

```
/*
 * -----
 * MemoryUsageDemo.java
 * -----
 * (C) Copyright 2002-2005, 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>
 * 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 historyCount the history count (in milliseconds).
     */
    public MemoryUsageDemo(int historyCount) {
        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.setHistoryCount(historyCount);
        this.free = new TimeSeries("Free Memory", Millisecond.class);
        this.free.setHistoryCount(historyCount);
        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 10

## Tooltips

### 10.1 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;

### 10.2 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.

#### 10.2.1 Pie Charts

The `PiePlot` class generates tool tips using the `PieToolTipGenerator` interface. A standard implementation (`StandardPieItemLabelGenerator`) 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.
```

### 10.2.2 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.
```

### 10.2.3 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.
```

## 10.3 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.

## 10.4 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.
```

## 10.5 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.

## 10.6 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 11

## Item Labels

### 11.1 Introduction

#### 11.1.1 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 11.1.

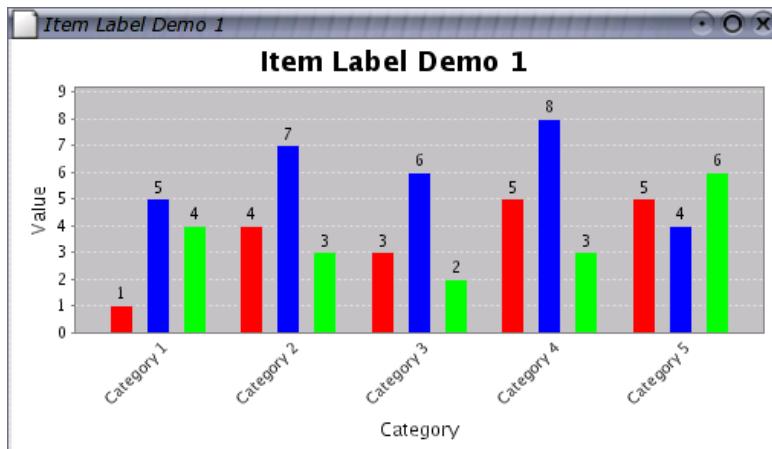


Figure 11.1: A bar chart with item labels

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.

### 11.1.2 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.

## 11.2 Displaying Item Labels

### 11.2.1 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 `CategoryLabelGenerator` or `XYLabelGenerator` 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.

### 11.2.2 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();
CategoryLabelGenerator generator = new StandardCategoryLabelGenerator(
    "{2}", new DecimalFormat("0.00")
);
renderer.setLabelGenerator(generator);
```

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

```
XYItemRenderer renderer = plot.getRenderer();
XYLabelGenerator generator = new StandardXYLabelGenerator(
    "{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 11.5.2.

### 11.2.3 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 11.2.4).

### 11.2.4 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 11.2.

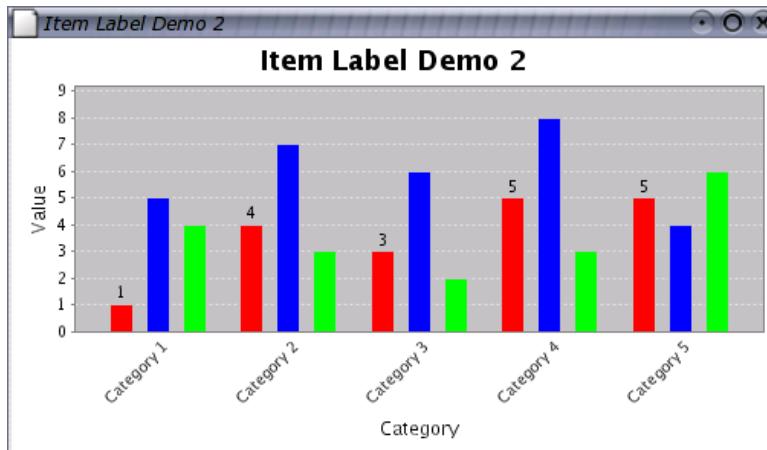


Figure 11.2: Item labels for selected series only

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.

### 11.2.5 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).

## 11.3 Item Label Appearance

### 11.3.1 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.*

### 11.3.2 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.

### 11.3.3 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.

## 11.4 Item Label Positioning

### 11.4.1 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.

### 11.4.2 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.

### 11.4.3 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.

#### 11.4.4 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.

#### 11.4.5 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.

### 11.5 Customising the Item Label Text

#### 11.5.1 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.

#### 11.5.2 Implementing a Custom Label Generator

To develop a custom label generator, you simply need to write a class that implements the method defined in the `CategoryLabelGenerator` 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.

### 11.6 Example 1 - Values Above a Threshold

#### 11.6.1 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 11.3).

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

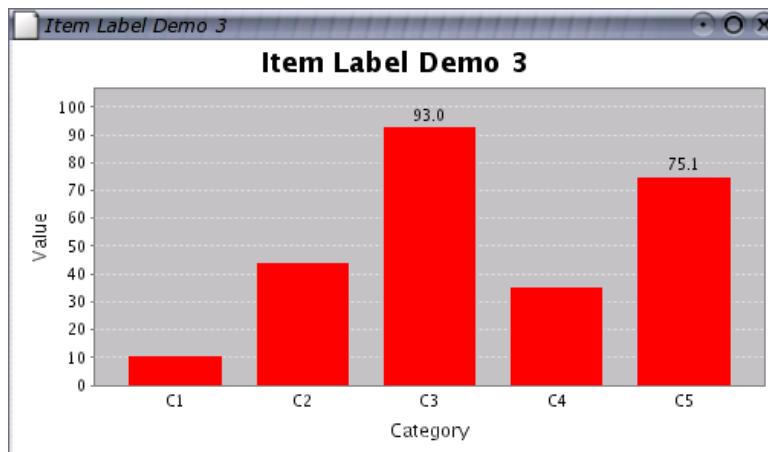


Figure 11.3: Item labels above a threshold

- write a class that implements the `CategoryLabelGenerator` 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.

### 11.6.2 Source Code

The complete source code is presented below.

```
/*
 * -----
 * ItemLabelDemo1.java
 * -----
 * (C) Copyright 2004, 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.labels.CategoryLabelGenerator;
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.RefineryUtilities;

/**
 * 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 implements CategoryLabelGenerator {

        /** 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) {
            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;
            final Number value = dataset.getValue(series, category);
            if (value != null) {
                final 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.setLabelGenerator(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);
}

/**
 * Returns a description of the demo.
 *
 * @return A description.
 */
public static String getDemoDescription() {
    return "A bar chart with item labels displayed only for values greater than a threshold.";
}

/**
 * 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);

    }

}

```

## 11.7 Example 2 - Displaying Percentages

### 11.7.1 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 11.4.

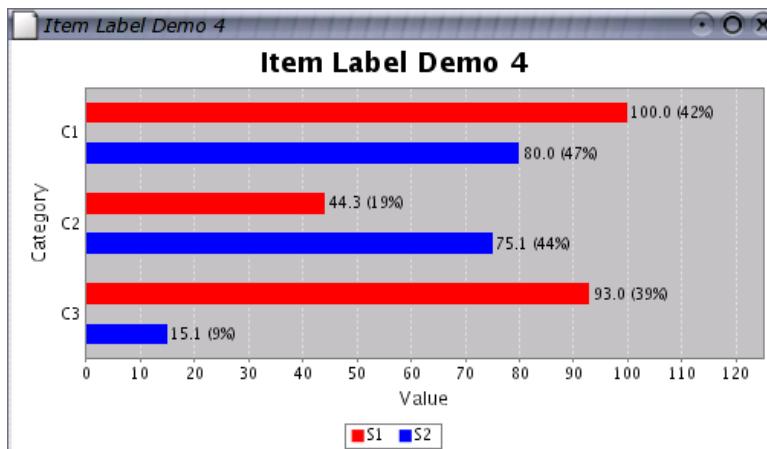


Figure 11.4: Percentage item labels

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.

### 11.7.2 Source Code

The complete source code follows.

```

/*
 * -----
 * ItemLabelDemo2.java
 * -----
 * (C) Copyright 2004, by Object Refinery Limited.

```

```

*
*/

package demo;

import java.awt.Color;
import java.awt.Dimension;
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.AxisLocation;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.labels.CategoryLabelGenerator;
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.RefineryUtilities;

/**
 * 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 implements CategoryLabelGenerator {

        /**
         * 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(final 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) {
            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.
 */
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.setLabelGenerator(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);
}

/**
 * Returns a description of the demo.
 *
 * @return A description.
 */
public static String getDemoDescription() {
    return "A bar chart with customised item labels.";
}

/**
 * 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 12

## Multiple Axes and Datasets

### 12.1 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 12.1 for an example.

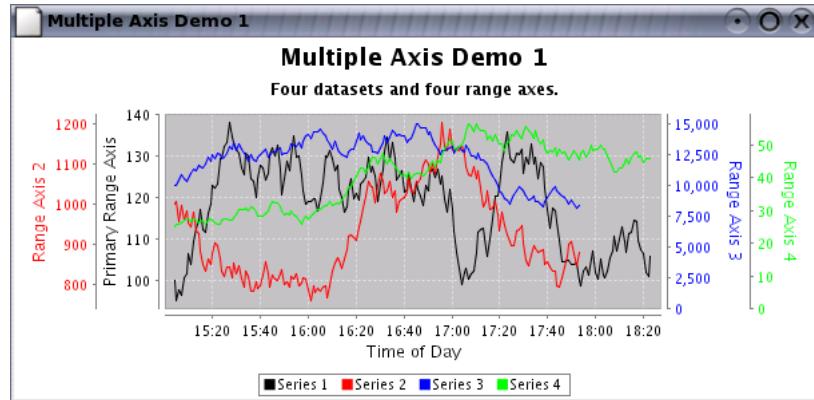


Figure 12.1: A chart with multiple axes

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.

## 12.2 An Example

### 12.2.1 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.

### 12.2.2 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
);
```

### 12.2.3 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.

### 12.2.4 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.

### 12.2.5 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.

## 12.3 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 13

## Combined Charts

### 13.1 Introduction

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

- `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 Premium Demo distribution.

### 13.2 Combined Domain Category Plot

#### 13.2.1 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 13.1.

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

#### 13.2.2 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",
```

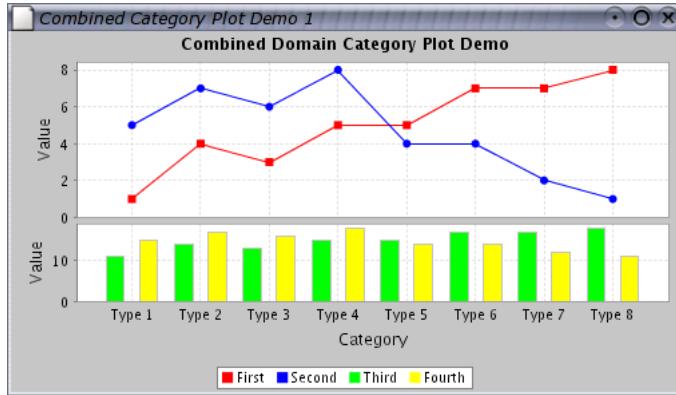


Figure 13.1: A combined domain category plot

```

        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);

```

## 13.3 Combined Range Category Plot

### 13.3.1 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 13.2.

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

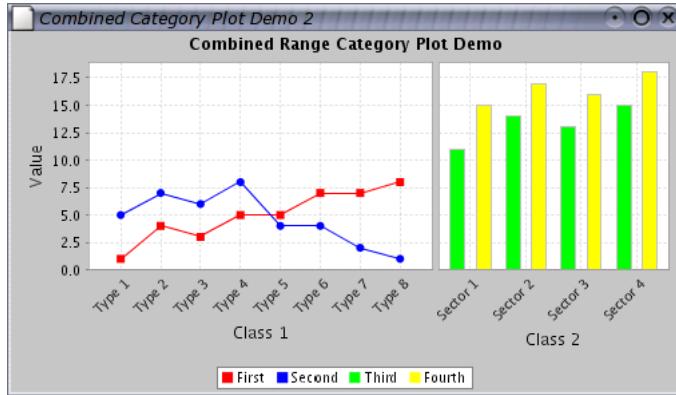


Figure 13.2: A combined range category plot.

### 13.3.2 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
);
```

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);
```

## 13.4 Combined Domain XY Plot

### 13.4.1 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 13.3.

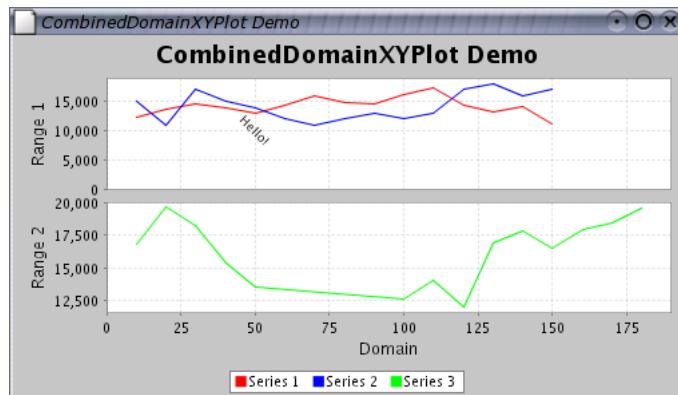


Figure 13.3: A combined domain XY plot

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

### 13.4.2 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);

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);

```

## 13.5 Combined Range XY Plot

### 13.5.1 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 13.4.

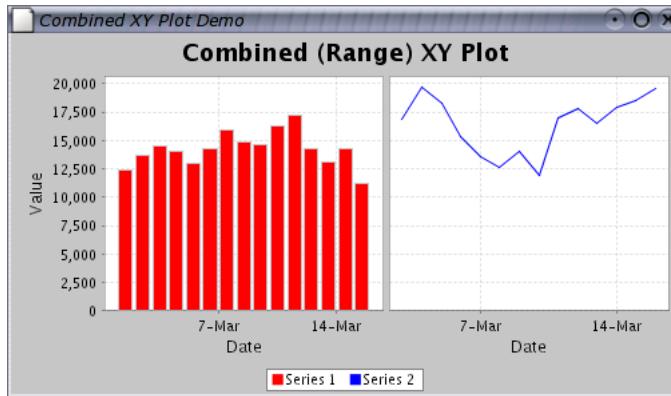


Figure 13.4: A combined range XY plot

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

### 13.5.2 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 14

## Datasets and JDBC

### 14.1 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](#)
- [JBCCCategoryDataset](#)
- [JDBCXYDataset](#)

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

### 14.2 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.

### 14.3 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 `category-data1`):

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.

## 14.4 PostgreSQL

### 14.4.1 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.

### 14.4.2 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.

#### 14.4.3 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);
```

#### 14.4.4 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);
```

#### 14.4.5 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;

```

## 14.5 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.

## 14.6 The Demo Applications

### 14.6.1 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() {
    JDBCPieDataset 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 JDBCPieDataset(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 [JDBCPieDataset](#) class documentation for details).

#### 14.6.2 JDBCCategoryChartDemo

The `JDBCCategoryChartDemo` application generates a bar chart using the data in the `categorydata1` table. The code is almost identical to the `JDBCPieChartDemo`. Once again, you can use any SQL query as long as it returns columns in the required format (refer to the [JDBCCategoryDataset](#) class documentation for details).

#### 14.6.3 JDBCXYChartDemo

The `JDBCXYChartDemo` application generates a time series chart using the data in the `xydata1` table. The code is almost identical to the `JDBCPieChartDemo`. Once again, you can use any SQL query as long as it returns columns in the required format (refer to the [JDBCXYDataset](#) class documentation for details).

# Chapter 15

## Exporting Charts to Acrobat PDF

### 15.1 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.

### 15.2 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.

### 15.3 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 1.01.

## 15.4 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.

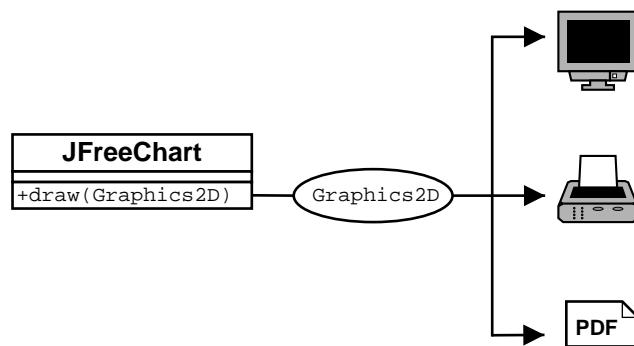


Figure 15.1: The JFreeChart `draw(...)` method

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.

## 15.5 Getting Started

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

File:	Description:
<code>jfreechart-1.0.0-pre2.jar</code>	The JFreeChart class library.
<code>jcommon-1.0.0-pre2.jar</code>	The JCommon class library (used by JFreeChart).
<code>iText-1.01.jar</code>	The iText class library.

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

## 15.6 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):

```
package com.jrefinery.chart.demo;

import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.text.SimpleDateFormat;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.StandardLegend;
import org.jfree.chart.axis.DateAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.StandardXYItemRenderer;
import org.jfree.chart.renderer.XYItemRenderer;
import org.jfree.data.XYDataset;
import org.jfree.data.time.Month;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;

import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
```

```

import com.lowagie.text.Rectangle;
import com.lowagie.text.pdf.DefaultFontMapper;
import com.lowagie.text.pdf.FontMapper;
import com.lowagie.text.pdf.PdfContentByte;
import com.lowagie.text.pdf.PdfTemplate;
import com.lowagie.text.pdf.PdfWriter;

< /**
 * A simple demonstration showing how to write a chart to PDF format using
 * JFreeChart and iText.
 * <P>
 * You can download iText from http://www.lowagie.com/iText.
 */
public class ChartToPDFDemo {

    /**
     * Saves a chart to a PDF file.
     *
     * @param file the file.
     * @param chart the chart.
     * @param width the chart width.
     * @param height the chart height.
     */
    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();
    }

    /**
     * Writes a chart to an output stream in PDF format.
     *
     * @param out the output stream.
     * @param chart the chart.
     * @param width the chart width.
     * @param height the chart height.
     */
    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();
    }

    /**
     * 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,
            true,
            false
        );

        // some additional chart customisation here...
        StandardLegend sl = (StandardLegend) chart.getLegend();
        sl.setDisplaySeriesShapes(true);
        XYPlot plot = chart.getXYPlot();
        XYItemRenderer renderer = plot.getRenderer();
        if (renderer instanceof StandardXYItemRenderer) {
            StandardXYItemRenderer rr = (StandardXYItemRenderer) renderer;
            rr.setPlotShapes(true);
            rr.setShapesFilled(true);
        }
    }
}

```

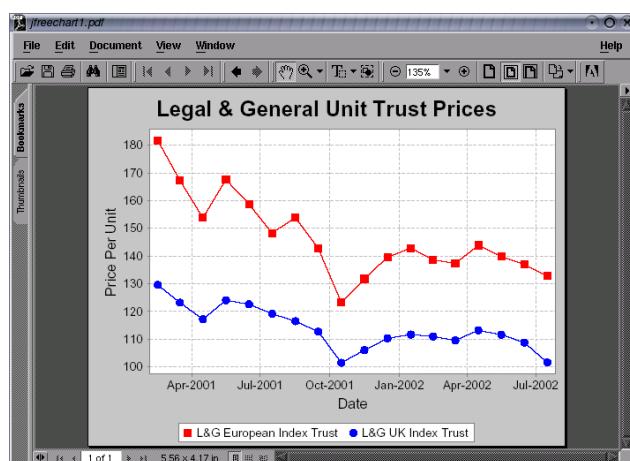
```
        }
        DateAxis axis = (DateAxis) plot.getDomainAxis();
        axis.setDateFormatOverride(new SimpleDateFormat("MMM-yyyy"));

        // write the chart to a PDF file...
        File fileName = new File(System.getProperty("user.home")
                                + "/jfreechart1.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 15.5 on your classpath.

## 15.7 Viewing the PDF File

After compiling and running the sample application, you can view the resulting PDF file using Acrobat Reader:



Acrobat Reader provides a zooming facility to allow you to get a close up view of your charts.

## 15.8 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.

### 15.8.1 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!).

### 15.8.2 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.

### 15.8.3 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:

`/usr/lib/SunJava2/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("/usr/lib/SunJava2/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 15.2. 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.

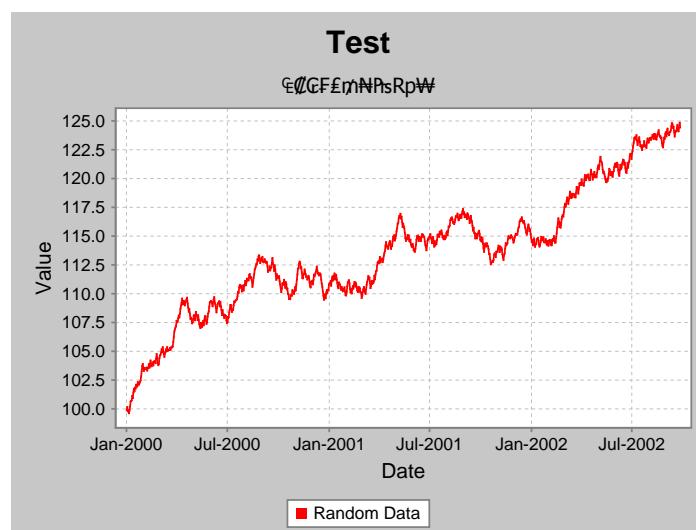


Figure 15.2: A Unicode subtitle

# Chapter 16

## Exporting Charts to SVG Format

### 16.1 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).

### 16.2 Background

#### 16.2.1 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).

#### 16.2.2 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 1.5.

### 16.3 A Sample Application

#### 16.3.1 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.

### 16.3.2 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:
<code>jcommon-1.0.0-pre2.jar</code>	Common classes from The Object Refinery.
<code>jfreechart-1.0.0-pre2.jar</code>	The JFreeChart class library.
<code>batik-awt-util.jar</code>	Batik runtime files.
<code>batik-dom.jar</code>	Batik runtime files.
<code>batik-ext.jar</code>	Batik runtime files.
<code>batik-svggen.jar</code>	Batik runtime files.
<code>batik-util.jar</code>	Batik runtime files.
<code>batik-xml.jar</code>	Batik runtime files.

### 16.3.3 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:

```
package com.jrefinery.chart.demo;

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.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);

    }

}

```

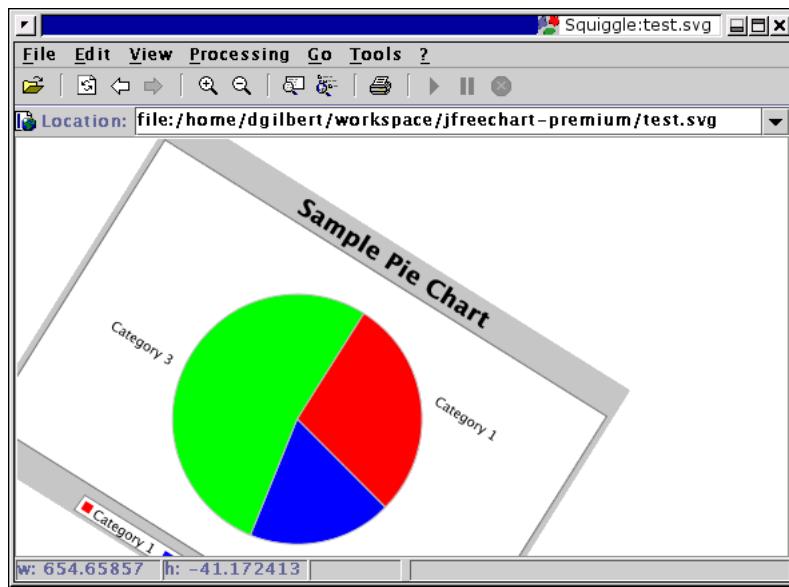
Running this program creates a file `test.svg` in SVG format.

#### 16.3.4 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 17

## Applets

### 17.1 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.

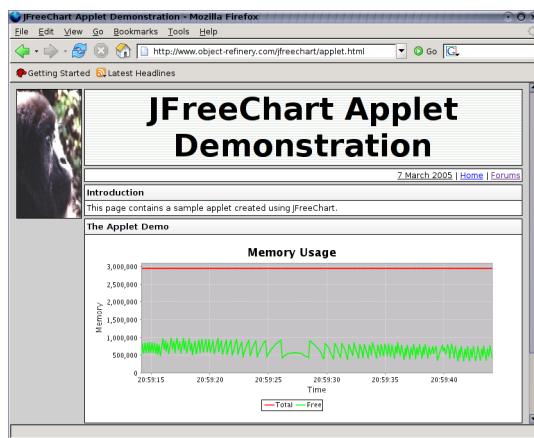


Figure 17.1: An applet using JFreeChart

Figure 17.1 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.

### 17.2 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.

### 17.2.1 Browser Support

The *vast majority* of web browsers provide support for the latest version of Java (JDK 1.4) and will therefore have no problems running applets that use JFreeChart (recall that JFreeChart will run on any version of the JDK from 1.2.2 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).

### 17.2.2 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.

### 17.2.3 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.

---

<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 1.4. 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.

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).

### 17.3 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.

#### 17.3.1 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-1.0.0-pre2-applet-demo.jar,
jfreechart-1.0.0-pre2.jar,jcommon-1.0.0-pre2.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).

#### 17.3.2 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;
```

---

<sup>2</sup>If the applet does not work for you, please check that your web browser is configured correctly and supports JDK 1.2.2 or later.

```
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.setHistoryCount(30000);
        this.free = new TimeSeries("Free", Millisecond.class);
        this.free.setHistoryCount(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();
    }
}

class DataGenerator implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // Add data to the series here...
    }
}
```

```
}

 /**
 * 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 18

## Servlets

### 18.1 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-1.0.0-pre2-demo.zip`.<sup>1</sup>

### 18.2 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 18.1. Typically, you will not present raw output in this way, so this servlet is not especially useful on its own, but the example is:

- 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:

---

<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.

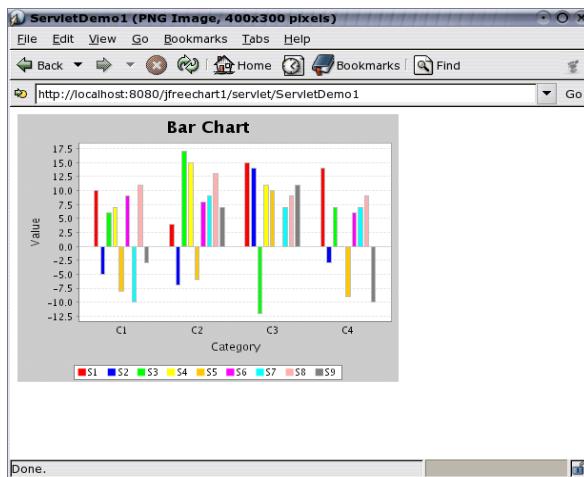


Figure 18.1: ServletDemo1 in a browser

```

/*
 * -----
 * 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.

### 18.3 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-api.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://jakarta.apache.org/tomcat
```

You will also require the JFreeChart and JCommon jar files to compile the above servlet. Change your working directory to `jfreechart-1.0.0-pre2-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-1.0.0-pre2.jar:lib/jcommon-1.0.0-pre2.jar:lib/servlet-api.jar
source/demo/ServletDemo1.java
```

This should create a `ServletDemo1.class` file. The next section describes how to deploy this servlet using Tomcat.

### 18.4 Deploying the Servlet

Servlets are deployed in the `webapps` directory provided by your servlet engine. In my case, I am using Tomcat 5.0.28 on SUSE Linux 9.1, and the directory is:<sup>2</sup>

```
/opt/jakarta-tomcat-5.0.28/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-1.0.0-pre2.jar
.../jfreechart1/WEB-INF/lib/jcommon-1.0.0-pre2.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>
```

---

<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.

```

<servlet-name>
    ServletDemo1
</servlet-name>
<servlet-class>
    com.jrefinery.chart.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 18.1.

## 18.5 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-1.0.0-pre2-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 18.2.

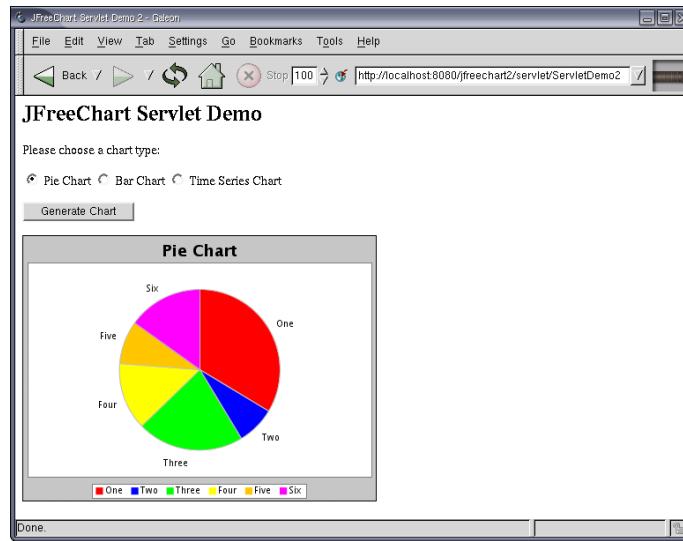


Figure 18.2: `ServletDemo2` in a browser

Here is the code for `ServletDemo2`:

```

/*
 * -----
 * 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.println("</HTML>");
        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;
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;

```

---

<sup>3</sup>The `Writer` is wrapped in a `PrintWriter` in order to use the more convenient methods available in the latter class.

```


/**
 * 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-0.9.21.jar:lib/jcommon-0.9.6.jar:lib/servlet-api.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.

## 18.6 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>

```

---

<sup>4</sup>You'll find this file in the `servlets` directory of the demo distribution, along with the other servlet support files.

```

<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
    <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.

## 18.7 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-1.0.0-pre2.jar
webapps/jfreechart2/WEB-INF/lib/jfreechart-1.0.0-pre2.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 5.0.29 running on SuSE Linux 9.1).

# Chapter 19

## Miscellaneous

### 19.1 Introduction

This section contains miscellaneous information about JFreeChart.

### 19.2 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>

### 19.3 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.

### 19.4 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 20

## Packages

### 20.1 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.entity</code>	Classes representing chart entities.
<code>org.jfree.chart.event</code>	The event classes.
<code>org.jfree.chart.htmlmap</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.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.ui</code>	User interface 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.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 Javadoc HTML files.

# Chapter 21

## Package: org.jfree.chart

### 21.1 Overview

This package contains the major classes and interfaces in the *JFreeChart Class Library*, including the all important `JFreeChart` class.

### 21.2 ChartColor

#### 21.2.1 Overview

This class defines some standard colors.

#### 21.2.2 Notes

The `DefaultDrawingSupplier` class uses the `createDefaultPaintArray()` method to generate the default paint sequence for charts.

### 21.3 ChartFactory

#### 21.3.1 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.*

#### 21.3.2 Pie Charts

To create a regular pie chart:

```
public static JFreeChart createPieChart(String title,  
PieDataset dataset, boolean legend, boolean tooltips, boolean urls);  
Creates a pie chart for the specified PieDataset (null permitted). The  
chart is constructed using a PiePlot.
```

To create a pie chart with a “3D effect”:

```
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.
```

### 21.3.3 Methods

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 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.
```

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.
```

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 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.
```

To create a high-low-open-close chart:

```
public static JFreeChart createHighLowChart(String title,
String timeAxisLabel, String valueAxisLabel, HighLowDataset dataset,
Timeline timeline, boolean legend)
Creates a high-low-open-close chart for the given HighLowDataset.
```

To create a candlestick chart:

```
public static JFreeChart createCandlestickChart(String title,
String timeAxisLabel, String valueAxisLabel, HighLowDataset data, boolean
legend)
Creates a candlestick chart for the given HighLowDataset.
```

To create an area chart using data from a `XYDataset`:

```
public static JFreeChart createXYAreaChart(String title, String xAxisLabel,
String yAxisLabel, 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
xAxisLabel, String yAxisLabel, 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.
```

## 21.4 ChartFrame

### 21.4.1 Overview

A frame containing chart within a `ChartPanel`.

### 21.4.2 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.
```

### Methods

To access the chart's panel:

```
public ChartPanel getChartPanel();
Returns the panel that contains the chart.
```

## 21.5 ChartMouseEvent

### 21.5.1 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.

### 21.5.2 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.

### 21.5.3 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.

public ChartEntity getEntity();
    Returns the chart entity underneath the mouse pointer (this may be null).
```

### 21.5.4 Notes

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 21.7.6).

## 21.6 ChartMouseListener

### 21.6.1 Overview

An interface that defines the callback method for a *chart mouse listener*. Any class that implements this interface can be registered with a `ChartPanel` and receive notification of mouse events.

### 21.6.2 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.
```

## 21.7 ChartPanel

### 21.7.1 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;
- 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.

### 21.7.2 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).

### 21.7.3 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.
```

### 21.7.4 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 a different size then scaled down (up) to fit the actual panel size.

The default minimum threshold is 300 pixels (width) x 200 pixels (height). You can change these defaults using the following methods:

```
public int getMinimumDrawWidth();
>Returns the lower threshold for the chart drawing width.

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 getMinimumDrawHeight();
>Returns the lower threshold for the chart drawing height.

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.
```

Similarly, the default maximum threshold is 800 pixels (width) by 600 pixels (height). You can change these defaults using the following methods:

```
public int getMaximumDrawWidth();
>Returns the upper threshold for the chart drawing width.

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.

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.
```

### 21.7.5 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.
```

### 21.7.6 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.
```

### 21.7.7 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.
```

### 21.7.8 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.

The chart panel provides a “mouse zooming” feature. A demonstration of this is provided in the `MouseZoomDemo` application.

#### See Also

[JFreeChart](#).

## 21.8 ChartRenderingInfo

### 21.8.1 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 are 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.

### 21.8.2 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.
```

### 21.8.3 Notes

The `ChartPanel` class automatically collects entity information using this class, because it needs it to generate tool tips.

## 21.9 ChartUtilities

### 21.9.1 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`

### 21.9.2 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 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>

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.
```

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.
```

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.
```

### 21.9.3 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.*

To write a chart to a file in JPEG format:

```
public static void saveChartAsJPEG(File file, JFreeChart chart, int width,
int height);
Saves a chart to a JPEG format image file.
```

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. If an info object is supplied,
it will be populated with information about the structure of the chart.
```

#### 21.9.4 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.

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,
String hrefPrefix, ChartRenderingInfo info);
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`.

#### 21.9.5 Notes

PNG tends to be a better format for charts than JPEG since the compression is “lossless” for PNG.

### 21.10 ClipPath

#### 21.10.1 Overview

Not yet documented.

### 21.11 DrawableLegendItem

#### 21.11.1 Overview

Used to represent a `LegendItem` plus its physical drawing characteristics (position, label location etc.) as it is being laid out on the chart.

## 21.12 Effect3D

### 21.12.1 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.

#### See Also

[BarRenderer3D](#), [CategoryAxis3D](#), [NumberAxis3D](#).

## 21.13 JFreeChart

### 21.13.1 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. The following table summarises the combinations that are currently available:

Dataset:	Compatible Plot Types:
<code>MeterDataset</code>	<code>CompassPlot</code> , <code>MeterPlot</code> and <code>ThermometerPlot</code> .
<code>PieDataset</code>	<code>PiePlot</code> .
<code>CategoryDataset</code>	<code>CategoryPlot</code> subclasses with various renderers.
<code>XYDataset</code>	<code>XYPlot</code> with various renderers.
<code>IntervalXYDataset</code>	<code>XYPlot</code> with a <code>XYBarRenderer</code> .
<code>HighLowDataset</code>	<code>XYPlot</code> with a <code>HighLowRenderer</code> .
<code>HighLowDataset</code>	<code>XYPlot</code> with a <code>CandlestickRenderer</code> .

### 21.13.2 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.

### 21.13.3 Attributes

The attributes maintained by the [JFreeChart](#) class are listed in Table 21.1.

Attribute:	Description:
<code>borderVisible</code>	A flag that controls whether or not a border is drawn around the outside of the chart.
<code>borderStroke</code>	The <a href="#">Stroke</a> used to draw the chart's border.
<code>borderPaint</code>	The <a href="#">Paint</a> used to paint the chart's border.
<code>title</code>	The chart title (an instance of <a href="#">TextTitle</a> ).
<code>subTitles</code>	A list of subtitles.
<code>legend</code>	The chart legend.
<code>plot</code>	The plot.
<code>antialias</code>	A flag that indicates whether or not the chart should be drawn with anti-aliasing.
<code>backgroundPaint</code>	The background paint for the chart.
<code>backgroundImage</code>	An optional background image for the chart.
<code>backgroundImageAlignment</code>	The alignment of the background image (if there is one).
<code>backgroundImageAlpha</code>	The alpha transparency for the background image.
<code>notify</code>	A flag that controls whether or not change events are passed on to the chart's registered listeners;
<code>renderingHints</code>	The Java2D rendering hints that will be applied when the chart is drawn.

Table 21.1: Attributes for the [JFreeChart](#) class

### 21.13.4 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 set the legend for a chart:

```
public void setLegend(Legend legend);
Sets the legend for a chart.
```

You can control whether or not the chart is drawn with anti-aliasing (switching anti-aliasing *on* can improve the on-screen appearance of charts):

```
public void setAntiAlias(boolean flag);
Sets a flag controlling whether or not anti-aliasing is used when drawing the chart.
```

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.
```

### 21.13.5 Background Image

A chart can have a background image (optional):

```
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 19.4 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.

### 21.13.6 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.

```
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.

### 21.13.7 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.

### 21.13.8 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.
```

### 21.13.9 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, although hopefully less of an issue as browser support for Java 2 improves).

## 21.14 Legend

### 21.14.1 Overview

The base class for a *chart legend* (displays the series names and colors used in a chart). The legend can appear at the top, bottom, left or right of a chart. [StandardLegend](#) is the only subclass available.

### 21.14.2 Usage

If you create charts using the [ChartFactory](#) class, a legend will often be created for you. You can access the legend using the `getLegend()` method in the [JFreeChart](#) class.

To change the position of the legend relative to the chart to one of the positions NORTH, SOUTH, EAST or WEST, use the following code:

```
Legend legend = myChart.getLegend();
legend.setAnchor(Legend.WEST);
```

If you don't want a legend to appear on your chart, you can set it to `null`:

```
myChart.setLegend(null);
```

### 21.14.3 Constructor

This is an abstract class, so the constructor is `protected`.

### 21.14.4 Notes

This class implements a listener mechanism which can be used by subclasses.

#### See Also

[StandardLegend](#).

## 21.15 LegendItem

### 21.15.1 Overview

A class that records the attributes of an item that should appear in a legend. 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 21.2 lists the attributes defined by the class.

### 21.15.2 Constructors

To create a legend item:

```
public LegendItem(String label, String description, Shape shape,
Paint fillPaint);
Creates a legend item with a filled shape (no outline). No line is visible.
```

<b>Attribute:</b>	<b>Description:</b>
<i>label</i>	The label (usually the series name).
<i>description</i>	A description of the item (not currently used).
<i>shapeVisible</i>	A flag that indicates whether or not the shape is visible.
<i>shape</i>	The shape displayed for the legend item.
<i>shapeFilled</i>	A flag that controls whether or not the shape is filled.
<i>fillPaint</i>	The fill paint.
<i>shapeOutlineVisible</i>	A flag that indicates whether or not the shape outline is visible.
<i>outlinePaint</i>	The outline paint.
<i>outlineStroke</i>	The outline stroke.
<i>lineVisible</i>	A flag that indicates whether or not the line is visible.
<i>lineStroke</i>	The line stroke.
<i>linePaint</i>	The line paint.

Table 21.2: Attributes for the *LegendItem* class

```

public LegendItem(String label, String description, Shape shape,
Paint fillPaint, Stroke outlineStroke, Paint outlinePaint);
Creates a legend item with a filled and outlined shape. No line is visible.

public LegendItem(String label, String description, Shape line,
Stroke lineStroke, Paint linePaint);
Creates a legend item with a colored line (and no shape).

public LegendItem(String label, String description,
boolean shapeVisible,
Shape shape, boolean shapeFilled, Paint fillPaint,
boolean shapeOutlineVisible, Paint outlinePaint, Stroke outlineStroke,
boolean lineVisible, Shape line, Stroke lineStroke, Paint linePaint);
Creates a legend item with all attributes specified by the caller.

```

### 21.15.3 Notes

Some points to note:

- instances of this class are immutable;
- this class implements the [Serializable](#) interface.

## 21.16 LegendItemCollection

### 21.16.1 Overview

A collection of legend items.

#### See Also

[Legend](#).

ID:	Description:
LegendRenderingOrder.STANDARD	Items are rendered in order.
LegendRenderingOrder.REVERSE	Items are rendered in reverse order.

Table 21.3: Tokens defined by LegendRenderingOrder

## 21.17 LegendItemSource

### 21.17.1 Overview

An interface for obtaining a collection of legend items. This interface is implemented (or extended) by:

- [CategoryPlot](#);
- [CategoryItemRenderer](#);
- [XYPlot](#);
- [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.

### 21.17.2 Methods

To obtain a collection of legend items:

```
public LegendItemCollection getLegendItems();
    Returns a collection of legend items (possibly empty, but never null).
```

## 21.18 LegendRenderingOrder

### 21.18.1 Overview

A class that defines tokens that control the order of the items in the legend. See table 21.3 for the tokens that are defined.

## 21.19 MeterLegend

### 21.19.1 Overview

To be documented.

## 21.20 PolarChartPanel

### 21.20.1 Overview

An extension of the [ChartPanel](#) class with a pop-up menu that applies to polar charts.

## 21.21 StandardLegend

### 21.21.1 Overview

A chart legend displays the names of the series in a chart.

### 21.21.2 Methods

The legend position is controlled using methods inherited from the [Legend](#) class.

To set the color and thickness of the legend outline, use the following methods:

```
public void setOutlineStroke(Stroke stroke);
Sets the Stroke used to draw the outline for the legend and sends a
LegendChangeEvent to all registered listeners.

public void setOutlinePaint(Paint paint);
Sets the Paint used to draw the outline for the legend and sends a LegendChangeEvent
to all registered listeners.
```

To set the background color for the legend:

```
public void setBackgroundPaint(Paint paint);
Sets the Paint used to fill the background of the legend and sends a
LegendChangeEvent to all registered listeners.
```

To set the title (optional) and title font for the legend:

```
public void setTitle(String title);
Sets the title for the legend and sends a LegendChangeEvent to all registered
listeners. You can set the title to null if you prefer no title for the legend.

public void setTitleFont(Font font);
Sets the title font for the legend and sends a LegendChangeEvent to all
registered listeners.
```

To set the color and font used for the legend item text:

```
public voidsetItemFont(Font font);
Sets the font used to display the text for the legend items.

public voidsetItemPaint(Paint paint);
Sets the paint used to display the text for the legend items.
```

### 21.21.3 Legend Item Shapes

If your chart displays shapes to represent the items in a series, you can get the legend to reflect this using the following method:

```
public void setDisplaySeriesShapes(boolean flag);
Sets the flag that controls whether shapes are displayed for the legend
items.
```

A range of methods are available to change the appearance of the shapes in the legend. The fill color is obtained from the chart's renderer, but the outline paint and stroke is set in the legend:

```
public void setShapeOutlinePaint(Paint paint);
Sets the Paint used to outline shapes in the legend.
```

```
public void setShapeOutlineStroke(Stroke stroke);  
Sets the Stroke used to outline shapes in the legend.
```

You can also scale the size of the shapes displayed in the legend:

```
public void setShapeScaleX(double factor);
```

Sets the x scale factor for the shapes displayed in the legend.

```
public void setShapeScaleY(double factor);
```

Sets the y scale factor for the shapes displayed in the legend.

#### 21.21.4 Notes

Some points to note:

- the legend does not have methods to get or set the items that will be displayed. At the time a chart is drawn, the legend items are obtained via a call to the `getLegendItems()` method in the `Plot` class;
- it is planned that this class should be replaced by a `LegendTitle` class, so that the legend can be treated (for layout purposes) as if it were a chart title.

# Chapter 22

## Package: org.jfree.chart.annotations

### 22.1 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.

### 22.2 CategoryAnnotation

#### 22.2.1 Overview

The interface that must be supported by annotations that are to be added to a `CategoryPlot`.

The `CategoryTextAnnotation` class is the only implementation of this interface that is included in the JFreeChart distribution.

#### 22.2.2 Methods

This interface defines a single method:

```
public void draw(Graphics2D g2, CategoryPlot plot, Rectangle2D dataArea,  
CategoryAxis domainAxis, ValueAxis rangeAxis);  
Draws the annotation.
```

### 22.3 CategoryTextAnnotation

#### 22.3.1 Overview

An annotation that can be used to display an item of text at some location (defined by a *(category, value)* pair) on a `CategoryPlot`.

## 22.4 TextAnnotation

### 22.4.1 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.

### 22.4.2 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 with the specified attributes.

### 22.4.3 Methods

There are methods for accessing the `text`, `font`, `paint`, `anchor` and `rotation` attributes.

### 22.4.4 Notes

`CategoryTextAnnotation` and `XYTextAnnotation` are the two subclasses included in the JFreeChart distribution.

## 22.5 XYAnnotation

### 22.5.1 Overview

The interface that must be supported by annotations that are to be added to an `XYPlot`.

This interface is implemented by:

- `XYDrawableAnnotation`;
- `XYLineAnnotation`;
- `XYPointerAnnotation`;
- `XYTextAnnotation`;

You can, of course, provide your own implementations of the interface.

### 22.5.2 Methods

This class 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.

## 22.6 XYDrawableAnnotation

### 22.6.1 Overview

An annotation 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).

### 22.6.2 Notes

See the `MarkerDemo1.java` source file in the JFreeChart Premium Demo distribution for an example.

## 22.7 XYImageAnnotation

### 22.7.1 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 19.4).

### 22.7.2 Constructor

There is just one constructor:

```
public XYImageAnnotation(double x, double y, Image image);
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 centered about the
specified location.
```

### 22.7.3 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.
```

### 22.7.4 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 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).

### 22.7.5 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.

## 22.8 XYLineAnnotation

### 22.8.1 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);
```

### 22.8.2 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.
```

### 22.8.3 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.

### 22.8.4 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`.

### 22.8.5 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”.

## 22.9 XYPointerAnnotation

### 22.9.1 Overview

An annotation that displays an arrow pointing towards a specific  $(x, y)$  location on an `XYPlot` (see figure 22.1). The arrow can have a label at one end.



Figure 22.1: An `XYPointerAnnotation` example

## 22.9.2 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);
```

## 22.9.3 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.
```

## 22.9.4 Methods

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).
```

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.
```

## 22.10 XYPolygonAnnotation

### 22.10.1 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).

### 22.10.2 Constructors

To create a new annotation:

```
public XYPolygonAnnotation(double[] polygon);
Creates a new annotation that draws a polygon with the supplied coordinates. The array contains (x, y) coordinates of the polygon's vertices, and the polygon will be drawn with a black outline, one unit wide.
```

```
public XYPolygonAnnotation(double[] polygon, Stroke stroke,
Paint outlinePaint)
Creates a new annotation that draws the specified polygon with the given stroke and outline paint. 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.
```

### 22.10.3 Methods

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.
```

### 22.10.4 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.

## 22.11 XYShapeAnnotation

### 22.11.1 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).

### 22.11.2 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.

## 22.12 XYTextAnnotation

### 22.12.1 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.

The annotation inherits font, paint, alignment and rotation settings from the `TextAnnotation` class.

### 22.12.2 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.

### 22.12.3 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.
```

### 22.12.4 Methods

This class defines methods to get and set the `x` and `y` values (defining the location of the annotation against the domain and range axes):

```
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.

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.
```

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);
```

### 22.12.5 Notes

Some points to note:

- this class is cloneable and serializable;
- the `AnnotationDemo1.java` application (included in the premium demo collection) provides an example;
- the `XYPointerAnnotation` subclass can be used to display a label with an arrow pointing to some  $(x, y)$  value.

# Chapter 23

## Package: org.jfree.chart.axis

### 23.1 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 12 for more information.

The axis classes are `Cloneable` and `Serializable`.

### 23.2 Axis

#### 23.2.1 Overview

An abstract base class representing an axis. Some subclasses of `Plot`, including `CategoryPlot` and `XYPlot`, will use axes to display data.

Figure 23.1 illustrates the axis class hierarchy.

#### 23.2.2 Constructors

The constructors for this class are `protected`, you cannot create an instance of this class directly—you must use a subclass.

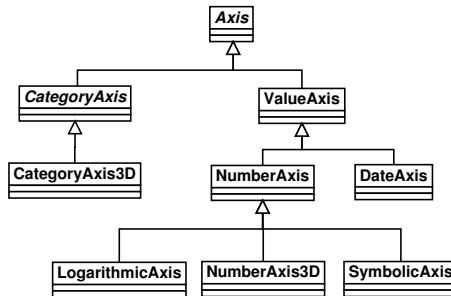


Figure 23.1: Axis classes

### 23.2.3 Attributes

The attributes maintained by the `Axis` class are listed in Table 23.1. 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.

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 color for the axis label.
<code>labelInsets</code>	The space to leave around the outside of 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.
<code>tickMarkOutsideLength</code>	The amount by which the tick marks extend outside the plot area.

Table 23.1: Attributes for the `Axis` class

The default values used to initialise the axis attributes are listed in Table 23.2.

Name:	Value:
DEFAULT_AXIS_LABEL_FONT	new Font("SansSerif", Font.PLAIN, 14);
DEFAULT_AXIS_LABEL_PAINT	Color.black;
DEFAULT_AXIS_LABEL_INSETS	new Insets(2, 2, 2, 2);
DEFAULT_TICK_LABEL_FONT	new Font("SansSerif", Font.PLAIN, 10);
DEFAULT_TICK_LABEL_PAINT	Color.black;
DEFAULT_TICK_LABEL_INSETS	new Insets(2, 1, 2, 1);
DEFAULT_TICK_STROKE	new BasicStroke(1);

Table 23.2: *Axis* class default attribute values

### 23.2.4 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.

### 23.2.5 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).

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.
```

### 23.2.6 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.

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).

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).

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 visibility of the tick labels for an axis:

```
public boolean isTickLabelsVisible();
>Returns the flag that controls whether or not the tick labels are visible.

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.

public void setTickLabelFont(Font font);
>Sets the tick label font and sends an AxisChangeEvent to all registered
>listeners.
```

To control the paint used to draw the tick labels:

```
public Paint getTickLabelPaint();
>Returns the tick label paint.

public void setTickLabelPaint(Paint paint);
>Sets the tick label paint and sends an AxisChangeEvent to all registered
>listeners.
```

### 23.2.7 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.
```

### 23.2.8 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](#).

## 23.3 AxisCollection

### 23.3.1 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.

### 23.3.2 Notes

Axis collections are maintained only temporarily during the process of drawing a chart.

## 23.4 AxisLocation

### 23.4.1 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.

### 23.4.2 Notes

The axis location is set using methods in the `CategoryPlot` and `XYPlot` classes.

## 23.5 AxisSpace

### 23.5.1 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.

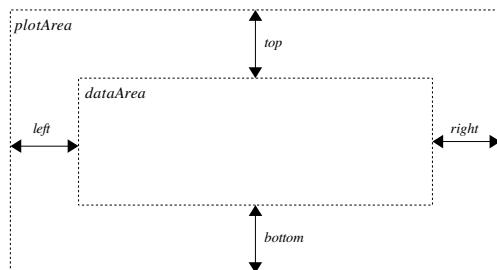


Figure 23.2: `AxisSpace` Attributes

Axes are always drawn around the edges of the *data area* but should never extend outside the *plot area*.

### 23.5.2 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.
```

## 23.6 AxisState

### 23.6.1 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.

### 23.6.2 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.

## 23.7 CategoryAnchor

### 23.7.1 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.

### 23.7.2 Usage

This class is used to control the position of the domain axis gridlines drawn in a [CategoryPlot](#) (see the `setDomainGridlinePosition()` method).

## 23.8 CategoryAxis

### 23.8.1 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 23.3: 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 [23.8.5](#).

### 23.8.2 Constructor

There is a single 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.

### 23.8.3 Attributes

The attributes maintained by the `CategoryAxis` class are listed in Table 23.3. These attributes are in addition to those inherited from the [Axis](#) class (see section [23.2.3](#) for details).

The following default values are used:

Default:	Value:
<code>DEFAULT_AXIS_MARGIN</code>	0.05 (5 percent).
<code>DEFAULT_CATEGORY_MARGIN</code>	0.20 (20 percent).

### 23.8.4 Setting Axis Margins

To set the lower margin for the axis:

<b>Attribute:</b>	<b>Description:</b>
<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>	The offset between the axis line and the category labels.
<i>categoryLabelPositions</i>	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 23.3: Attributes for the `CategoryAxis` class*

```
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 set the upper margin for the axis:

```
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 set the margin 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.

### 23.8.5 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.

```
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.
```

```
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 set the position of the category labels:

```
public void setCategoryLabelPositions(CategoryLabelPositions positions);
Sets the attribute that controls the position, alignment and rotation of the category labels along the axis.
```

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 23.4). Within this rectangle, an *anchor point* is specified using the `RectangleAnchor` class.



Figure 23.4: 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.

### 23.8.6 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.

### 23.8.7 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.
```

### 23.8.8 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.
```

### 23.8.9 Cloning and Serialization

This class is `Cloneable` and `Serializable`.

### 23.8.10 Notes

Some points to note:

- tick marks are not supported by this axis (yet).

## 23.9 CategoryAxis3D

### 23.9.1 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.

## 23.10 CategoryLabelPosition

### 23.10.1 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 23.4);
- 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).

### 23.10.2 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.

### 23.10.3 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.

## 23.11 CategoryLabelPositions

### 23.11.1 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 23.4: Static instances of the [CategoryLabelPositions](#) class

### 23.11.2 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.

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 23.5: Tokens defined by `CategoryLabelWidthType`

## 23.12 CategoryLabelWidthType

### 23.12.1 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 23.5 for the tokens that are defined.

### 23.12.2 Usage

This class is used for the creation of `CategoryLabelPosition` instances.

### 23.12.3 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.

## 23.13 CategoryTick

### 23.13.1 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.

## 23.14 ColorBar

### 23.14.1 Overview

A *color bar* is used with a `ContourPlot`.

## 23.15 CompassFormat

### 23.15.1 Overview

A custom `NumberFormat` class that can be used to display numerical values as compass directions—see figure 23.5 for an example. In the example, the range

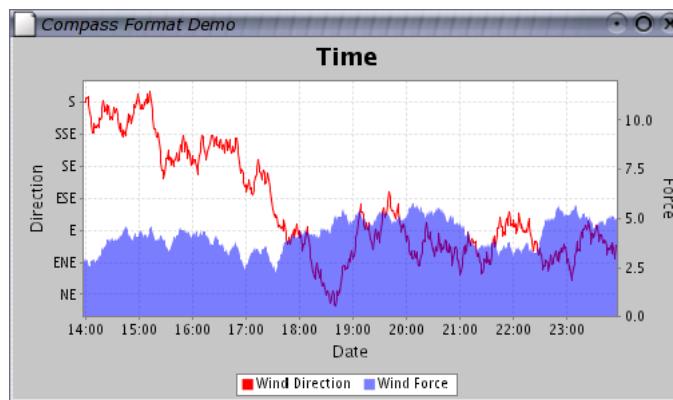


Figure 23.5: A chart that uses the `CompassFormat` class

axis on the left side of the chart displays compass directions in place of numerical values.

### 23.15.2 Usage

There is a demo (`CompassFormatDemo1.java`) included in the JFreeChart Premium Demo distribution.

## 23.16 CyclicNumberAxis

### 23.16.1 Overview

An extension of the `NumberAxis` class that is used to generate cyclic plots. See the `CyclicXYPlotDemo.java` file.

### 23.16.2 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.
```

### 23.16.3 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();
???

public void setBoundMappedToLastCycle(boolean boundMappedToLastCycle);
```

## 23.17 DateAxis

### 23.17.1 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.

### 23.17.2 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 [23.17.5](#);
- the size and formatting of the tick labels, see section [23.17.6](#);
- other inherited attributes, see section [23.41.4](#).

### 23.17.3 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.
```

### 23.17.4 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 <a href="#">DateTickUnit</a> ).
<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 [23.41.3](#) for information about the attributes inherited by this class.

### 23.17.5 The Axis Range

```
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>1</sup>

---

<sup>1</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.

```
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 (usu-
ally 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);
```

### 23.17.6 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);
```

### 23.17.7 Tick Label Orientation

To control the orientation of the tick labels on the axis:

```
axis.setVerticalTickLabels(true);
```

*This code survives from the HorizontalDateAxis class...it needs to be changed to be more generic for axes that could have either a horizontal or vertical orientation.*

### 23.17.8 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.
```

### 23.17.9 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.
```

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.

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.

```

### 23.17.10 Notes

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`).

## 23.18 DateTickMarkPosition

### 23.18.1 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.

## 23.19 DateTick

### 23.19.1 Overview

A class used to represent a single tick on a `DateAxis`.

### 23.19.2 Usage

This class is used internally and it is unlikely that you should ever need to use it directly.

## 23.20 DateTickUnit

### 23.20.1 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.

### 23.20.2 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.

### 23.20.3 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 [23.20.1](#). The `count` argument specifies the multiplier (often just 1).

### 23.20.4 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 23.20.1.
```

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.
```

### 23.20.5 Notes

This class is immutable, a requirement for all subclasses of [TickUnit](#).

#### See Also

[NumberTickUnit](#).

## 23.21 ExtendedCategoryAxis

### 23.21.1 Overview

An extension of the [CategoryAxis](#) class that allows sublabels to be displayed with the categories.

## 23.22 LogarithmicAxis

### 23.22.1 Overview

A numerical axis that displays values using a logarithmic scale. Extends [NumberAxis](#).

## 23.23 MarkerAxisBand

### 23.23.1 Overview

A band that can be added to a [NumberAxis](#) to highlight certain value ranges.

### 23.23.2 Usage

To use this class, first create a new band:

```
MarkerAxisBand band = new MarkerAxisBand(
    axis, 2.0, 2.0, 2.0, 2.0,
    new Font("SansSerif", Font.PLAIN, 9));
```

Next, add as many ranges as you require to be displayed on the axis:

```
IntervalMarker m1 = new IntervalMarker(0.0, 33.0,
                                         "Low", Color.gray,
                                         new BasicStroke(0.5f),
                                         Color.green, 0.75f);
band.addMarker(m1);

IntervalMarker m2 = new IntervalMarker(33.0, 66.0,
                                         "Medium", Color.gray,
                                         new BasicStroke(0.5f),
                                         Color.orange, 0.75f);
band.addMarker(m2);

IntervalMarker m3 = new IntervalMarker(66.0, 100.0,
                                         "High", Color.gray,
                                         new BasicStroke(0.5f),
                                         Color.red, 0.75f);
band.addMarker(m3);
```

## 23.24 ModuloAxis

### 23.24.1 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 distribution) provides one example of this axis in use—see figure 23.6.

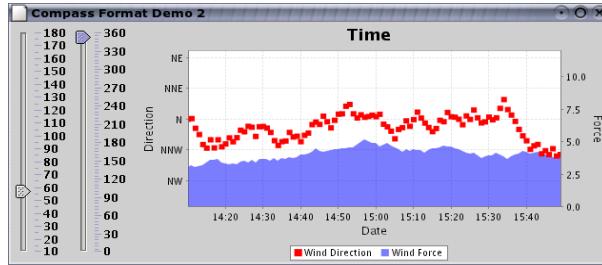


Figure 23.6: A chart that uses a `ModuloAxis`

### 23.24.2 Constructor

There is a single constructor:

```
public ModuloAxis(String label, Range fixedRange);
Creates a new axis with the specified label and fixedRange.
```

### 23.24.3 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.
```

### 23.24.4 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 per-
centage (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.
```

## 23.25 NumberAxis

### 23.25.1 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.

### 23.25.2 Constructors

To create a new axis:

```
public NumberAxis(String label);
Creates a new axis with the specified label (null permitted).
```

### 23.25.3 Usage

A `NumberAxis` can be used for the domain and/or range axes in an `XYPlot`, and for the range axis in a `CategoryPlot`.

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.

### 23.25.4 The Axis Range

You can control most aspects of the axis range using methods inherited from the `ValueAxis` class—see section 23.41.5 for details.

Two additional controls are added by this class. First, you can specify whether or not zero must be included in the axis range:

```
axis.setAutoRangeIncludesZero(true);
```

If the *auto-range-includes-zero* 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 *auto-range-sticky-zero* flag to `true`:

```
axis.setAutoRangeStickyZero(true);
```

...you can truncate the margin at zero.

### 23.25.5 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.

### 23.25.6 Attributes

The following table lists the properties maintained by `NumberAxis`, in addition to those inherited from `ValueAxis`.

Attribute:	Description:
<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 <a href="#">MarkerAxisBand</a> ).

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(0.0000001);</code>
<code>DEFAULT_TICK_UNIT</code>	<code>new NumberTickUnit(new Double(1.0), new DecimalFormat("0"));</code>

### 23.25.7 Methods

If you have set the *auto-range* 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 void setAutoRangeIncludesZero(boolean flag);
Sets the auto-range-includes-zero flag.
```

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.
```

You can reverse the direction of the values on the axis:

```
public void setInverted(boolean flag);
An inverted axis has values that run from high to low, the reverse of the normal case.
```

### 23.25.8 Notes

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](#).

## 23.26 NumberAxis3D

### 23.26.1 Overview

An extension of the `NumberAxis` class that adds a 3D effect. Eventually, this class will be combined with the `NumberAxis` class.

## 23.27 NumberTick

### 23.27.1 Overview

A class used to represent a single tick on a `NumberAxis`.

### 23.27.2 Usage

This class is used internally and it is unlikely that you should ever need to use it directly.

## 23.28 NumberTickUnit

### 23.28.1 Overview

A number tick unit for use by subclasses of `NumberAxis` (extends the `TickUnit` class).

### 23.28.2 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 = myChart.getXYPlot();
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).

### 23.28.3 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.
```

### 23.28.4 Methods

To format a value using the tick unit's internal formatter:

```
public String valueToString(double value);
Formats the value as a String.
```

### 23.28.5 Notes

This class is immutable, a requirement for all subclasses of [TickUnit](#).

#### See Also

[DateTickUnit](#).

## 23.29 PeriodAxis

### 23.29.1 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 23.7 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.

### 23.29.2 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.
```

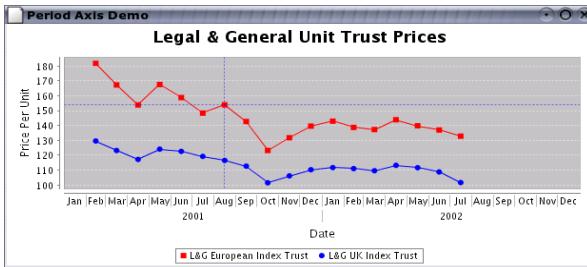


Figure 23.7: A chart that uses a *PeriodAxis*

### 23.29.3 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*.

### 23.29.4 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.

### 23.29.5 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.
```

### 23.29.6 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`.

### 23.29.7 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.

## See Also

[DateAxis](#), [PeriodAxisLabelInfo](#).

## 23.30 PeriodAxisLabelInfo

### 23.30.1 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.

### 23.30.2 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`.

### 23.30.3 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.
```

### 23.30.4 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`.

## 23.31 SegmentedTimeline

### 23.31.1 Overview

A segmented timeline for use with a [DateAxis](#).

### 23.31.2 Usage

Please refer to the Javadocs.

## 23.32 StandardTickUnitSource

### 23.32.1 Overview

A standard implementation of the [TickUnitSource](#) interface.

## 23.33 SubCategoryAxis

### 23.33.1 Overview

An extension of the [CategoryAxis](#) class that allows subcategories to be displayed. See the `StackedBarChartDemo4.java` file for an example.

## 23.34 SymbolicAxis

### 23.34.1 Overview

An axis that displays numerical data using symbols.

## 23.35 SymbolicTickUnit

### 23.35.1 Overview

Not yet documented.

## 23.36 Tick

### 23.36.1 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](#).

## 23.37 TickUnit

### 23.37.1 Overview

An abstract class representing a tick unit, with subclasses including:

- `DateTickUnit` – for use with a `DateAxis`;
- `NumberTickUnit` – for use with a `NumberAxis`.

### 23.37.2 Constructors

The standard constructor:

```
public TickUnit(double size);  
Creates a new tick unit with the specified size.
```

### 23.37.3 Notes

Implements the `Comparable` interface, so that a collection of tick units can be sorted easily using standard Java methods.

#### See Also

[TickUnits](#).

## 23.38 TickUnits

### 23.38.1 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.

### 23.38.2 Constructors

The default constructor:

```
public TickUnits();  
Creates a new collection of tick units, initially empty.
```

### 23.38.3 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.
```

### 23.38.4 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](#).

## 23.39 TickUnitSource

### 23.39.1 Overview

The interface through which an axis obtains standard tick units.

## 23.40 Timeline

### 23.40.1 Overview

The interface that defines the methods for a timeline that can be used with a [DateAxis](#).

### 23.40.2 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 time-
line 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 oth-
erwise.

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.
```

### 23.40.3 Notes

The [SegmentedTimeline](#) class implements this interface.

## 23.41 ValueAxis

### 23.41.1 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.

### 23.41.2 Constructors

The constructors for this class are protected, you cannot create a `ValueAxis` directly—you must use a subclass.

### 23.41.3 Attributes

The attributes maintained by this class, in addition to those that it inherits from the `Axis` class, are listed in Table 23.6. 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.

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>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>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 23.6: *Attributes for the `ValueAxis` class*

The default values used to initialise the axis attributes (when necessary) are listed in Table 23.7.

Name:	Value:
DEFAULT_AUTO_RANGE	true;
DEFAULT_MINIMUM_AXIS_VALUE	0.0;
DEFAULT_MAXIMUM_AXISVALUE	1.0;
DEFAULT_UPPER_MARGIN	0.05 (5 percent)
DEFAULT_LOWER_MARGIN	0.05 (5 percent)

Table 23.7: *ValueAxis* class default attribute values

#### 23.41.4 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:

```
CategoryPlot plot = myChart.getCategoryPlot();
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 23.41.5;

#### 23.41.5 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.

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. If you want to, you can turn off this default behaviour, using:

```
axis.setAutoRange(false);
```

You can exercise some control over the auto-range calculation. To set the upper and lower margins (a percentage of the overall axis range):

```
// set margins to 10 percent each...
axis.setLowerMargin(0.10);
axis.setUpperMargin(0.10);
```

### 23.41.6 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.
```

### 23.41.7 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);
>Converts a data value into a co-ordinate along one edge of the dataArea
>(the dataArea is the rectangle inside the plot's axes). Whether the coor-
>dinate relates to the (left) vertical or (bottom) horizontal edge, depends
>on the orientation of the axis subclass.
```

The inverse function converts a Java2D coordinate back to a data value:

```
public double java2DToValue(double java2DValue, Rectangle2D dataArea);
>Converts a Java2D coordinate back to a data value.
```

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.
```

To manually set the axis range (which automatically disables the *auto-range* flag):

```
public void setRange(Range range);
>Sets the axis range.
```

An alternative method that achieves the same thing:

```
public void setRange(double lower, double upper);
>Sets the axis range.
```

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.

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.

### 23.41.8 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](#).

## 23.42 ValueTick

### 23.42.1 Overview

The base class for the [NumberTick](#) and [DateTick](#) classes.

# Chapter 24

## Package: org.jfree.chart.block

### 24.1 Introduction

The `org.jfree.chart.block` package contains classes that are used for laying out rectangular items (blocks) within containers.

### 24.2 AbstractBlock

#### 24.2.1 Overview

A base class for implementing a “block”, which is used as a layout unit in JFreeChart (particularly for the `LegendTitle` class).

#### 24.2.2 Constructor

To create a new block:

```
protected AbstractBlock();  
Creates a new block.
```

#### 24.2.3 Methods

The following accessor methods are defined:

```
public String getID();  
Returns the block id.  
  
public void setID(String id);  
Sets the block id.  
  
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.
```

```
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 RectangleInsets getMargin();
>Returns the margin around the outside of the block’s border.

public void setMargin(RectangleInsets margin);
>Sets the margin around the outside of the block’s border.

public BlockBorder getBorder();
>Returns the border that will be drawn around the block.

public void setBorder(BlockBorder border);
>Sets the border that will be drawn around the block.

public RectangleInsets getPadding();
>Returns the padding between the block’s content and its border.

public void setPadding(RectangleInsets padding);
>Sets the padding between the block’s content and its border.

public Size2D arrange(Graphics2D g2);
>Arranges the block 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 and returns its size. Keep in mind that the block may
be a BlockContainer that contains other blocks.

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.

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.

protected void drawBorder(Graphics2D g2, Rectangle2D area);
>Draws the border for the block.
```

#### 24.2.4 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.
```

### 24.3 Arrangement

#### 24.3.1 Overview

A layout manager that can arrange blocks.

#### 24.3.2 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.
```

### 24.4 Block

#### 24.4.1 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.

#### 24.4.2 Methods

```
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.

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.

public Rectangle2D getBounds();
Gets the bounds for the block.

public void setBounds(Rectangle2D bounds);
Sets the bounds for the block.

```

## 24.5 BlockBorder

### 24.5.1 Overview

A simple border that can be assigned to any subclass of [AbstractBlock](#).

### 24.5.2 Constructors

There are two constructors:

```

public BlockBorder();
Creates a new block border, using insets of 1.0 on all four sides of the border.

public BlockBorder(RectangleInsets insets);
Creates a new block border using the specified insets.

```

### 24.5.3 Methods

```

public RectangleInsets getInsets();
Returns the insets that define the available drawing space for the border.

public void draw(Graphics2D g2, Rectangle2D area);
Draws the border around the edges of the specified area, always staying within the area.

public boolean equals(Object obj);
Tests this border for equality with an arbitrary object.

```

## 24.6 BlockContainer

### 24.6.1 Overview

A container for blocks that uses an [Arrangement](#) to organise the layout of the blocks.

### 24.6.2 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.

```

### 24.6.3 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.
```

```
public boolean isEmpty();
>Returns true if the container is empty (contains no blocks), and false otherwise.
```

```
public List getBlocks();
>Returns an unmodifiable list of the blocks in the container.
```

```
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).
```

```
public void clear();
>Clears all the blocks in the container.
```

```
public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);
>Arranges the blocks in the container, subject to the specified constraint.
```

```
public void draw(Graphics2D g2, Rectangle2D area);
>Draws the blocks within the specified area.
```

### 24.6.4 Equals, Cloning and Serialization

```
public boolean equals(Object obj);
>Returns true if this container is equal to obj and false otherwise.
```

```
public Object clone() throws CloneNotSupportedException;
>Returns a clone of the container.
```

## 24.7 BorderArrangement

### 24.7.1 Overview

A layout manager ([Arrangement](#)) that is similar to the `BorderLayout` class in AWT.

### 24.7.2 Constructor

To create a new instance:

```
public BorderArrangement();
>Creates a new layout manager.
```

### 24.7.3 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.

## 24.8 ColorBlock

### 24.8.1 Overview

A simple block that is filled with a color. This is a useful class for visual testing of layout classes.

### 24.8.2 Constructor

To create a new block:

```
public ColorBlock(Paint paint, double width, double height);
```

Creates a new block with the specified “preferred” dimensions.

### 24.8.3 Methods

To draw the block:

```
public void draw(Graphics2D g2, Rectangle2D area);
```

Draws the block inside the given `area`.

## 24.9 ColumnArrangement

### 24.9.1 Overview

An overview.

```
public ColumnArrangement();
```

```
public ColumnArrangement(HorizontalAlignment hAlign, VerticalAlignment
vAlign, double hGap, double vGap);
```

```
public void add(Block block, Object key);
```

```
public void arrange(BlockContainer container, RectangleConstraint constraint,
Graphics2D g2);
```

```

public void clear();

public boolean equals(Object obj);

```

## 24.10 EmptyBlock

### 24.10.1 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.

```

### 24.10.2 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.

```

## 24.11 FlowArrangement

### 24.11.1 Overview

An overview.

```

public FlowArrangement();

public FlowArrangement(HorizontalAlignment hAlign, VerticalAlignment vAlign,
double hGap, double vGap);

public void add(Block block, Object key);

public void arrange(BlockContainer container, RectangleConstraint constraint,
Graphics2D g2);

public void clear();

public boolean equals(Object obj);

```

## 24.12 GridArrangement

### 24.12.1 Overview

A layout manager ([Arrangement](#)) that places blocks within a fixed size grid.

### 24.12.2 Constructor

To create a new instance:

```
public GridArrangement(int rows, int columns);
```

Creates a new instance with the specified number of rows and columns.

### 24.12.3 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](#)

## 24.13 LabelBlock

### 24.13.1 Overview

A label that can be incorporated into a block layout.

### 24.13.2 Constructors

To create a new instance:

```
public LabelBlock(String label);
```

Creates a new label block with a default font.

```
public LabelBlock(String label, Font font);
```

Creates a new label block with the specified `font`.

### 24.13.3 Methods

A label can be placed within a layout by a layout manager that calls the following method:

```
public Size2D arrange(Graphics2D g2, RectangleConstraint constraint);
```

Fits the label block to the specified constraints, and returns the dimensions.

To draw the label block:

```
public void draw(Graphics2D g2, Rectangle2D area);
```

Draws the label within the specified `area`.

### 24.13.4 Notes

Some points to note:

- this class implements the [Block](#) interface, and thus supports margins, borders and padding as do all blocks.

## 24.14 LengthConstraintType

### 24.14.1 Overview

This class defines three constraint types:

- `LengthConstraintType.NONE`;
- `LengthConstraintType.FIXED`;
- `LengthConstraintType.RANGE`;

These types are used when creating `RectangleConstraint` instances.

### 24.14.2 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.
```

## 24.15 RectangleConstraint

### 24.15.1 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` (indicated by the `LengthConstraintType` class). These constraints are used by the layout code implemented by JFreeChart.

### 24.15.2 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.
```

### 24.15.3 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).
```

### 24.15.4 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 “con-
strained” dimensions.

public String toString();
Returns a string representing this class, primarily for debugging purposes.
```

# Chapter 25

## Package: `org.jfree.chart.entity`

### 25.1 Introduction

The `org.jfree.chart.entity` package contains classes that represent entities in a chart.

### 25.2 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.

### 25.3 CategoryItemEntity

#### 25.3.1 Overview

This class is used to convey information about an item within a category plot. The information captured includes the area occupied by the item, the tool tip and URL text (if any) generated for the item, the dataset, and the series and category that the item represents.

### 25.3.2 Constructors

To construct a new instance:

```
public CategoryItemEntity(Shape area, String toolTipText, String urlText,
    CategoryDataset dataset, int series, Object category, int categoryIndex);
Creates a new entity instance.
```

### 25.3.3 Methods

Accessor methods are implemented for the `dataset`, `series` and `category` attributes. Other methods are inherited from the `ChartEntity` class.

### 25.3.4 Notes

Most `CategoryItemRenderer` implementations will generate entities using this class, as required.

#### See Also

[ChartEntity](#), [CategoryPlot](#).

## 25.4 ChartEntity

### 25.4.1 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.

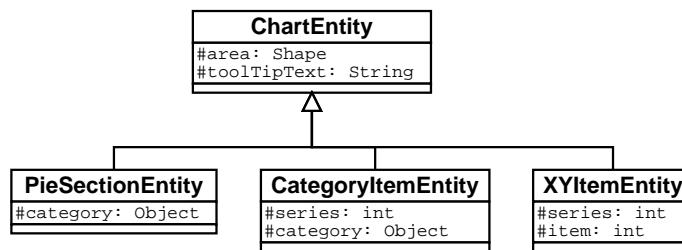


Figure 25.1: Chart entity classes

### 25.4.2 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.

### 25.4.3 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.

### 25.4.4 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](#).

## 25.5 ContourEntity

### 25.5.1 Overview

Not yet documented.

## 25.6 EntityCollection

### 25.6.1 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).

### 25.6.2 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.
```

### 25.6.3 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](#).

## 25.7 LegendItemEntity

### 25.7.1 Overview

An entity that records information about a legend item.

## 25.8 PieSectionEntity

### 25.8.1 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.

### 25.8.2 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.
```

### 25.8.3 Methods

Accessor methods are implemented for the `dataset`, `pieIndex`, `sectionIndex` and `sectionKey` attributes. Other methods are inherited from the [ChartEntity](#) class.

### 25.8.4 Notes

The [PiePlot](#) class generates pie section entities as required.

#### See Also

[ChartEntity](#), [PiePlot](#).

## 25.9 StandardEntityCollection

### 25.9.1 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.

### 25.9.2 Methods

This class implements the methods in the [EntityCollection](#) interface.

### 25.9.3 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](#).

## 25.10 TickLabelEntity

### 25.10.1 Overview

An entity that records information about a tick label.

## 25.11 XYItemEntity

### 25.11.1 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.

### 25.11.2 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.
```

### 25.11.3 Methods

Accessor methods are implemented for the `dataset`, `series` and `item` attributes. Other methods are inherited from the [ChartEntity](#) class.

### 25.11.4 Notes

Most [XYItemRenderer](#) implementations will generate entities using this class, as required.

#### See Also

[ChartEntity](#), [XYPlot](#).

---

<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.

# Chapter 26

## Package: org.jfree.chart.event

### 26.1 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.

### 26.2 AxisChangeEvent

#### 26.2.1 Overview

An event that can be sent to an `AxisChangeListener` to provide information about a change to an axis.

#### 26.2.2 Notes

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).

### 26.3 AxisChangeListener

#### 26.3.1 Overview

An interface through which axis change event notifications are posted.

#### 26.3.2 Methods

The interface defines a single method:

```
public void axisChanged(AxisChangeEvent event);  
Receives notification of a change to an axis.
```

### 26.3.3 Notes

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.

## 26.4 ChartChangeEvent

### 26.4.1 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.

### 26.4.2 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, ChartChangeEvent.Type  
type);  
Creates a new event with the specified type.
```

### 26.4.3 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 ChartChangeEvent.Type getType();  
Returns the event type.  
  
public void setType(ChartChangeEvent.Type type);  
Sets the event type.
```

### 26.4.4 Notes

The `ChartPanel` class automatically registers itself with the chart it is displaying. When it receives a `ChartChangeEvent`, it repaints the chart.

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 26.1: ChartChangeEvent.Type tokens*

## 26.5 ChartChangeEvent.Type

### 26.5.1 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.

## 26.6 ChartChangeListener

### 26.6.1 Overview

An interface through which chart change event notifications are posted.

### 26.6.2 Methods

The interface defines a single method:

```
public void chartChanged(ChartChangeEvent event);
```

Receives notification of a change to a chart.

### 26.6.3 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.

## 26.7 ChartProgressEvent

### 26.7.1 Overview

Not yet documented.

## 26.8 ChartProgressListener

### 26.8.1 Overview

Not yet documented.

## 26.9 LegendChangeEvent

### 26.9.1 Overview

An event that is used to provide information about changes to a legend.

#### See Also

[LegendChangeListener](#).

## 26.10 LegendChangeListener

### 26.10.1 Overview

An interface through which legend change event notifications are posted.

### 26.10.2 Methods

The interface defines a single method:

```
public void legendChanged(LegendChangeEvent event);
```

Receives notification of a change to a legend.

### 26.10.3 Notes

If a class needs to receive notification of changes to a legend, then it needs to implement this interface and register itself with the legend.

#### See Also

[LegendChangeEvent](#).

## 26.11 PlotChangeEvent

### 26.11.1 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.

### 26.11.2 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).

## 26.12 PlotChangeListener

### 26.12.1 Overview

An interface through which plot change event notifications are posted.

### 26.12.2 Methods

The interface defines a single method:

```
public void plotChanged(PlotChangeEvent event);
```

Receives notification of a change to a plot.

### 26.12.3 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.

## 26.13 RendererChangeEvent

### 26.13.1 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.

In the default setup, a change to a renderer will cause the plot to receive notification of the event. The plot will usually respond by firing a `PlotChangeEvent` (which usually gets passed on to the chart and results in a `ChartChangeEvent` being fired).

### 26.13.2 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.

## 26.14 RendererChangeListener

### 26.14.1 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).

### 26.14.2 Methods

The interface defines a single method:

```
public void rendererChanged(RendererChangeEvent event);
```

Receives notification of a change to a renderer.

### 26.14.3 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.

## 26.15 TitleChangeEvent

### 26.15.1 Overview

An event that is used to provide information about changes to a chart title (any subclass of `Title`).

### 26.15.2 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](#).

## 26.16 TitleChangeListener

### 26.16.1 Overview

An interface through which title change event notifications are posted.

### 26.16.2 Methods

The interface defines a single method:

```
public void titleChanged(TitleChangeEvent event);
```

Receives notification of a change to a title.

### 26.16.3 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 27

## Package: org.jfree.chart.imageMap

### 27.1 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.

### 27.2 DynamicDriveToolTipTagFragmentGenerator

#### 27.2.1 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.

### 27.3 ImageMapUtilities

#### 27.3.1 Overview

This class contains some utility methods that are useful for creating HTML image maps.

#### 27.3.2 Methods

```
public static void writeImageMap(PrintWriter writer, String name,
ChartRenderingInfo info);
Writes an image map using info as the source of chart entity information.

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.
```

```
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.

public static String getImageMap(String name, ChartRenderingInfo info);
Returns an image map based on the chart entity information in info.

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.
```

## 27.4 OverLIBToolTipTagFragmentGenerator

### 27.4.1 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.

## 27.5 StandardToolTipTagFragmentGenerator

### 27.5.1 Overview

A tool-tip generator that generates tool-tips using the HTML title attribute.  
This class implements the `ToolTipTagFragmentGenerator` interface.

## 27.6 StandardURLTagFragmentGenerator

### 27.6.1 Overview

A standard implementation of the `URLTagFragmentGenerator` interface.

## 27.7 ToolTipTagFragmentGenerator

### 27.7.1 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`;

### 27.7.2 Methods

This interface defines a single method:

```
public String generateToolTipFragment(String toolTipText);
```

Returns a tooltip fragment based on the supplied tool-tip text.

## 27.8 URLLTagFragmentGenerator

### 27.8.1 Overview

The interface that must be implemented by a class that generates URL tag fragments for an HTML image map.

The [StandardURLTagFragmentGenerator](#) class provides one implementation of this interface.

### 27.8.2 Methods

This interface defines a single method:

```
public String generateURLFragment(String urlText);
```

Returns a URL fragment based on the supplied URL text.

# Chapter 28

## Package: org.jfree.chart.labels

### 28.1 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 10 contains information about using tool tips and section 11 contains information about using item labels.

### 28.2 AbstractCategoryItemLabelGenerator

#### 28.2.1 Overview

An abstract base class for creating item labels for a `CategoryItemRenderer`. Both the `StandardCategoryToolTipGenerator` and `StandardCategoryLabelGenerator` classes extend this class.

The generator uses Java’s `MessageFormat` class to construct labels by substituting any or all of the objects listed in table 28.1.

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.

Code:	Description:
{0}	The series name.
{1}	The category label.
{2}	The (preformatted) data value.

Table 28.1: `MessageFormat` substitutions

### 28.2.2 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 28.1.

```
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 re-
quired 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.
```

### 28.2.3 Notes

Some points to note:

- the `StandardCategoryToolTipGenerator` and `StandardCategoryLabelGenerator` classes are extensions of this class;
- instances of this class are `Cloneable` and `Serializable`.

## 28.3 AbstractXYItemLabelGenerator

### 28.3.1 Overview

An abstract base class for creating item labels for an `XYItemRenderer`. Both the `StandardXYToolTipGenerator` and `StandardXYLabelGenerator` classes extend this class.

The generator uses Java's `MessageFormat` class to construct labels by substituting any or all of the objects listed in table 28.2.

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.

Code:	Description:
{0}	The series name.
{1}	The (preformatted) x-value.
{2}	The (preformatted) y-value.

Table 28.2: *MessageFormat substitutions*

### 28.3.2 Constructors

Various constructors are provided that 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 28.2.

```

protected AbstractXYItemLabelGenerator();
Creates a new generator that formats the data values using the default
number formatter for the current locale.

public AbstractXYItemLabelGenerator(String formatString,
NumberFormat xFormat, NumberFormat yFormat);
Creates a new generator that formats the data values using the supplied
NumberFormat instances.

public 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, DateFormat yFormat);
Creates a new generator that formats both the x and y values as dates.

```

### Methods

To generate a label string:

```

protected String generateLabelString(XYDataset dataset, int series, int
item);
Generates a label string. This method first calls the createItemArray()
function, then passes the result to Java's MessageFormat to build the re-
quired 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.

```

### 28.3.3 Notes

Some points to note:

- the `StandardXYToolTipGenerator` and `StandardXYLabelGenerator` classes are extensions of this class;
- instances of this class are `Cloneable` and `Serializable`.

## 28.4 BoxAndWhiskerToolTipGenerator

### 28.4.1 Overview

A *tool tip generator* for a box-and-whisker chart. This is the default generator used by the `BoxAndWhiskerRenderer` class.

## 28.5 BoxAndWhiskerXYToolTipGenerator

### 28.5.1 Overview

A *tool tip generator* for a box-and-whisker chart. This is the default generator used by the `XYBoxAndWhiskerRenderer` class.

## 28.6 CategoryLabelGenerator

### 28.6.1 Overview

A *category 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 `setLabelGenerator()` method in the `CategoryItemRenderer` interface. This interface defines the API through which the renderer will communicate with the generator.

### 28.6.2 Usage

Chapter 11 contains information about using item labels.

### 28.6.3 Methods

The renderer will call this method to obtain an item label:

```
public String generateItemLabel(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.

### 28.6.4 Notes

Some points to note:

- the `StandardCategoryLabelGenerator` 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.

## 28.7 CategoryToolTipGenerator

### 28.7.1 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.

### 28.7.2 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.
```

### 28.7.3 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 10 for information about using tool tips.

## 28.8 ContourToolTipGenerator

### 28.8.1 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.

### 28.8.2 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.
```

## 28.9 CustomXYToolTipGenerator

### 28.9.1 Overview

A tool tip generator (for use with an `XYItemRenderer`) that returns a predefined tool tip for each data item.

### 28.9.2 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.
```

### 28.9.3 Notes

See section 10 for information about using tool tips with JFreeChart.

## 28.10 HighLowItemLabelGenerator

### 28.10.1 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 `HighLowDataset` interface.

### 28.10.2 Constructors

To create a new label generator:

```
public HighLowItemLabelGenerator(DateFormatter dateFormatter, NumberFormat  
numberFormatter);  
Creates a new label generator that uses the specified date and number  
formatters.
```

### 28.10.3 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 HighLowDataset 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.
```

### 28.10.4 Notes

See section 10 for an overview of tool tips with JFreeChart.

## 28.11 IntervalCategoryLabelGenerator

### 28.11.1 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.

### 28.11.2 Constructors

The default constructor will create a label generator that formats the data values as numbers, using the platform default number format:

```
public IntervalCategoryLabelGenerator();
Creates a new label generator with a default number formatter.
```

If you prefer to set the number format yourself, use the following constructor:

```
public IntervalCategoryLabelGenerator(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 IntervalCategoryLabelGenerator(DateFormat formatter);
Creates a new label generator that formats the start and end data values as dates.
```

### 28.11.3 Notes

The `createGanttChart()` in the `ChartFactory` class uses this type of label generator (with date formatting).

## 28.12 IntervalCategoryToolTipGenerator

### 28.12.1 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.

### 28.12.2 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.
```

### 28.12.3 Notes

The `createGanttChart()` in the `ChartFactory` class uses this type of label generator (with date formatting).

## 28.13 ItemLabelAnchor

### 28.13.1 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 28.1 for an illustration.

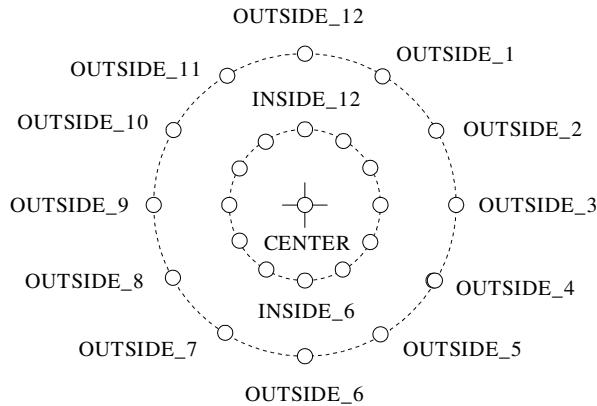


Figure 28.1: The Item Label Anchors

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).

### 28.13.2 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.

## 28.14 ItemLabelPosition

### 28.14.1 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.

### 28.14.2 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.
```

## 28.15 PieSectionLabelGenerator

### 28.15.1 Overview

A *pie section label generator* is an object that assumes responsibility for generating labels for the sections in a [PiePlot](#). This interface defines the method used by the plot to request a section label. The [StandardPieItemLabelGenerator](#) class provides an implementation of this interface.

### 28.15.2 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 im-
    plementing this method can return null, in which case no label will be
    displayed for the pie section.
```

### 28.15.3 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.

## 28.16 PieToolTipGenerator

### 28.16.1 Overview

The interface that must be implemented by a *pie tool tip generator*, a class used to generate tool tips for a pie chart.

### 28.16.2 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.
```

### 28.16.3 Notes

Some points to note:

- the `StandardPieItemLabelGenerator` 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 10 contains information about using tool tips with JFreeChart.

## 28.17 StandardCategoryLabelGenerator

### 28.17.1 Overview

A generator that can be assigned to a `CategoryItemRenderer` for the purpose of generating item labels (this class implements the `CategoryLabelGenerator` 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 28.1. The data value can be formatted using any `NumberFormat` instance.

### 28.17.2 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();
CategoryLabelGenerator generator = new StandardCategoryLabelGenerator(
    "{2}", new DecimalFormat("0.00")
);
renderer.setLabelGenerator(generator);
renderer.setItemLabelsVisible(true);
```

The renderer will call the generator's methods when necessary. See section 11 for more information.

### 28.17.3 Constructors

To create a default generator:

```
public StandardCategoryLabelGenerator();
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 StandardCategoryLabelGenerator(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 28.1.
```

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 StandardCategoryLabelGenerator(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
28.1.
```

### 28.17.4 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.
```

### 28.17.5 Notes

Some points to note:

- instances of this class are cloneable and serializable, and the `PublicCloneable` interface is implemented;

## 28.18 StandardCategoryToolTipGenerator

### 28.18.1 Overview

A generator that can be assigned to a `CategoryItemRenderer` for the purpose of generating tooltips. This class implements the `CategoryToolTipGenerator` interface.

### 28.18.2 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();
renderer.setToolTipGenerator(new StandardCategoryToolTipGenerator());
```

The renderer will call the generator's methods when necessary.

### 28.18.3 Constructors

This class has a default constructor:

```
public StandardCategoryToolTipGenerator();
Creates a new generator that formats values 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 formats values using the supplied formatter.
```

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 formats values as dates using the supplied for-
matter.
```

### 28.18.4 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.
```

### 28.18.5 Notes

Some points to note:

- this class implements the `PublicCloneable` interface;
- section 10 contains information about using tool tips with JFreeChart.

Code:	Description:
{0}	The item key.
{1}	The item value.
{2}	The item value as a percentage of the total.

*Table 28.3: MessageFormat substitutions*

## 28.19 StandardContourToolTipGenerator

### 28.19.1 Overview

A default implementation of the [ContourToolTipGenerator](#) interface.

## 28.20 StandardPieItemLabelGenerator

### 28.20.1 Overview

A label generator that can be used to generate section labels and tool tips for a [PiePlot](#) (implements [PieSectionLabelGenerator](#) and [PieToolTipGenerator](#)).

The generator uses Java's [MessageFormat](#) class to construct labels by substituting any or all of the objects listed in table 28.3.

The default tool tip format string is "{0}: ({1}, {2})", which displays the item key, followed by the item value and percentage. Similarly, the default section label format is "{0} = {1}", which displays the item key followed by the item value (the percentage is not displayed).

### 28.20.2 Usage

You can use this class when you want to change the format of the the section labels or tool tips on a pie chart. For example, to change the section labels:

```
PiePlot plot = (PiePlot) chart.getPlot();
PieSectionLabelGenerator generator = new StandardPieItemLabelGenerator(
    "{0} = {2}", new DecimalFormat("0"), new DecimalFormat("0.00%")
);
plot.setLabelGenerator(generator);
```

### 28.20.3 Constructors

The default constructor uses number and percentage formatters appropriate for the default locale:

```
public StandardPieItemLabelGenerator();
Creates a default label generator.
```

You can create a generator with a specific format string:

```
public StandardPieItemLabelGenerator(String labelFormat);
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 current locale.
```

The final constructor allows you to specify the item value and percentage formatters:

```
public StandardPieItemLabelGenerator(String labelFormat,
NumberFormat numberFormat, NumberFormat percentFormat)
Creates a generator using the specified format string, with custom formatters for the item value and item percentage.
```

#### 28.20.4 Notes

Some points to note:

- instances of this class are cloneable and serializable;
- section 10 contains information about using tool tips with JFreeChart.

### 28.21 StandardXYLabelGenerator

#### 28.21.1 Overview

A generator that can be assigned to an `XYItemRenderer` for the purpose of generating item labels (this class implements the `XYLabelGenerator` 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 28.2. The x and y values can be formatted using any instance of `NumberFormat` or `DateFormat`.

#### 28.21.2 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();
XYLabelGenerator generator = new StandardXYLabelGenerator(
    "{2}", new DecimalFormat("0.00"), new DecimalFormat("0.00")
);
renderer.setLabelGenerator(generator);
renderer.setItemLabelsVisible(true);
```

The renderer will call the generator's methods when necessary. See section 11 for more information.

#### 28.21.3 Constructors

To create a default generator:

```
public StandardXYLabelGenerator();
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 StandardXYLabelGenerator(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 28.2.
```

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 StandardXYLabelGenerator(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 28.2.
```

#### 28.21.4 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.
```

#### 28.21.5 Notes

Some points to note:

- instances of this class are cloneable and serializable, and the `PublicCloneable` interface is implemented;

### 28.22 StandardXYToolTipGenerator

#### 28.22.1 Overview

A generator that can be assigned to an `XYItemRenderer` for the purpose of generating tool tips (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 28.2. The x and y values can be formatted using any instance of `NumberFormat` or `DateFormat`.

#### 28.22.2 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 10 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);
```

### 28.22.3 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 28.2.
```

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
28.2.
```

### 28.22.4 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.
```

### 28.22.5 Notes

Some points to note:

- instances of this class are cloneable and serializable, and the `PublicCloneable` interface is implemented;

## 28.23 StandardXYZToolTipGenerator

### 28.23.1 Overview

A default implementation of the `XYZItemLabelGenerator` interface. This generator is used with the `XYBubbleRenderer` class.

## 28.24 SymbolicXYItemLabelGenerator

### 28.24.1 Overview

An item label generator for use with symbolic plots.

## 28.25 XYLabelGenerator

### 28.25.1 Overview

An *xy label generator* is an object that assumes responsibility for generating the text strings that will be used for the item labels in a chart. A generator is assigned to a renderer using the `setLabelGenerator()` method in the `XYItemRenderer` interface.

### 28.25.2 Usage

Chapter 11 contains information about using item labels.

### 28.25.3 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.

### 28.25.4 Notes

Some points to note:

- the `StandardXYLabelGenerator` 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;

## 28.26 XYToolTipGenerator

### 28.26.1 Overview

The interface that must be implemented by an *XY tool tip generator*, a class used to generate tool tips for an `XYPlot`.

### 28.26.2 Methods

The plot will call the following method whenever it requires a tool tip for an item:

```
public String generateToolTip(XYDataset data, 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.

### 28.26.3 Notes

Some points to note:

- to “install” a tool tip generator, use the `setToolTipGenerator()` method in the `XYItemRenderer` interface.
- `StandardXYZToolTipGenerator` implements this interface, but you are free to write your own implementation to suit your requirements.

Section 10 contains information about using tool tips with JFreeChart.

## 28.27 XYZToolTipGenerator

### 28.27.1 Overview

A tool tip generator that creates labels for items in an `XYZDataset`.

### 28.27.2 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.
```

### 28.27.3 Notes

Some points to note:

- this interface extends `XYToolTipGenerator`;
- the `StandardXYZToolTipGenerator` class is the only implementation of this interface provided by JFreeChart.

# Chapter 29

## Package: org.jfree.chart.needle

### 29.1 Overview

This package contains classes for drawing needles in a compass plot:

- `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;

## 29.2 ArrowNeedle

### 29.2.1 Overview

A class that draws an *arrow needle* for the `CompassPlot` class (see figure 29.1).

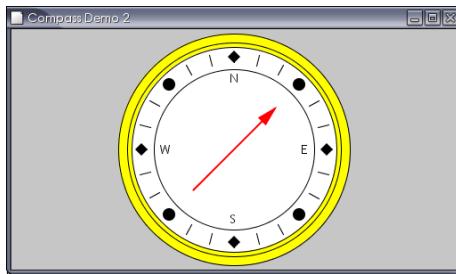


Figure 29.1: An arrow needle

## 29.3 LineNeedle

### 29.3.1 Overview

A class that draws a *line needle* for the `CompassPlot` class (see figure 29.2).

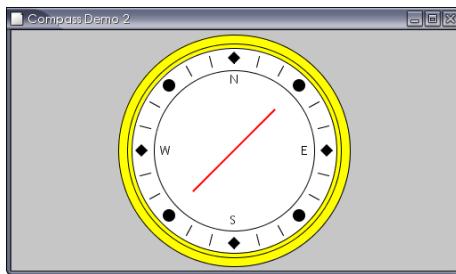


Figure 29.2: A line needle

## 29.4 LongNeedle

### 29.4.1 Overview

A class that draws a *long needle* for the `CompassPlot` class (see figure 29.3).

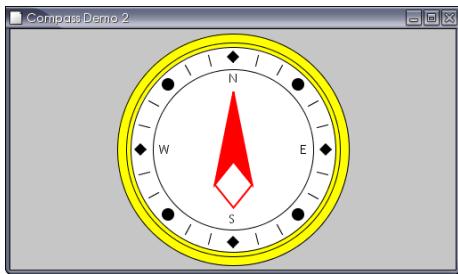


Figure 29.3: A long needle

## 29.5 MeterNeedle

### 29.5.1 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;

## 29.6 PinNeedle

### 29.6.1 Overview

A class that draws a *pin needle* for the [CompassPlot](#) class (see figure 29.4).

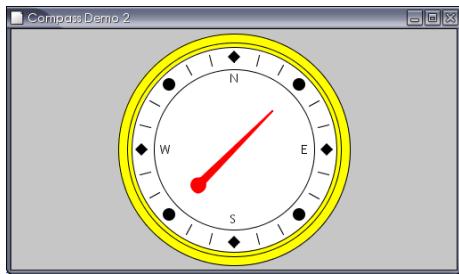


Figure 29.4: A pin needle

## 29.7 PlumNeedle

### 29.7.1 Overview

A class that draws a *plum needle* for the [CompassPlot](#) class (see figure 29.5).

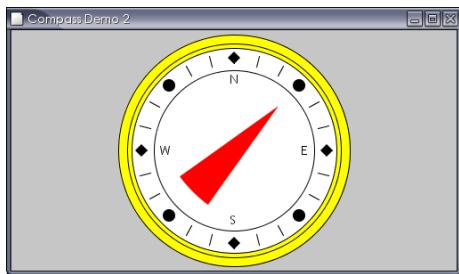


Figure 29.5: A plum needle

## 29.8 PointerNeedle

### 29.8.1 Overview

A class that draws a *pointer needle* for the [CompassPlot](#) class (see figure 29.6).

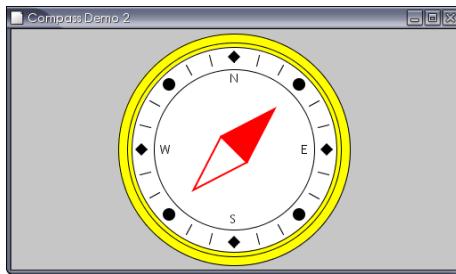


Figure 29.6: A pointer needle

## 29.9 ShipNeedle

### 29.9.1 Overview

A class that draws a *ship needle* for the [CompassPlot](#) class (see figure 29.7).

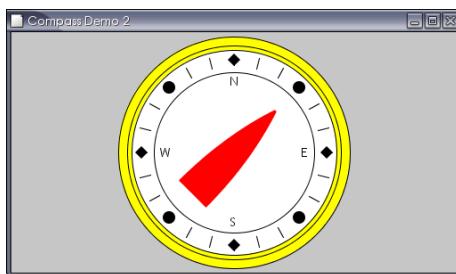


Figure 29.7: A ship needle

## 29.10 WindNeedle

### 29.10.1 Overview

A class that draws a *wind needle* for the `CompassPlot` class (see figure 29.8).

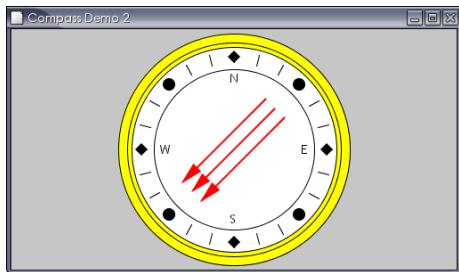


Figure 29.8: A wind needle

# Chapter 30

## Package: org.jfree.chart.plot

### 30.1 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.

### 30.2 CategoryPlot

#### 30.2.1 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).

### 30.2.2 Attributes

The attributes maintained by the `CategoryPlot` class, which are in addition to those inherited from the `Plot` class, are listed in Table 30.1.

Attribute:	Description:
<code>orientation</code>	The plot orientation (horizontal or vertical).
<code>axisOffset</code>	The offset between the data area and the axes.
<code>domainAxes</code>	The domain axes (used to display categories).
<code>domainAxisLocations</code>	The locations of the domain axes.
<code>rangeAxes</code>	The range axes (used to display values).
<code>rangeAxisLocations</code>	The locations of the range axes.
<code>datasets</code>	The dataset(s).
<code>renderers</code>	The plot's renderers ("pluggable" objects responsible for drawing individual data items within the plot).
<code>renderingOrder</code>	The order for rendering data items (see <code>DatasetRenderingOrder</code> ). ???
<code>columnRenderingOrder</code>	???
<code>rowRenderingOrder</code>	???
<code>domainGridlinesVisible</code>	A flag that controls whether gridlines are drawn against the domain axis.
<code>domainGridlinePosition</code>	The position of the gridlines against the domain axis.
<code>domainGridlinePaint</code>	The paint used to draw the domain gridlines.
<code>domainGridlineStroke</code>	The stroke used to draw the domain gridlines.
<code>rangeGridlinesVisible</code>	A flag that controls whether gridlines are drawn against the range axis.
<code>rangeGridlinePaint</code>	The paint used to draw the range gridlines.
<code>rangeGridlineStroke</code>	The stroke used to draw the range gridlines.
<code>foregroundRangeMarkers</code>	A list of markers (constants) to be highlighted on the plot.
<code>backgroundRangeMarkers</code>	A list of markers (constants) to be highlighted on the plot.
<code>weight</code>	The weight for the plot (only used when the plot is a subplot).
<code>fixedDomainAxisSpace</code>	Specifies a fixed amount of space to allocate to the domain axis ( <code>null</code> permitted).
<code>fixedRangeAxisSpace</code>	Specifies a fixed amount of space to allocate to the range axis ( <code>null</code> permitted).

Table 30.1: Attributes for the `CategoryPlot` class

### 30.2.3 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`.

### 30.2.4 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);
```

The axis classes have many attributes that can be customised to control the appearance of your charts.

The axes can be offset slightly from the edges of the plot area, if required. Use the following methods:

```
public Spacer getAxisOffset();
```

Returns the object that controls the offset between the plot area and the axes.

```
public void setAxisOffset(Spacer offset);
```

Sets the object that controls the offset between the plot area and the axes, and sends a `PlotChangeEvent` to all registered listeners.

### 30.2.5 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`.

### 30.2.6 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.

### 30.2.7 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));
```

### 30.2.8 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.

### 30.2.9 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.
```

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.

### 30.2.10 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).
```

### 30.2.11 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 spec-
ified 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.
```

### 30.2.12 Notes

A number of `CategoryItemRenderer` implementations are included in the JFreeChart distribution.

#### See Also

`CombinedDomainCategoryPlot`, `CombinedRangeCategoryPlot`.

## 30.3 CombinedDomainCategoryPlot

### 30.3.1 Overview

A *category plot* that allows multiple subplots to be displayed together using a shared domain axis—see figure 30.1 for an example.

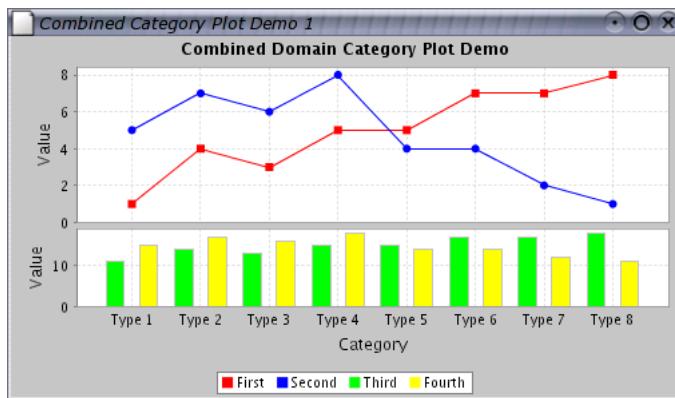


Figure 30.1: A `CombinedDomainCategoryPlot`

### 30.3.2 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 per-
mitted).
```

After creating a new parent plot, you need to add some subplots.

### 30.3.3 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.
```

```
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 being added to the `CombinedDomainCategoryPlot` can be any instance of `CategoryPlot` and will have its domain axis 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 getSubplots();
    Returns an unmodifiable list of the subplots.
```

### 30.3.4 Notes

The `CombinedCategoryPlotDemo1.java` file (included in the JFreeChart Premium Demo distribution) provides an example of this type of plot.

#### See Also

[CombinedRangeCategoryPlot](#).

## 30.4 CombinedDomainXYPlot

### 30.4.1 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 30.2 illustrates the relationship between the `CombinedDomainXYPlot` and its subplots).

The `CombinedXYPlotDemo1` class (included in the JFreeChart Premium Demo distribution) provides an example of this type of plot.

### 30.4.2 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.

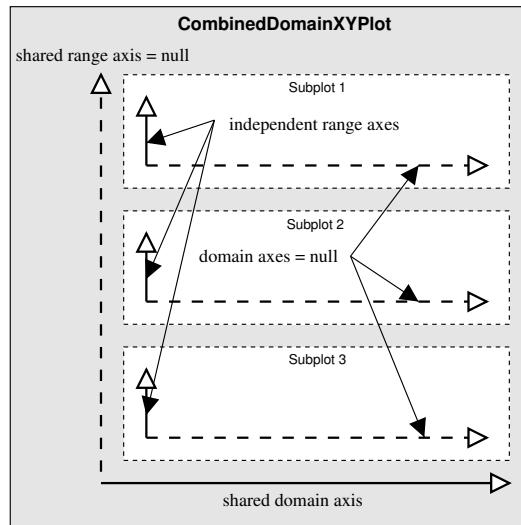


Figure 30.2: `CombinedDomainXYPlot` axes

### 30.4.3 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.

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.
```

The subplot being added to the `CombinedDomainXYPlot` can be any instance of `XYPlot` and will have its domain axis 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.
```

### 30.4.4 The Plot Orientation

To set the plot orientation:

```
public void setOrientation(PlotOrientation orientation);
    Sets the orientation of this plot and all its subplots.
```

### 30.4.5 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.
```

### 30.4.6 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](#).

## 30.5 CombinedRangeCategoryPlot

### 30.5.1 Overview

A *category plot* that allows multiple subplots to be displayed together using a shared range axis—see figure 30.3 for an example.

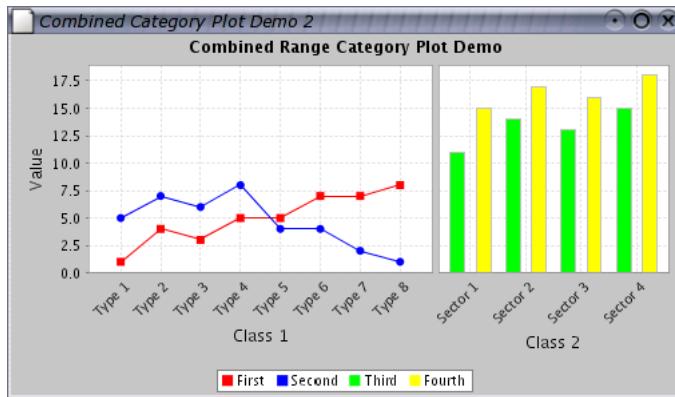


Figure 30.3: A *CombinedRangeCategoryPlot*

### 30.5.2 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.

### 30.5.3 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.

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 being added to the `CombinedRangeCategoryPlot` can be any instance of `CategoryPlot` and will have its range axis 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 getSubplots();
Returns an unmodifiable list of the subplots.
```

### 30.5.4 Notes

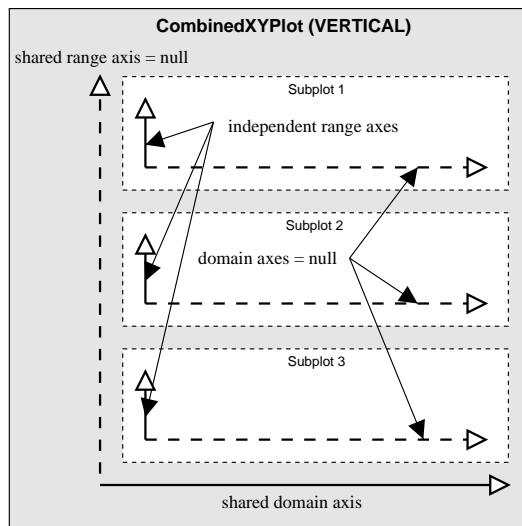
The `CombinedCategoryPlotDemo2.java` file (included in the JFreeChart Premium Demo distribution) provides an example of this type of plot.

## 30.6 CombinedRangeXYPlot

### 30.6.1 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 30.4 illustrates the relationship between the `CombinedRangeXYPlot` and its subplots).



*Figure 30.4: CombinedRangeXYPlot axes*

The `CombinedRangeXYPlotDemo` class provides an example of this type of plot.

### 30.6.2 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.

public void add(XYPlot subplot, int weight);
    Adds a subplot to the combined plot, with the specified weight.
```

The subplot being added to the `CombinedRangeXYPlot` can be any instance of `XYPlot` and should have one of its axes (the shared axis) 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 control the amount of space between the subplots:

```
public void setGap(double gap);
    Sets the gap (in points) between the subplots.
```

### 30.6.3 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 ??.

## 30.7 CompassPlot

### 30.7.1 Overview

A *compass plot* presents directional data in the form of a compass dial.

### 30.7.2 Notes

There is a demonstration `CompassDemo.java` application included in the JFreeChart Premium Demo distribution.

## 30.8 ContourPlot

### 30.8.1 Overview

A custom plot that displays  $(x, y, z)$  data in the form of a 2D contour plot.

## 30.9 ContourPlotUtilities

### 30.9.1 Overview

A class that contains static utility methods used by the contour plot implementation.

## 30.10 ContourValuePlot

### 30.10.1 Overview

An interface used by the contour plot implementation.

## 30.11 CrosshairState

### 30.11.1 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.

### 30.11.2 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.
```

### 30.11.3 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 can-
didate is the “closest” to the anchor point.
```

## 30.12 DatasetRenderingOrder

### 30.12.1 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 30.2.

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 30.2: *DatasetRenderingOrder* tokens

The default setting is `DatasetRenderingOrder.REVERSE`—this ensures that the primary dataset appears “on top” of the secondary datasets.

### 30.12.2 Usage

To change the rendering order, use the following code:

```
CategoryPlot plot = (CategoryPlot) chart.getPlot();
plot.setDatasetRenderingOrder(DatasetRenderingOrder.FORWARD);
```

### 30.12.3 Notes

Some points to note:

- an example (`OverlaidBarChartDemo.java`) is included in the premium demo collection.

## 30.13 DefaultDrawingSupplier

### 30.13.1 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).

### 30.13.2 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 colors, line styles and shapes.

## 30.14 DialShape

### 30.14.1 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 30.3.

Token:	Description:
<code>DialShape.CIRCLE</code>	A circle.
<code>DialShape.CHORD</code>	A chord.
<code>DialShape.PIE</code>	A pie.

Table 30.3: *DialShape* tokens

### 30.14.2 Usage

The `MeterPlot` class has a method named `setDialShape()` that accepts the tokens defined by this class.

## 30.15 DrawingSupplier

### 30.15.1 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).

### 30.15.2 Methods

To obtain the next `Paint` object in the sequence:

```
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 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 coordi-
nates.
```

## 30.16 FastScatterPlot

### 30.16.1 Overview

A custom plot that aims to be fast rather than flexible. 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.

This class is still at the “proof of concept” stage. It works reasonably well but doesn’t provide a lot of options.

### 30.16.2 Methods

This class overrides the `draw()` method defined in the `Plot` class:

```
public void draw(Graphics2D g2, Rectangle2D plotArea,
PlotState parentState, PlotRenderingInfo info);
Draws the plot in the specified area. You won't normally call this method
directly, it is called for you by the JFreeChart class.
```

### 30.16.3 Gridlines

You can display gridlines against the *domain axis* using the following methods:

```
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 void setDomainGridlinePaint(Paint paint);
Sets the Paint used for the domain gridlines and sends a PlotChangeEvent
to all registered listeners.

public void setDomainGridlineStroke(Stroke stroke);
Sets the Stroke used for the domain gridlines and sends a PlotChangeEvent
to all registered listeners.
```

Similarly, you can display gridlines against the *range axis*:

```
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 void setRangeGridlinePaint(Paint paint);
Sets the Paint used for the range gridlines and sends a PlotChangeEvent to
all registered listeners.

public void setRangeGridlineStroke(Stroke stroke);
Sets the Stroke used for the range gridlines and sends a PlotChangeEvent
to all registered listeners.
```

### 30.16.4 Notes

Some points to note:

- this plot does not support secondary axes;
- there is a demo (`FastScatterPlotDemo.java`) included in the JFreeChart Premium Demo distribution.

## 30.17 IntervalMarker

### 30.17.1 Overview

An *interval marker* 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.

### 30.17.2 Usage

There is a demo application (`DifferenceChartDemo2.java`) included in the JFreeChart Premium Demo distribution that illustrates the use of this class.

### 30.17.3 Notes

Some points to note:

- this class is `Cloneable` and `Serializable`.

## 30.18 Marker

### 30.18.1 Overview

The base class for markers that can be added to a `CategoryPlot` or an `XYPlot`. There are two subclasses, as listed in Table 30.4.

Class:	Description:
<code>ValueMarker</code>	A marker that highlights a single value.
<code>IntervalMarker</code>	A marker that highlights a range of values.

Table 30.4: Subclasses of `Marker`

Markers are used to highlight particular values or value ranges against either the domain or range axes. Labels can be added to the markers.

### 30.18.2 Usage

There is a demo application (`MarkerDemo1.java`) included in the JFreeChart Premium Demo distribution that illustrates the use of markers.

### 30.18.3 Notes

Some points to note:

- markers should be `Cloneable` and `Serializable`.

## 30.19 MeterPlot

### 30.19.1 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.

Three ranges on the dial provide some context for the value: the *normal range*, the *warning range* and the *critical range*.

### 30.19.2 Constructors

To create a new `MeterPlot`:

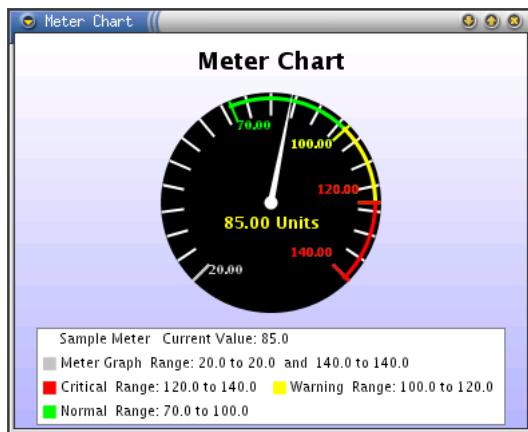
```
public MeterPlot(MeterDataset dataset);
```

Creates a dial with default settings, using the supplied dataset.

If you want to have more control over the appearance of the dial:

```
public MeterPlot(MeterDataset dataset, Insets insets, Paint backgroundPaint,
Image backgroundImage, float backgroundAlpha, Stroke outlineStroke, Paint
outlinePaint, float foregroundAlpha, int tickLabelType, Font tickLabelFont);
```

Creates a dial with the supplied settings and dataset.



*Figure 30.5: A meter chart*

### 30.19.3 Methods

A needle is used to indicate the current value on the dial. To change the color of the needle:

```
public void setNeedlePaint(Paint paint);
Sets the color of the needle on the dial. The default is Color.green. If you
pass in null to this method, the needle color reverts to the default.
```

The current value is also displayed (near the center of the dial) in text format. To change the font used to display the current value:

```
public void setValueFont(Font font);
Sets the font used to display the current value.
```

To change the color used to display the current value:

```
public void setValuePaint(Paint paint);
Sets the paint used to display the current value.
```

To change the background color of the dial:

```
public void setDialBackgroundPaint(Paint paint);
Sets the color of the dial background. The default is Color.black. If you
set this to null, no background is painted.
```

By default, the needle on the dial is free to rotate through 270 degrees (centered at 12 o'clock). To change this, use this method:

```
public void setMeterAngle(int angle);
Sets the range within which the dial's needle can move.
```

Related to the above is the shape of the dial: circular (the default), pie or chord:

```
public void setDialType(int type);
Sets the shape of the dial. The default is DIALTYPE_CIRCLE. The other
options are DIALTYPE_PIE and DIALTYPE_CHORD.
```

The three context ranges are drawn as color highlights near the outer edge of the dial. To change the highlight color of the normal range:

```
public void setNormalPaint(Paint paint);
```

Sets the color of the normal range. The default is `Color.green`. If you pass in `null` to this method, the color reverts to the default.

To change the highlight color of the warning range:

```
public void setWarningPaint(Paint paint);
```

Sets the color of the warning range. The default is `Color.yellow`. If you pass in `null` to this method, the color reverts to the default.

To change the highlight color of the critical range:

```
public void setCriticalPaint(Paint paint);
```

Sets the color of the critical range. The default is `Color.red`. If you pass in `null` to this method, the color reverts to the default.

To control whether or not labels are displayed for the values in the normal, warning, critical and overall ranges:

```
public void setTickLabelType(int type);
```

Controls whether or not tick labels are displayed. The `type` should be one of: `NO_LABELS` and `VALUE_LABELS`.

If tick labels are displayed, the font can be set using:

```
public void setTickLabelFont(Font font);
```

Sets the font used to display tick labels (if they are visible).

### 30.19.4 Notes

This chart type was contributed by Hari.

The `MeterPlotDemo` class in the `org.jfree.chart.demo` package provides a working example of this class.

In the current version, a fixed number of ticks (20) are drawn for the dial range, irrespective of the maximum and minimum data values. The tick generation will be enhanced in a future release.

#### See Also

[MeterDataset](#), [MeterLegend](#).

## 30.20 MultiplePiePlot

### 30.20.1 Overview

A specialised plot that displays data from a `CategoryDataset` in the form of multiple pie charts. Figure 30.6 shows an example.

### 30.20.2 Notes

Some points to note:

- a demo application (`MultiplePieChartDemo1.java`) is included in the JFreeChart Premium Demo distribution.
- the `createMultiplePieChart()` and `createMultiplePieChart3D()` methods in the `ChartFactory` class that create charts using this plot.

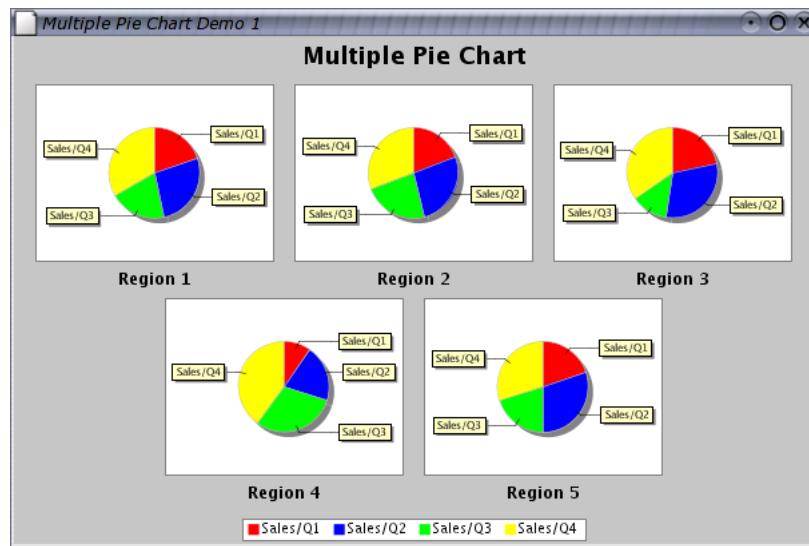


Figure 30.6: A multiple pie chart

## 30.21 PieLabelDistributor

### 30.21.1 Overview

The [PiePlot](#) class uses this class to arrange section labels so that they do not overlap one another.

## 30.22 PieLabelRecord

### 30.22.1 Overview

A temporary holder of information about the label for one section of a [PiePlot](#).

## 30.23 PiePlot

### 30.23.1 Overview

The [PiePlot](#) class draws pie charts using data obtained through the [PieDataset](#) interface. A sample chart is shown in figure 30.7. A related class, [PiePlot3D](#), draws pie charts with a 3D effect.

### 30.23.2 Constructors

To construct a pie plot:

```
public PiePlot(PieDataset dataset);
Creates a pie plot for the given dataset. All plot attributes are initialised
with default values—these can be changed at any time.
```

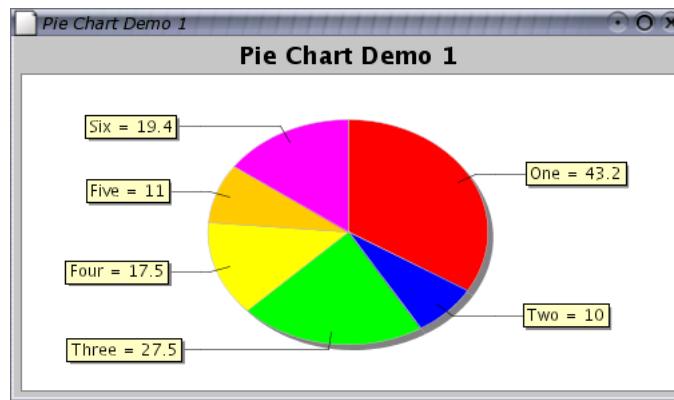


Figure 30.7: A sample pie chart

### 30.23.3 Attributes

The attributes maintained by the `PiePlot` class, which are in addition to those inherited from the `Plot` class, are listed in table 30.5.

The following default values are used where necessary:

Name:	Value:
DEFAULT_INTERIOR_GAP	0.25 (25 percent)
DEFAULT_START_ANGLE	90.0
DEFAULT_LABEL_FONT	<code>new Font("SansSerif", Font.PLAIN, 10);</code>
DEFAULT_LABEL_PAINT	<code>Color.black;</code>
DEFAULT_LABEL_BACKGROUND_PAINT	<code>new Color(255, 255, 192);</code>
DEFAULT_LABEL_GAP	0.10 (10 percent)

<b>Attribute:</b>	<b>Description:</b>
<i>interiorGap</i>	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.
<i>circular</i>	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.
<i>startAngle</i>	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).
<i>direction</i>	Pie sections can be ordered in a clockwise ( <code>Rotation.CLOCKWISE</code> ) or anticlockwise ( <code>Rotation.ANTI_CLOCKWISE</code> ) direction.
<i>sectionPaint</i>	The paint used for all sections (usually <code>null</code> ).
<i>sectionPaintList</i>	The paint used for each section, unless overridden by <i>sectionPaint</i> .
<i>baseSectionPaint</i>	The default paint, used when no other setting is specified.
<i>sectionOutlinePaint</i>	The outline paint used for all sections (usually <code>null</code> ).
<i>sectionOutlinePaintList</i>	The outline paint used for each section.
<i>baseSectionOutlinePaint</i>	The default outline paint, used when no other setting is specified.
<i>sectionOutlineStroke</i>	The outline stroke used for all sections (usually <code>null</code> ).
<i>sectionOutlineStrokeList</i>	The outline stroke used for each section.
<i>baseSectionOutlineStroke</i>	The default outline stroke, used when no other setting is specified.
<i>shadowPaint</i>	The shadow paint.
<i>shadowXOffset</i>	The x-offset for the shadow effect.
<i>shadowYOffset</i>	The y-offset for the shadow effect.
<i>explodePercentages</i>	The amount (percentage) to "explode" each pie section.
<i>labelGenerator</i>	The section label generator, an instance of <code>PieSectionLabelGenerator</code> .
<i>labelFont</i>	The font for the section labels.
<i>labelPaint</i>	The color for the section labels.
<i>labelBackgroundPaint</i>	The background color for the section labels.
<i>maximumLabelWidth</i>	The maximum label width as a percentage of the plot width.
<i>labelGap</i>	The gap for the section labels.
<i>labelLinkMargin</i>	The label link margin.
<i>labelLinkPaint</i>	The <code>Paint</code> used for the lines that connect the pie sections with their corresponding labels.
<i>labelLinkStroke</i>	The <code>Stroke</code> used for the lines that connect the pie sections to their corresponding labels.
<i>toolTipGenerator</i>	A plug-in tool tip generator.
<i>urlGenerator</i>	A plug-in URL generator (for image map generation).
<i>pieIndex</i>	The index for this plot (only used by the <code>MultiplePiePlot</code> class).

Table 30.5: Attributes for the `PiePlot` class

### 30.23.4 Methods

To replace the dataset being used by the plot:

```
public void setDataset(PieDataset dataset);
    Replaces the dataset being used by the plot (this triggers a DatasetChangeEvent).
```

To control whether the pie chart is circular or elliptical:

```
public void setCircular(boolean flag);
    Sets a flag that controls whether the pie chart is circular or elliptical in shape.
```

To control the position of the first section in the chart:

```
public void setStartAngle(double angle);
    Defines the angle (in degrees) at which the first 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.
```

To control the direction (clockwise or anticlockwise) of the sections in the pie chart:

```
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 void setInteriorGapPercent(double percent);
    Sets the amount of space inside the plot area.
```

A pie plot is drawn with this method:

```
public void draw(Graphics2D g2, Rectangle2D drawArea,
ChartRenderingInfo info);
    Draws the pie plot within the specified drawing area. Typically, this method will be called for you by the JFreeChart class.
```

The `info` parameter is optional. If you pass in an instance of `ChartRenderingInfo`, it will be populated with information about the chart (for example, chart dimensions and tool tip information).

If you are displaying your pie chart in a `ChartPanel` and you want to customise the tooltip text, you can register your own tool tip generator with the plot:

```
public void setToolTipGenerator(PieToolTipGenerator generator);
    Registers a tool tip generator with the pie plot. You can set this to null if you do not require tooltips.
```

### 30.23.5 Section Colors

The colors used to fill the sections in a pie chart are fully customisable. To set the color used to fill a particular section:

```
public void setSectionPaint(int section, Paint paint);
    Sets the paint used to fill a particular section in the chart and sends a PlotChangeEvent to all registered listeners.
```

In a similar way, you can control the paint and stroke used to outline individual sections in the chart. To set the outline paint:

```
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.
```

To set the outline stroke:

```
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.
```

### 30.23.6 Shadow Effect

The pie plot will draw a “shadow” effect. To set the paint used to draw the shadow:

```
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.
```

To set the x-offset for the shadow effect:

```
public void setShadowXOffset(double offset);
Sets the x-offset (in Java2D units) for the shadow effect.
```

To set the y-offset for the shadow effect:

```
public void setShadowYOffset(double offset);
Sets the y-offset (in Java2D units) for the shadow effect.
```

### 30.23.7 Exploded Sections

It is possible to “explode” sections of the pie chart. The `PieChartDemo2` application (included in the JFreeChart Premium Demo distribution) provides a demo.

### 30.23.8 Section Labels

Section labels are now generated by a plugin object that is an instance of any class that implements the `PieSectionLabelGenerator` interface:

```
public PieSectionLabelGenerator getLabelGenerator();
Returns the section label generator for the plot (possibly null).

public void setLabelGenerator(PieSectionLabelGenerator generator);
Sets the label generator for the plot and sends a PlotChangeEvent to all reg-
istered 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 StandardPieItemLabelGenerator(
    "{0} = {2}", new DecimalFormat("0"), new DecimalFormat("0.00%")
);
plot.setLabelGenerator(generator);
```

To set the color of the lines connecting the pie sections to their corresponding labels:

```
public void setLabelLinkPaint(Paint paint);
Sets the Paint used for the lines connecting the pie sections to their cor-
responding labels and sends a PlotChangeEvent to all registered listeners.
```

To set the line style for the linking lines:

```
public void setLabelLinkStroke(Stroke stroke);
Sets the Stroke used for the lines connecting the pie sections to their cor-
responding labels and sends a PlotChangeEvent to all registered listeners.
```

At the current time, there is no facility to hide the linking lines.

### 30.23.9 Notes

Some points to note:

- 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.

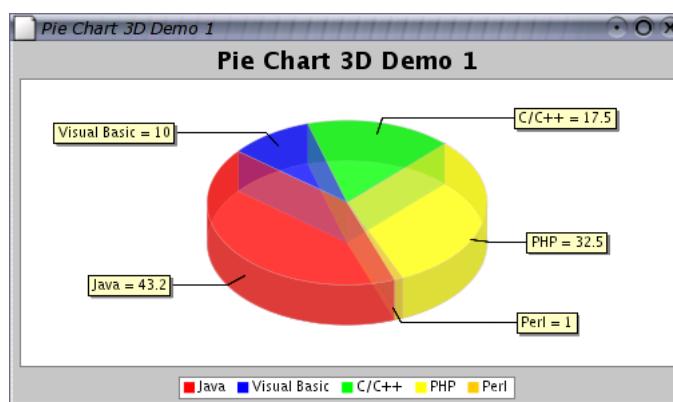
#### See Also

[PieDataset](#), [PieSectionLabelGenerator](#), [PieToolTipGenerator](#), [Plot](#).

## 30.24 PiePlot3D

### 30.24.1 Overview

An extension of the [PiePlot](#) class that draws pie charts with a 3D effect.



### 30.24.2 Notes

This class does not yet support the “exploded” sections that can be displayed by the regular pie charts.

## 30.25 PiePlotState

### 30.25.1 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).

## 30.26 Plot

### 30.26.1 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 30.8 illustrates the plot class hierarchy.

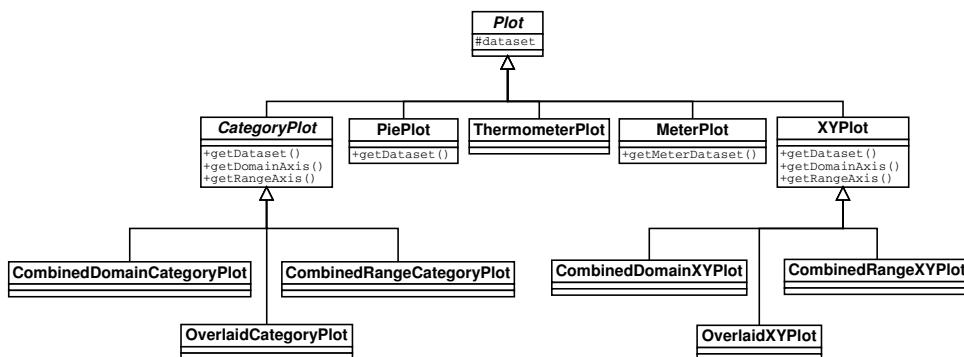


Figure 30.8: Plot classes

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.

### 30.26.2 Constructors

This class is abstract, so the constructors are `protected`. You cannot create an instance of this class directly, you must use a subclass.

### 30.26.3 Attributes

This class maintains the following attributes:

Attribute:	Description:
<code>parent</code>	The parent plot (possibly <code>null</code> ).
<code>datasetGroup</code>	The dataset group (not used).
<code>insets</code>	The amount of space to leave around the outside of the plot.
<code>outlineStroke</code>	The <code>Stroke</code> used to draw an outline around the plot area.
<code>outlinePaint</code>	The <code>Paint</code> used to draw an outline around the plot area.
<code>backgroundPaint</code>	The <code>Paint</code> used to draw the background of the plot area.
<code>backgroundImage</code>	An image that is displayed in the background of the plot (can be <code>null</code> ).
<code>backgroundImageAlignment</code>	The image alignment.
<code>backgroundAlpha</code>	The alpha transparency value used when coloring the plot's background, and also when drawing the background image (if there is one).
<code>foregroundAlpha</code>	The alpha transparency used to draw items in the plot's foreground.
<code>noDataMessage</code>	A string that is displayed by some plots when there is no data to display.
<code>noDataMessageFont</code>	The <code>Font</code> used to display the “no data” message.
<code>noDataMessagePaint</code>	The <code>Paint</code> used to display the “no data” message.
<code>drawingSupplier</code>	The drawing supplier (provides default colors and line strokes).
<code>dataAreaRatio</code>	The aspect ratio for the data area.
<code>datasetGroup</code>	The dataset group (to be used for synchronising dataset access).

Table 30.6: Attributes for the `Plot` class

All subclasses will inherit these core attributes.

### 30.26.4 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 its 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.

### 30.26.5 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.

By default, the background area for most plot's in JFreeChart is white. You can change this with the following method:

```
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.

You can also add an image to the background area.

```
public void setBackgroundImage(Image image);
```

Sets the background image for the plot area and sends a `PlotChangeEvent` to all registered listeners. If `image` is `null`, no background image will be drawn.

When using the preceding method, 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 19.4 for more information.

By default, the background image will be stretched to fill the plot area. 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).

Both the background paint and the background image can be drawn using an alpha-transparency, you can set this as follows:

```
plot.setBackgroundAlpha(0.6f);
```

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).

### 30.26.6 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.

### 30.26.7 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 plotArea,
    ChartRenderingInfo 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`.

### 30.26.8 Notes

Refer to specific subclasses for information about setting the colors, shapes and line styles for data drawn by the plot.

## 30.27 PlotOrientation

### 30.27.1 Overview

Used to represent the orientation of a plot (in particular, the `CategoryPlot` and `XYPlot` classes). There are two values, as listed in table 30.7.

Class:	Description:
<code>PlotOrientation.HORIZONTAL</code>	A "horizontal" orientation.
<code>PlotOrientation.VERTICAL</code>	A "vertical" orientation.

Table 30.7: 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.

### 30.27.2 Notes

For interesting effects, in addition to changing the orientation of a chart you can:

- change the location of the chart's axes;
- invert the scale of the axes.

## 30.28 PlotRenderingInfo

### 30.28.1 Overview

This class is used to record information about the individual elements in a single rendering of a plot. See also the `ChartRenderingInfo` class.

## 30.29 PlotState

### 30.29.1 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).

## 30.30 PolarPlot

### 30.30.1 Overview

A plot that is used to display data from an [XYDataset](#) using polar coordinates—see figure 30.9 for an example.

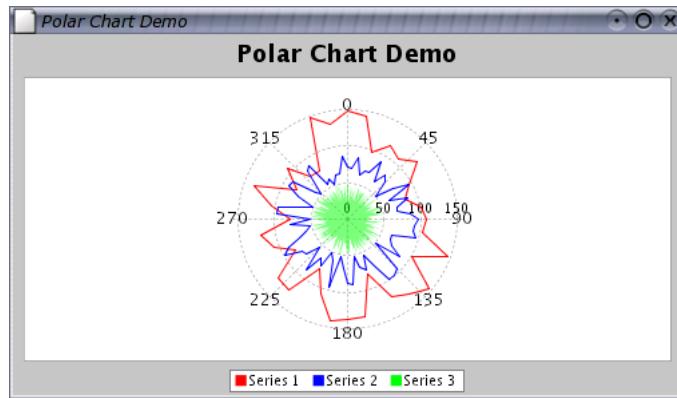


Figure 30.9: A polar chart

The items in the plot are drawn by a [PolarItemRenderer](#).

### 30.30.2 Usage

There is a demo application (`PolarChartDemo1.java`) included in the JFreeChart Premium Demo distribution that illustrates the use of this class.

### 30.30.3 Notes

Some points to note:

- instances of this class are cloneable and serializable.

## 30.31 RingPlot

### 30.31.1 Overview

A *ring plot* is an adaptation of a pie plot, where a hole is left in the middle of the “pie”.

### 30.31.2 Constructor

```
public RingPlot(PieDataset dataset);  
Creates a new instance.
```

### 30.31.3 Methods

```
public boolean getSeparatorsVisible();  
  
public void setSeparatorsVisible(boolean visible);  
  
public Stroke getSeparatorStroke();  
  
public void setSeparatorStroke(Stroke stroke);  
  
public Paint getSeparatorPaint();  
  
public void setSeparatorPaint(Paint paint);  
  
public double getInnerSeparatorExtension();  
  
public void setInnerSeparatorExtension(double percent);  
  
public double getOuterSeparatorExtension();  
  
public void setOuterSeparatorExtension(double percent);  
  
public boolean equals(Object obj);
```

## 30.32 ThermometerPlot

### 30.32.1 Overview

A plot that displays a single value in a thermometer-style representation.

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 color 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.

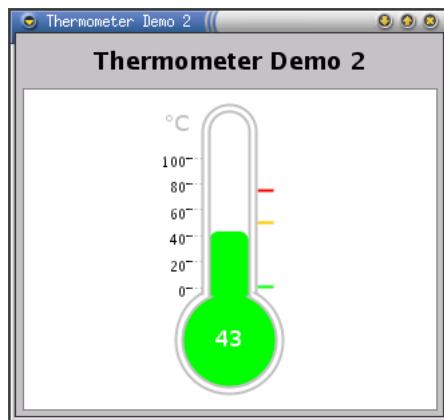


Figure 30.10: A thermometer chart

### 30.32.2 Constructors

To create a new ThermometerPlot:

```
public ThermometerPlot(ValueDataset dataset);
Creates a thermometer with default settings, using the supplied dataset.
```

### 30.32.3 Methods

The current value can be displayed as text in the thermometer bulb or to the right of the thermometer. To set the position:

```
public void setValueLocation(int location);
Sets the position of the value label. Use one of the constants: NONE, RIGHT
or BULB.
```

The font for the value label can be set as follows:

```
public void setValueFont(Font font);
Sets the font used to display the current value.
```

Similarly, the paint for the value label can be set as follows:

```
public void setValuePaint(Paint paint);
Sets the paint used to display the current value.
```

You can set a formatter for the value label:

```
public void setValueFormatter(NumberFormat formatter);
Sets the formatter for the value label.
```

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".
```

You can specify the bounds for any of the three sub-ranges:

```
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 sub-range:

```
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. You can set a flag that controls whether or not this automatic adjustment happens:

```
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`.

To set the default color of the “mercury” in the thermometer:

```
public void setMercuryPaint(Paint paint);
```

Sets the default color of the mercury in the thermometer.

To set the color of the mercury for each sub-range:

```
public void setSubrangePaint(int range, Paint paint);
```

Sets the paint used for the mercury when the data value is within the specified sub-range. Use one of the constants `NORMAL`, `WARNING` or `CRITICAL` to indicate the sub-range.

The sub-range mercury colors are only used if the `useSubrangePaint` flag is set to `true` (the default):

```
public void setUseSubrangePaint(boolean flag);
```

Sets the flag that controls whether or not the sub-range colors are used for the mercury in the thermometer.

To show grid lines within the thermometer stem:

```
public void setShowValueLines(boolean flag);
```

Sets a flag that controls whether or not grid lines are displayed inside the thermometer stem.

To control the color of the thermometer outline:

```
public void setThermometerPaint(Paint paint);
```

Sets the paint used to draw the outline of the thermometer.

To control the pen used to draw the thermometer outline:

```
public void setThermometerStroke(Stroke stroke);
```

Sets the stroke used to draw the outline of the thermometer.

You can control the amount of white space at the top and bottom of the thermometer:

```
public void setPadding(RectangleInsets padding);
```

Sets the padding around the thermometer.

### 30.32.4 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.
- various dimensions for the thermometer (for example, the bulb radius) are hard-coded constants in the current implementation. A useful enhancement would be to replace these constants with attributes that could be modified via methods in the `ThermometerPlot` class.
- the `ThermometerDemo` class in the `org.jfree.chart.demo` package provides a working example of this class.

## 30.33 ValueAxisPlot

### 30.33.1 Overview

This interface allows the `ChartPanel` class to communicate with different plot types, mostly for the purpose of executing zooming operations.

### 30.33.2 Methods

This interface defines the following methods:

```
public Range getDataRange(ValueAxis axis);
Returns the range that is required to display all data values that are plotted against the specified axis.

public void zoomHorizontalAxes(double factor);
Zooms in or out on the plot's horizontal axes.

public void zoomHorizontalAxes(double lowerPercent, double upperPercent);
Zooms in on the plot's horizontal axes.

public void zoomVerticalAxes(double factor);
Zooms in or out on the plot's vertical axes.

public void zoomVerticalAxes(double lowerPercent, double upperPercent);
Zooms in on the plot's vertical axes.
```

## 30.34 ValueMarker

### 30.34.1 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.

### 30.34.2 Usage

There is a demo application (`MarkerDemo1.java`) included in the JFreeChart Premium Demo distribution that illustrates the use of this class.

### 30.34.3 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;
- this class is `Cloneable` and `Serializable`.

## 30.35 WaferMapPlot

### 30.35.1 Overview

To be documented.

## 30.36 XYPlot

### 30.36.1 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.

### 30.36.2 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:



*Figure 30.11: The plot regions*

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).

### 30.36.3 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(int index);  
Returns the dataset at the specified index (possibly null).
```

```
public void setDataset(int index, XYDataset 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 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`;

### 30.36.4 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.

### 30.36.5 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 12 for details.

### 30.36.6 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.

### 30.36.7 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.
```

### 30.36.8 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.
```

### 30.36.9 Gridlines

By default, the plot will draw *gridlines* in the background of the plot area. Vertical lines are drawn for each tick mark on the domain axis, and horizontal lines are drawn for each tick mark on the range axis.

You can customise both the color (`Paint`) and line-style (`Stroke`) of the gridlines. For example, to change the grid lines to solid black lines:

```
XYPlot plot = myChart.getXYPlot();
plot.setDomainGridStroke(new BasicStroke(0.5f));
plot.setDomainGridPaint(Color.black);
plot.setRangeGridStroke(new BasicStroke(0.5f));
plot.setRangeGridPaint(Color.black);
```

If you prefer to have no gridlines at all, you can turn them off:

```
XYPlot plot = myChart.getXYPlot();
plot.setDomainGridVisible(false);
plot.setRangeGridVisible(false);
```

Note that the settings for the domain grid lines and the range grid lines are independent of one another.

### 30.36.10 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.
```

### 30.36.11 Annotations

You can add annotations to a chart 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);
```

To clear all annotations:

```
plot.clearAnnotations();
```

### 30.36.12 Constructors

To create a plot with a specific renderer:

```
public XYPlot(XYDataset data, ValueAxis domainAxis, ValueAxis rangeAxis,  
XYItemRenderer renderer);  
Creates an XY plot with a specific renderer.
```

### 30.36.13 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`.

# Chapter 31

## Package: `org.jfree.chart.renderer`

### 31.1 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.

### 31.2 AbstractRenderer

#### 31.2.1 Overview

An abstract class that provides common services for renderer implementations. This base class is extended by both the `AbstractCategoryItemRenderer` class and the `AbstractXYItemRenderer` class.

#### 31.2.2 Attributes

The attributes maintained by the `AbstractRenderer` class are listed in Table 31.1.

#### 31.2.3 Setting Series Colors

Renderers are responsible for drawing the data items within a plot, so this class provides attributes for controlling the colors that will be used. Colors 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 colors (using the `DrawingSupplier` interface). However, you can manually update the paint list at any time. First, you need to obtain a reference to the

Attribute:	Description:
<i>paint</i>	The paint that applies to ALL series ( <code>null</code> permitted).
<i>paintList</i>	A list of paints that apply to individual series (only referenced if <i>paint</i> is <code>null</code> ).
<i>basePaint</i>	The paint that is used if there is no other setting.
<i>outlinePaint</i>	The outline paint that applies to ALL series ( <code>null</code> permitted).
<i>outlinePaintList</i>	A list of outline paints that apply to individual series (only referenced if <i>outlinePaint</i> is <code>null</code> ).
<i>baseOutlinePaint</i>	The outline paint that is used if there is no other setting.
<i>stroke</i>	The stroke that applies to ALL series ( <code>null</code> permitted).
<i>strokeList</i>	A list of stroke objects that apply to individual series (only referenced if <i>stroke</i> is <code>null</code> ).
<i>baseStroke</i>	The stroke that is used if there is no other setting.
<i>outlineStroke</i>	The outline stroke that applies to ALL series ( <code>null</code> permitted).
<i>outlineStrokeList</i>	A list of outline strokes that apply to individual series (only referenced if <i>outlineStroke</i> is <code>null</code> ).
<i>baseOutlineStroke</i>	The outline stroke that is used if there is no other setting.
<i>shape</i>	The shape that applies to ALL series ( <code>null</code> permitted).
<i>shapeList</i>	A list of shapes that apply to individual series (only referenced if <i>shape</i> is <code>null</code> ).
<i>baseShape</i>	The shape that is used if there is no other setting.

Table 31.1: Attributes for the `AbstractRenderer` class

renderer(s) (note that many charts do not use a 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);
```

### 31.2.4 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.

Attribute:	Description:
<i>itemLabelsVisible</i>	The flag that applies to ALL series ( <code>null</code> permitted).
<i>itemLabelsVisibleList</i>	A list of flags that apply to individual series (only referenced if <i>itemLabelsVisible</i> is <code>null</code> ).
<i>baseItemLabelsVisible</i>	The flag that is used if there is no other setting.
<i>itemLabelFont</i>	The font that applies to ALL series ( <code>null</code> permitted).
<i>itemLabelFontList</i>	A list of fonts that apply to individual series (only referenced if <i>itemLabelFont</i> is <code>null</code> ).
<i>baseItemLabelFont</i>	The font that is used if there is no other setting.
<i>itemLabelPaint</i>	The paint that applies to ALL series ( <code>null</code> permitted).
<i>itemLabelPaintList</i>	A list of paints that apply to individual series (only referenced if <i>itemLabelPaint</i> is <code>null</code> ).
<i>baseItemLabelPaint</i>	The font that is used if there is no other setting.
<i>itemLabelAnchor</i>	The anchor that applies to ALL series ( <code>null</code> permitted).
<i>itemLabelAnchorList</i>	A list of anchors that apply to individual series (only referenced if <i>itemLabelAnchor</i> is <code>null</code> ).
<i>baseItemLabelAnchor</i>	The anchor that is used if there is no other setting.
<i>itemLabelTextAnchor</i>	The text anchor that applies to ALL series ( <code>null</code> permitted).
<i>itemLabelTextAnchorList</i>	A list of text anchors that apply to individual series (only referenced if <i>itemLabelTextAnchor</i> is <code>null</code> ).
<i>baseItemLabelTextAnchor</i>	The text anchor that is used if there is no other setting.
<i>itemLabelRotationAnchor</i>	The rotation anchor that applies to ALL series ( <code>null</code> permitted).
<i>itemLabelRotationAnchorList</i>	A list of rotation anchors that apply to individual series (only referenced if <i>itemLabelRotationAnchor</i> is <code>null</code> ).
<i>baseItemLabelRotationAnchor</i>	The anchor that is used if there is no other setting.
<i>itemLabelAngle</i>	The angle that applies to ALL series ( <code>null</code> permitted).
<i>itemLabelAngleList</i>	A list of angles that apply to individual series (only referenced if <i>itemLabelAngle</i> is <code>null</code> ).
<i>baseItemLabelAngle</i>	The angle that is used if there is no other setting.

Table 31.2: Attributes for the `AbstractRenderer` class

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.setDefaultShapeFilled(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);
}

```

### 31.2.5 Equals, Cloning and Serialization

This class overrides the equals(...) method. *TO DO: review equality tests for Paint and Stroke objects.*

## 31.3 AreaRendererEndType

### 31.3.1 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 31.3.

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 31.3: *AreaRendererEndType tokens*

### 31.3.2 Usage

The `AreaRenderer` class has a method named `setEndType()` that accepts the tokens defined by this class.

## 31.4 DefaultPolarItemRenderer

### 31.4.1 Overview

A default renderer for use by the `PolarPlot` class (implements the `PolarItemRenderer` interface).

## 31.5 NotOutlierException

### 31.5.1 Overview

Placeholder.

## 31.6 Outlier

### 31.6.1 Overview

Represents an outlier in a box-and-whisker plot.

## 31.7 OutlierList

### 31.7.1 Overview

Represents a collection of outliers for a single item in a box-and-whisker plot.

## 31.8 OutlierListCollection

### 31.8.1 Overview

Represents a collection of outlier lists for a box-and-whisker plot.

## 31.9 PolarItemRenderer

### 31.9.1 Overview

A renderer that is used by the [PolarPlot](#) class. The [DefaultPolarItemRenderer](#) class provides an implementation of this interface.

### 31.9.2 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.
```

### 31.9.3 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:

```
public void drawSeries();  
Renders the specified series.
```

## 31.10 RendererState

### 31.10.1 Overview

To be documented.

## 31.11 WaferMapRenderer

### 31.11.1 Overview

To be documented.

# Chapter 32

## Package: org.jfree.chart.renderer.category

### 32.1 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.

### 32.2 AbstractCategoryItemRenderer

#### 32.2.1 Overview

A base class that can be used to implement a new [CategoryItemRenderer](#).

#### 32.2.2 Constructors

The default constructor creates a renderer with no tooltip generator and no URL generator. The constructor is [protected](#).

#### 32.2.3 Attributes

The attributes maintained by this class are listed in Table 32.1.

#### 32.2.4 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).
```

## CHAPTER 32. PACKAGE: ORG.JFREE.CHART.RENDERER.CATEGORY306

Attribute:	Description:
<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>labelGenerator</i>	The <code>CategoryLabelGenerator</code> that generates item labels for ALL series (can be null).
<i>labelGeneratorList</i>	A list of <code>CategoryLabelGenerator</code> objects used to create item labels for individual series. If null, the <code>baseLabelGenerator</code> is used instead.
<i>baseLabelGenerator</i>	The base <code>CategoryLabelGenerator</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.

Table 32.1: Attributes for the `AbstractCategoryItemRenderer` class

The number of rows and columns in the dataset (a `CategoryDataset`) is cached by the renderer in the `initialise()` method.

To get the renderer type:

```
public RangeType getRangeType();
Returns the range type for the renderer (STANDARD or STACKED).
```

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.
```

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.
```

To draw a range marker:

```
public void drawRangeMarker(Graphics2D g2, CategoryPlot plot,
    ValueAxis axis, Marker marker, Rectangle2D dataArea);
Draws a range marker.
```

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 `CategoryLabelGenerator` for a data item:

```
public CategoryLabelGenerator getLabelGenerator(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 `CategoryLabelGenerator` for a series:

```
public CategoryLabelGenerator getSeriesLabelGenerator(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.
```

### 32.2.5 Notes

If you are implementing your own renderer, you do not have to use this base class, but it does save you some work.

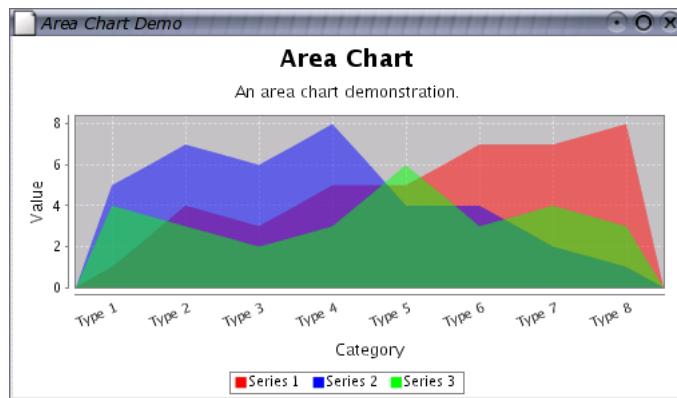


Figure 32.1: A chart that uses `AreaRenderer`

## 32.3 AreaRenderer

### 32.3.1 Overview

A *category item renderer* that represents each item in a `CategoryDataset` using a polygon that fills the area between the x-axis and the data point—an example is shown in figure 32.1.

This renderer is designed for use with the `CategoryPlot` class.

### 32.3.2 Methods

To control how the end points of the area chart are represented:

```
public void setEndType(AreaRendererEndType type);
Sets the attribute that controls how the end points are drawn on the area
chart.
```

### 32.3.3 Notes

Some notes:

- the `createAreaChart()` method in the `ChartFactory` class will create a default chart that uses this renderer.
- this class extends `AbstractCategoryItemRenderer`.

### See Also

[XYAreaRenderer](#).

## 32.4 BarRenderer

### 32.4.1 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 32.2) or a horizontal orientation (see figure 32.3).

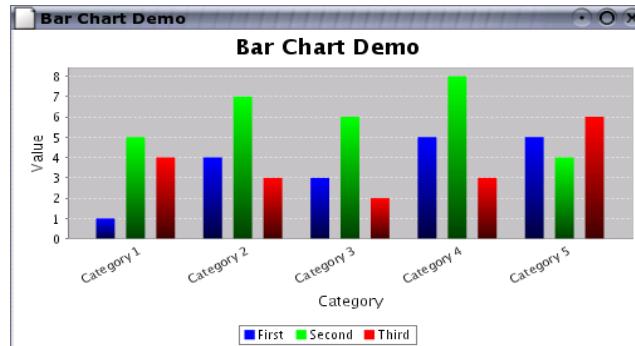


Figure 32.2: A vertical bar chart

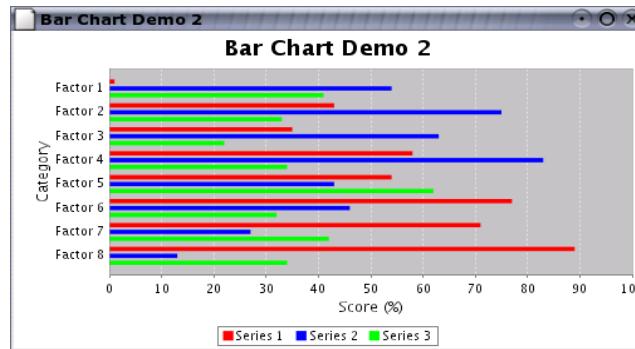


Figure 32.3: A horizontal bar chart

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 implements the `CategoryItemRenderer` interface, and is an extension of the `AbstractCategoryItemRenderer` base class.

### 32.4.2 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`.

### 32.4.3 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 23.8.1 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 *maxBarWidth* attribute:

```
public double getMaxBarWidth();
```

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 setMaxBarWidth(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.

### 32.4.4 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.

```
public void setDrawBarOutline(boolean draw);
```

Sets a 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.

### 32.4.5 Gradient Paint Support

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 Premium Demo distribution, provides an example of the use of this attribute.

### 32.4.6 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 32.4. Note that the numbers correspond (roughly) to the position of the hours on a clock face.

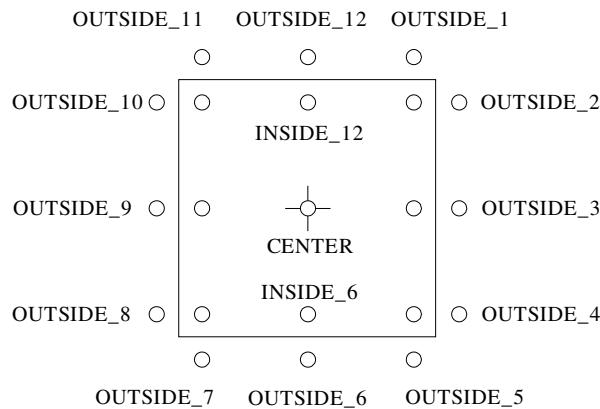


Figure 32.4: Item Label Anchors for Bars

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.
```

### 32.4.7 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 ren-  
derer 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.
```

### 32.4.8 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.

### 32.4.9 Notes

Some points to note:

- the [ChartFactory](#) class uses this renderer when it constructs bar charts.
- the [BarChartDemo1.java](#) class, included in the JFreeChart Premium Demo distribution, is one example that uses this renderer.

#### See Also

[StackedBarRenderer](#), [BarRenderer3D](#), [StackedBarRenderer3D](#).

## 32.5 BarRenderer3D

### 32.5.1 Overview

A renderer that draws items from a [CategoryDataset](#) using bars with a 3D effect. Figure 32.5 shows the renderer being used with a plot that has a vertical orientation and figure 32.6 shows the renderer being used with a plot that has a horizontal orientation.

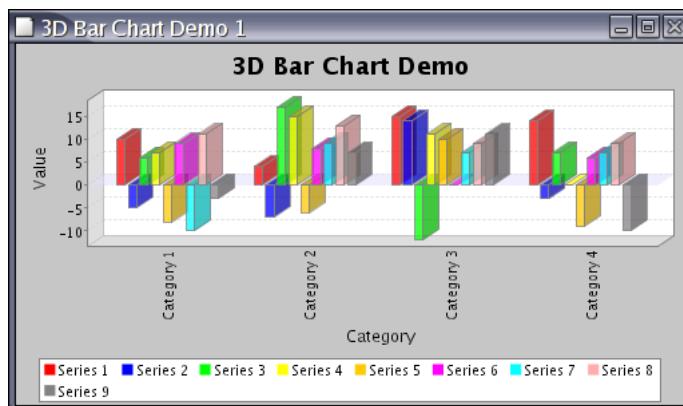


Figure 32.5: An example of the [BarRenderer3D](#) class at work

This renderer is designed for use with the [CategoryPlot](#) class.

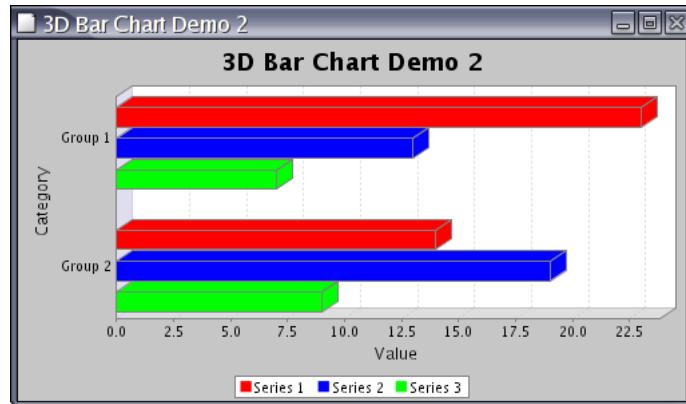


Figure 32.6: Another 3D bar chart

### 32.5.2 Notes

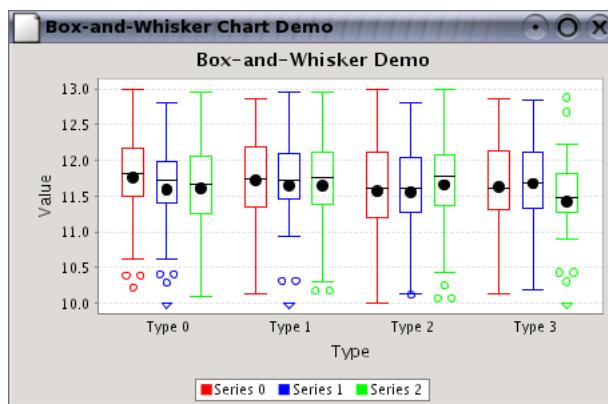
Some points to note:

- this class implements the [CategoryItemRenderer](#) interface.
- the [BarChart3DDemo1](#) and [BarChart3DDemo2](#) applications (included in the JFreeChart Premium Demo distribution) provide demonstrations of this renderer in use.

## 32.6 BoxAndWhiskerRenderer

### 32.6.1 Overview

A renderer that is used to create a box-and-whisker chart using data from a [BoxAndWhiskerCategoryDataset](#). A sample chart is shown in Figure 32.7

Figure 32.7: A chart generated with a [BoxAndWhiskerRenderer](#)

### 32.6.2 Constructors

To create a new renderer:

```
public BoxAndWhiskerRenderer();
Creates a new renderer.
```

### 32.6.3 Notes

Some points to note:

- there is a demo (`BoxAndWhiskerDemo.java`) included in the JFreeChart Premium Demo distribution.

#### See Also

[XYBoxAndWhiskerRenderer](#).

## 32.7 CategoryItemRenderer

### 32.7.1 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.

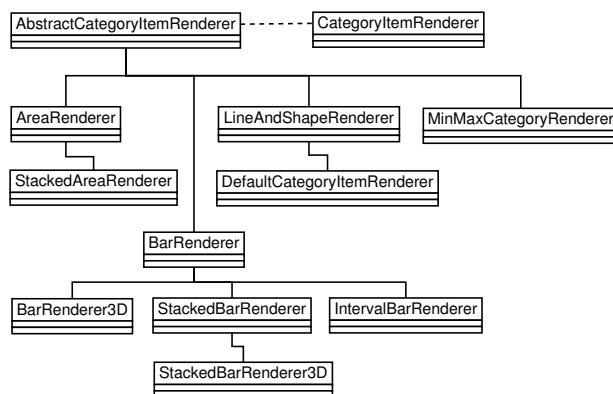


Figure 32.8: Category item renderers

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:

Class:	Description:
<code>AreaRenderer</code>	Used to create area charts.
<code>BarRenderer</code>	Represents data using bars (anchored at zero).
<code>BarRenderer3D</code>	Represents data using bars (anchored at zero) with a 3D effect.
<code>StackedBarRenderer</code>	Used to create a stacked bar charts.
<code>IntervalBarRenderer</code>	Draws intervals using bars. This renderer can be used to create simple Gantt charts.
<code>LineAndShapeRenderer</code>	Draws lines and/or shapes to represent data.

### 32.7.2 Methods

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 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.
```

### 32.7.3 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 CategoryLabelGenerator 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 that will be used for all data items in the chart:

```
public void setLabelGenerator(CategoryLabelGenerator generator);
Sets the 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.
```

To set a generator for a particular series:

```
public void setSeriesLabelGenerator(int series, CategoryLabelGenerator
generator);
Sets the item label generator for the specified series. If null, the baseIt-
emLabelGenerator will be used.
```

To make item labels visible for ALL series:

```
public void setItemLabelsVisible(boolean visible);
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);
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.
```

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);
Sets a flag that controls whether or not item labels are visible for the spec-
ified series. If this is set to null, the baseItemLabelsVisible flag determines
the visibility.
```

The position of the item labels is set using the following methods (one applies to positive data items and the other applies to negative data items):

```
public void setPositiveItemLabelPosition(ItemLabelPosition position);
Sets the position for labels for data items where the y-value is positive.

public void setNegativeItemLabelPosition(ItemLabelPosition position);
Sets the position for labels for data items where the y-value is negative.
```

### 32.7.4 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).
```

You can register a generator with the renderer using:

```
public void setToolTipGenerator(CategoryToolTipGenerator generator);
Sets the tool tip generator that will be used for ALL data items in the
chart, and sends a RendererChangeEvent to all registered listeners.
```

### 32.7.5 URL Generation

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 mechanism that is very similar to the tooltips mechanism.

*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`.*

You can associate a `CategoryURLGenerator` with the renderer using this method:

```
public void setItemURLGenerator(CategoryURLGenerator generator);
Sets the generator that will be used to generate URLs for items in ALL
series.
```

It is possible to specify a different URL generator for each series by first setting the generator in the previous method to `null` then using the following method to assign a generator to each series independently:

```
public void setSeriesItemURLGenerator(int series, CategoryURLGenerator
generator);
Sets the generator for the items in a particular series.
```

In most cases, a single generator for all series will suffice.

### 32.7.6 Notes

Some points to note:

- 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](#).

## 32.8 CategoryItemRendererState

### 32.8.1 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.

## 32.9 CategoryStepRenderer

### 32.9.1 Overview

A renderer that draws “steps” between each data value in a `CategoryPlot`.

### 32.9.2 Constructor

To create a new renderer:

```
public CategoryStepRenderer();
Creates a new renderer with stagger set to false.
```

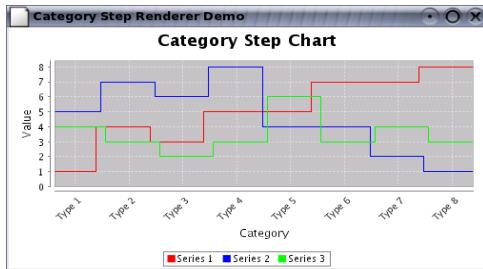


Figure 32.9: A chart using *CategoryStepRenderer*

```
public CategoryStepRenderer(boolean stagger);
```

Creates a new renderer. If `stagger` is `true`, the vertical steps for each series are offset slightly from one another.

### 32.9.3 Methods

To get/set the “stagger” flag:

```
public boolean getStagger();
```

Returns the flag that controls whether or not the “step” for each series is offset from the other series (to avoid the vertical lines overlapping). In the sample chart (see figure ??) this flag is set to `true`.

```
public void setStagger(boolean shouldStagger);
```

Sets the flag that controls whether or not the series are “staggered” and sends a `RenderChangeEvent` to all registered listeners.

## 32.10 DefaultCategoryItemRenderer

### 32.10.1 Overview

This class is an alias for the `LineAndShapeRenderer` class.

## 32.11 GanttRenderer

### 32.11.1 Overview

A renderer that is used to draw simple Gantt charts—an example is shown in figure 32.10.

The renderer is used with the `CategoryPlot` class and accesses data via the `GanttCategoryDataset` interface.

### 32.11.2 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:

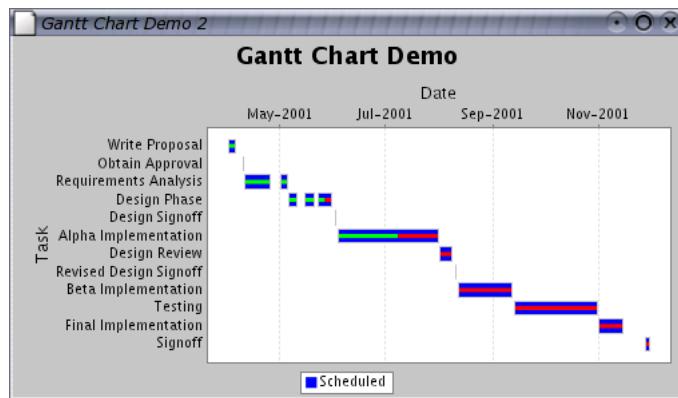


Figure 32.10: A Gantt chart

```
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 com-
pleted 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.

### 32.11.3 Notes

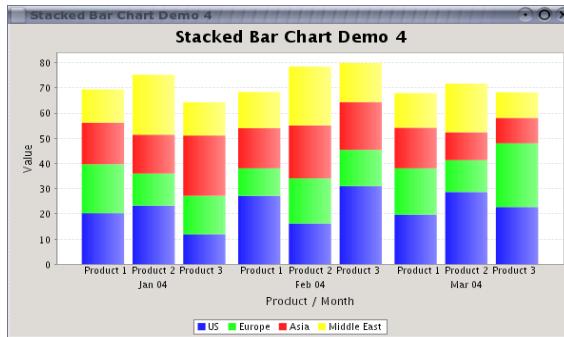
Some points to note:

- the `GanttDemo1.java` and `GanttDemo2.java` applications (included in the JFreeChart Premium Demo distribution) provide examples of this renderer being used.

## 32.12 GroupedStackedBarRenderer

### 32.12.1 Overview

This renderer is used to draw grouped and stacked bar charts using data from a `CategoryDataset` (see figure 32.11).



*Figure 32.11: A grouped and stacked bar chart*

This class extends the `StackedBarRenderer` class.

### 32.12.2 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.
```

### 32.12.3 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.

### 32.12.4 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.
```

### 32.12.5 Notes

Some points to note:

- there is a demo (`StackedBarChartDemo4.java`) included in the JFreeChart Premium Demo distribution.

## 32.13 IntervalBarRenderer

### 32.13.1 Overview

A renderer that draws bars to represent items from an [IntervalCategoryDataset](#).

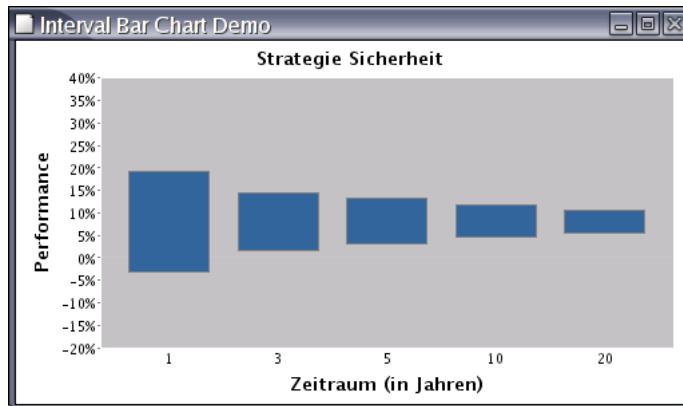


Figure 32.12: A chart that uses an *IntervalBarRenderer*

### 32.13.2 Notes

Some points to note:

- the [IntervalCategoryToolTipGenerator](#) interface can be used to generate tooltips with this renderer.

#### See Also

[GanttRenderer](#).

## 32.14 LayeredBarRenderer

### 32.14.1 Overview

A renderer that draws layered bars to represent items from an [CategoryDataset](#).

## 32.15 LevelRenderer

### 32.15.1 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.

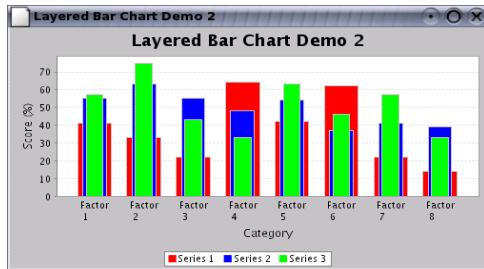


Figure 32.13: A chart that uses a `LayeredBarRenderer`

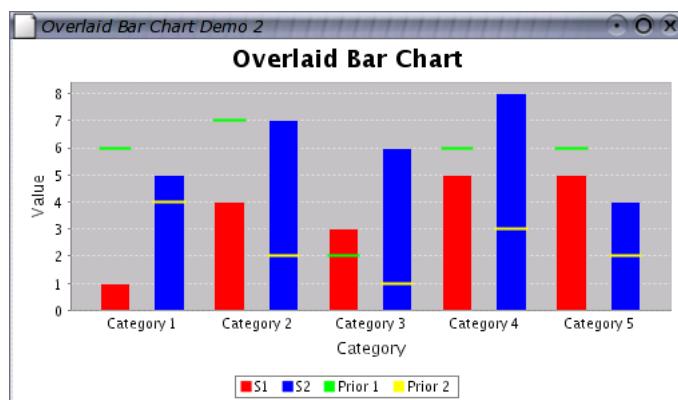


Figure 32.14: A chart that uses a `LevelRenderer`

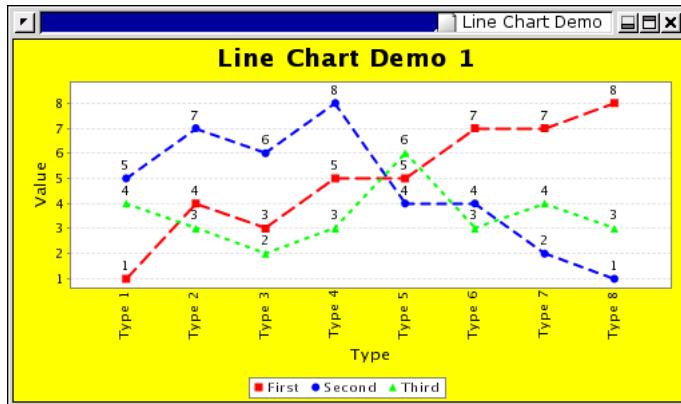
### 32.15.2 Notes

The `OverlaidBarChartDemo2` application (included in the JFreeChart Premium Demo distribution) provides a demo of this renderer.

## 32.16 LineAndShapeRenderer

### 32.16.1 Overview

A *line and shape* renderer displays items in a `CategoryDataset` by drawing a shape at each data point, or connecting data points with straight lines, or both.



This renderer is designed for use with the [CategoryPlot](#) class.

### 32.16.2 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.
```

The other constructor allows you to specify the type of renderer:

```
public LineAndShapeRenderer(int type);
Creates a new renderer of the specified type. Use one of the constants
defined by this class: SHAPES, LINES, or SHAPES_AND_LINES.
```

### 32.16.3 Methods

To control the drawing of lines between data points:

```
public void setDrawLines(boolean draw);
Sets a flag that controls whether or not lines are drawn between data
points. Notes that no line is drawn if a null data values is encountered.
```

To control the drawing of shapes at each data point:

```
public void setDrawShapes(boolean draw);
Sets the flag that controls whether or not shapes are drawn at each data
point.
```

If shapes are drawn at each data point, you can set a flag that controls whether or not the shapes are filled. The following two methods allow you to specify the setting for ALL series:

```
public void setShapesFilled(boolean filled);
Sets a flag that controls whether or not shapes are filled for ALL series.
```

```
public void setShapesFilled(Boolean filled);
As above, but using a Boolean object. This allows the flag to be set to
null, which means that the per series settings will apply.
```

This class implements the `drawCategoryItem()` method that is defined in the [CategoryItemRenderer](#) interface.

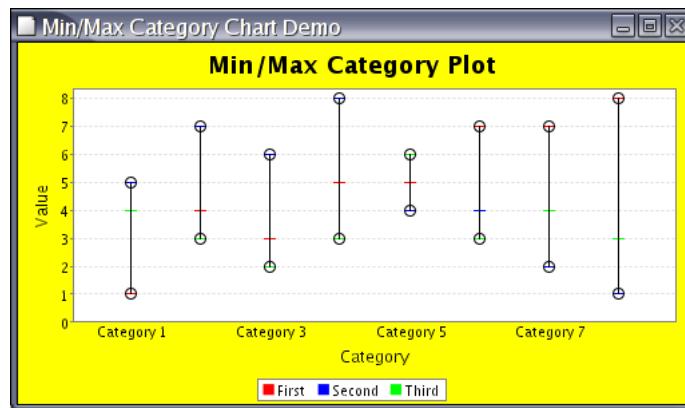
### 32.16.4 Equals, Cloning and Serialization

This renderer overrides the `equals()` method, and is `Cloneable` and `Serializable`. For general issues about these methods, refer to section [31.2.5](#).

## 32.17 MinMaxCategoryRenderer

### 32.17.1 Overview

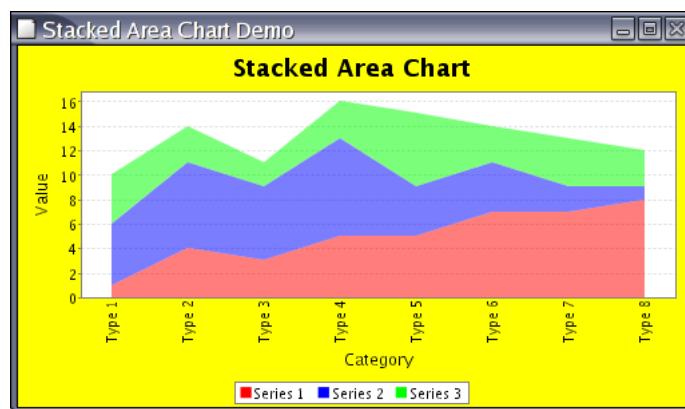
A renderer that draws minimum and maximum markers.



## 32.18 StackedAreaRenderer

### 32.18.1 Overview

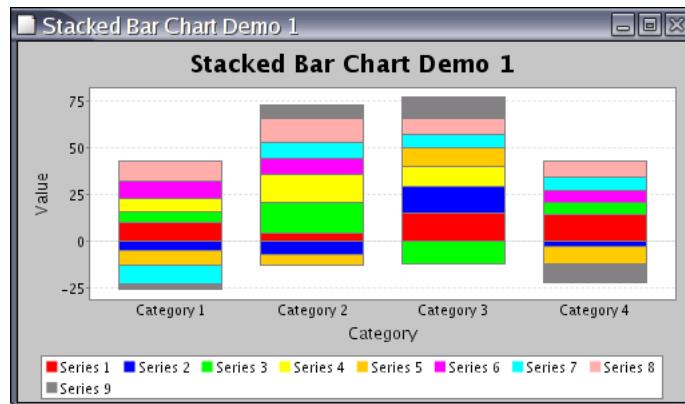
A stacked area renderer that draws items from a `CategoryDataset`.



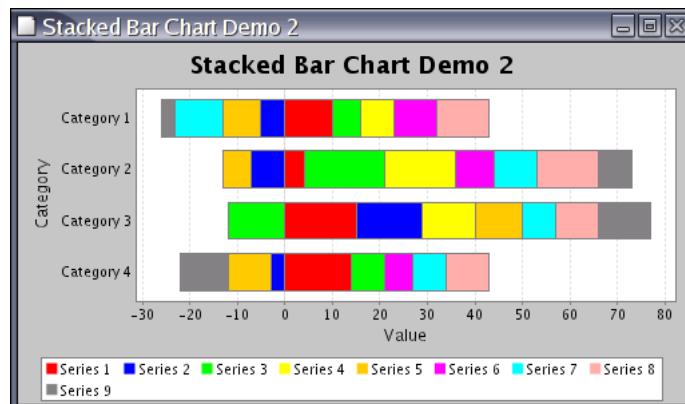
## 32.19 StackedBarRenderer

### 32.19.1 Overview

A *stacked bar renderer* draws each item in a [CategoryDataset](#) in the form of “stacked” bars. For example:



Here is another example, this time with a horizontal orientation:



This renderer is designed for use with the [CategoryPlot](#) class.

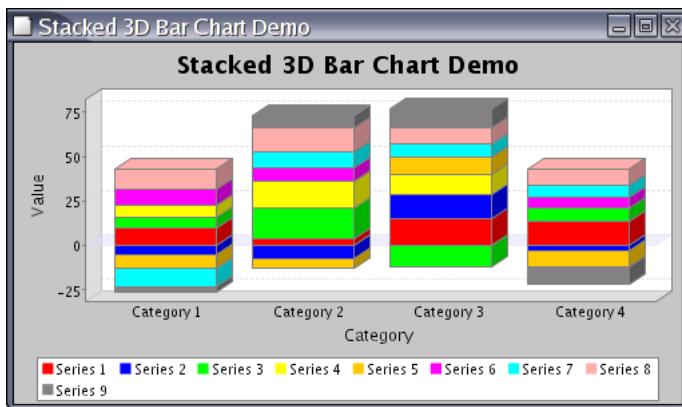
### 32.19.2 Methods

This class implements the methods in the [CategoryItemRenderer](#) interface.

## 32.20 StackedBarRenderer3D

### 32.20.1 Overview

A *stacked bar renderer (3D)* draws items from a [CategoryDataset](#) in the form of “stacked” bars with a 3D effect.



This renderer is designed for use with the [CategoryPlot](#) class.

### 32.20.2 Methods

This class implements the methods in the [CategoryItemRenderer](#) interface.

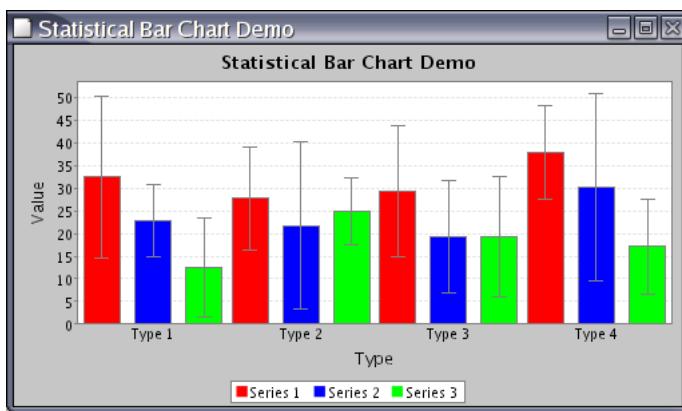
#### See Also

[StackedBarRenderer](#).

## 32.21 StatisticalBarRenderer

### 32.21.1 Overview

A *statistical bar renderer* draws items from a [StatisticalCategoryDataset](#) in the form of bars with a line indicating the standard deviation.



This renderer is designed for use with the [CategoryPlot](#) class.

### 32.21.2 Notes

This class implements the [CategoryItemRenderer](#) interface.

## 32.22 WaterfallBarRenderer

### 32.22.1 Overview

A renderer for drawing waterfall charts.

### 32.22.2 Notes

There is a demo (`WaterfallChartDemo1.java`) included in the JFreeChart Premium Demo distribution.

# Chapter 33

## Package: `org.jfree.chart.renderer.xy`

### 33.1 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.

### 33.2 AbstractXYItemRenderer

#### 33.2.1 Overview

A convenient base class for creating new `XYItemRenderer` implementations.

#### 33.2.2 Constructors

This class provides a default constructor which allocates storage for the label generator(s), the tool tip generator(s) and the URL generator.

```
protected AbstractXYItemRenderer();  
Creates a new renderer.
```

#### 33.2.3 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.
```

### 33.2.4 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.

### 33.2.5 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.

### 33.2.6 Grid Bands

It is possible to fill the space between alternate grid lines with a different color to create a “band” effect.

### 33.2.7 Methods

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).

### 33.2.8 Notes

Some points to note:

- this class provides a property change mechanism to support the requirements of the `XYItemRenderer` interface;

#### See Also

`XYItemRenderer`, `XYPlot`.

## 33.3 CandlestickRenderer

### 33.3.1 Overview

A *candlestick renderer* draws each item from a [HighLowDataset](#) 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).

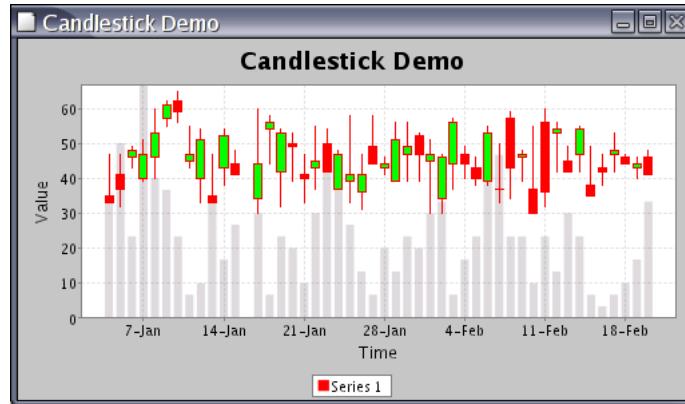


Figure 33.1: A sample chart using *CandlestickRenderer*

This renderer is designed for use with the [XYPlot](#) class.

This renderer also has the ability to represent volume information in the background of the chart.

### 33.3.2 Constructors

To create a new renderer:

```
public CandlestickRenderer(double candleWidth);
Creates a new renderer.
```

### 33.3.3 Methods

To set the width of the candles (in points):

```
public void setCandleWidth(double width);
Sets the width of each candle. If the value is negative, then the renderer
will automatically determine a width each time the chart is redrawn.
```

To set the color used to fill candles when the closing price is higher than the opening price (the price has moved up):

```
public void setUpPaint(Paint paint);
Sets the fill color for candles where the closing price is higher than the
opening price.
```

To set the color used to fill candles when the closing price is lower than the opening price (the price has moved down):

```
public void setDownPaint(Paint paint);
```

Sets the fill color for candles where the closing price is lower than the opening price.

To control whether or not volume bars are drawn in the background of the chart:

```
public void setDrawVolume(boolean flag);
```

Controls whether or not volume bars are drawn in the background of the chart.

These methods will fire a property change event that will be picked up by the [XYPlot](#) class, triggering a chart redraw.

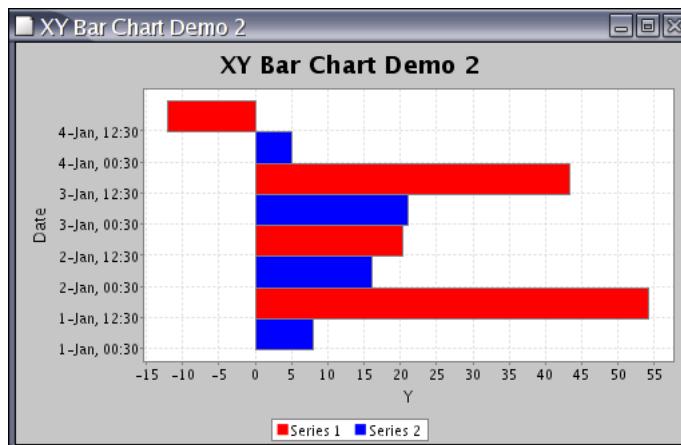
### 33.3.4 Notes

This renderer requires a [HighLowDataset](#).

## 33.4 ClusteredXYBarRenderer

### 33.4.1 Overview

An *XY bar renderer* draws items from an [IntervalXYDataset](#) in the form of bars.



This renderer is designed to work with an [XYPlot](#).

### 33.4.2 Constructors

The only constructor takes no arguments.

### 33.4.3 Methods

The `drawItem()` method handles the rendering of a single item for the plot.

### 33.4.4 Notes

This renderer casts the dataset to [IntervalXYDataset](#), so you should ensure that the plot is supplied with the correct type of data. It would probably be a good idea to merge this class with the [XYBarRenderer](#) class, but this hasn't been done yet.

## 33.5 CyclicXYItemRenderer

### 33.5.1 Overview

A renderer for drawing “cyclic” charts.

## 33.6 DefaultXYItemRenderer

### 33.6.1 Overview

This class is an alias for the [XYLineAndShapeRenderer](#) class.

## 33.7 HighLow

### 33.7.1 Overview

Represents one item used by a [HighLowRenderer](#) during the rendering process.

## 33.8 HighLowRenderer

### 33.8.1 Overview

A *high-low* renderer draws each item in an [XYDataset](#) using lines to mark the “high-low” range for a trading period, plus small marks to indicate the “open” and “close” values.

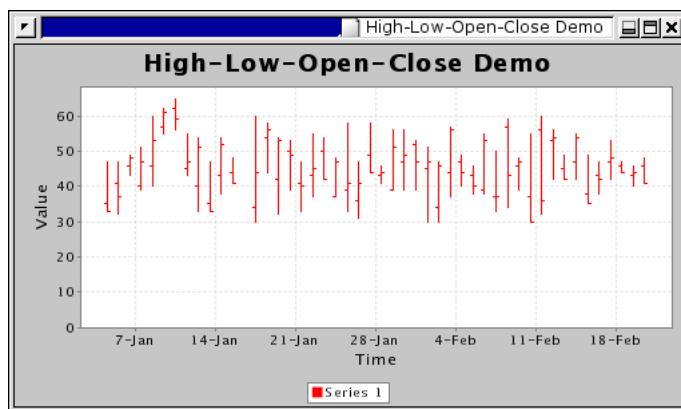


Figure 33.2: A chart that uses a *HighLowRenderer*

This renderer is designed for use with the [XYPlot](#) class. It requires a [HighLowDataset](#).

### 33.8.2 Constructors

To create a new renderer:

```
public HighLowRenderer();
Creates a new renderer.
```

### 33.8.3 Methods

Implements the `drawItem()` method defined in the [XYItemRenderer](#) interface.

### 33.8.4 Notes

This renderer requires the dataset to be an instance of [HighLowDataset](#).

The `createHighLowChart()` method in the [ChartFactory](#) class makes use of this renderer.

## 33.9 StackedXYAreaRenderer

### 33.9.1 Overview

A stacked area renderer that draws items from a [TableXYDataset](#). An example is shown in figure 33.3.

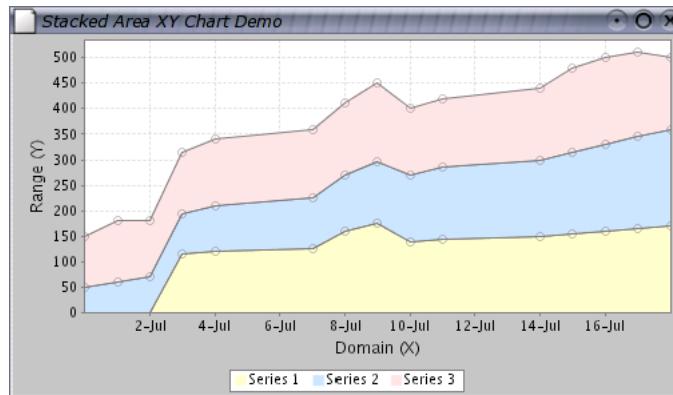


Figure 33.3: A chart created using `StackedXYAreaRenderer`

## 33.10 StackedXYBarRenderer

### 33.10.1 Overview

A renderer for drawing stacked bar charts using data from a [TableXYDataset](#)—see figure 33.4 for an example.

This class extends [XYBarRenderer](#).

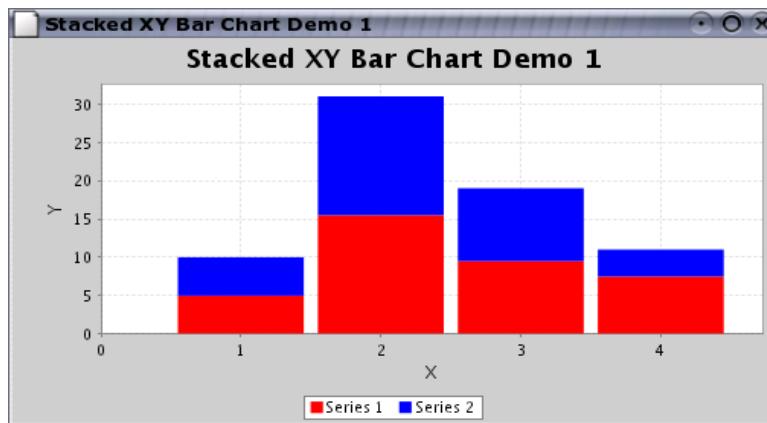


Figure 33.4: A chart generated with a `StackedXYBarRenderer`.

### 33.10.2 Constructors

There are two constructors:

```
public StackedXYBarRenderer();
```

Creates a new instance with default settings.

```
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.

### 33.10.3 Methods

This renderer extends `XYBarRenderer`. The following methods are overridden:

```
public Range getRangeExtent(XYDataset dataset);
```

Calculates the range of values represented by the dataset, taking into account the fact that the renderer “stacks” values and that the base value may be non-zero (see the `getBase()` method in the `XYBarRenderer` class).

```
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(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 need to call it yourself.

### 33.10.4 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”).
- a demo (`StackedXYBarChartDemo1.java`) is included in the JFreeChart Demo distribution.

## 33.11 StandardXYItemRenderer

### 33.11.1 Overview

A standard *renderer* for the `XYPlot` class. This renderer 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).

### 33.11.2 Constructors

To create a `StandardXYItemRenderer`:

```
public StandardXYItemRenderer(int type);
Creates a new renderer. The type argument should be one of: LINES,
SHAPES or SHAPES_AND_LINES.
```

### 33.11.3 Methods

To control whether or not the renderer draws lines between data points:

```
public void setPlotLines(boolean flag);
Sets the flag that controls whether or not lines are plotted between data
points. The stroke and paint used for the lines is determined by the plot,
per series.
```

To control whether or not the renderer draws shapes at each data point:

```
public void setPlotShapes(boolean flag);
Sets the flag that controls whether or not shapes are plotted at each data
point.
```

For each item, the shape to be plotted is obtained from the `getShape()` method which, unless overridden, delegates to the plot’s `getShape()` method (which will return a different shape for each series).

When the renderer draws each shape, it can draw an outline of the shape, or it can fill the shape with a solid color. This is controlled by a protected method:

```
protected boolean isShapeFilled();
Returns a flag that controls whether or not the shape is filled.
```

By default, this method returns the value from the `getDefaultShapeFilled()` method, but you can override the method in a subclass to customise the behaviour.

### 33.11.4 Notes

This class implements the `XYItemRenderer` interface.

The `XYPlot` class will use an instance of this class as its default renderer.

## 33.12 WindItemRenderer

### 33.12.1 Overview

A renderer that [XYPlot](#) uses to draw wind plots.

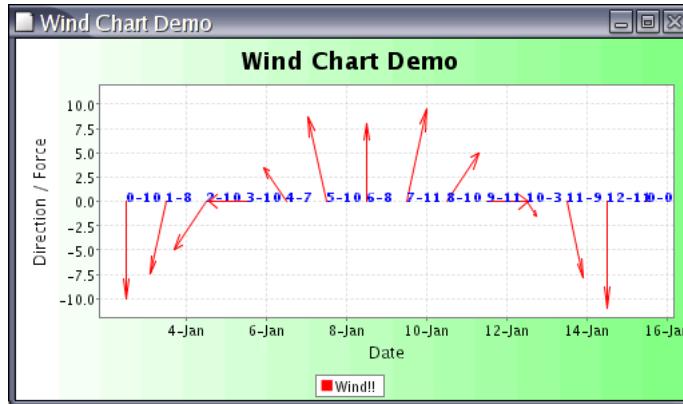


Figure 33.5: A sample chart using `WindItemRenderer`

## 33.13 XYAreaRenderer

### 33.13.1 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 33.6 for an example.

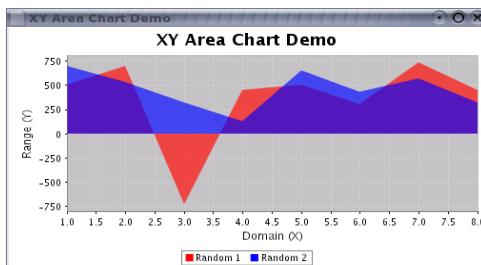


Figure 33.6: A chart using `XYAreaRenderer`

This renderer is designed to be used with the [XYPlot](#) class.

### 33.13.2 Constructors

The default constructor sets up the renderer to draw area charts:

```
public XYAreaRenderer();
Creates a new renderer.
```

You can change the appearance of the chart by specifying the type:

```
public XYAreaRenderer(int type);
Creates a new XYAreaRenderer using one of the following types: SHAPES,
LINES, SHAPES_AND_LINES, AREA, AREA_AND_SHAPES.
```

A further constructor allows you to specify the tool tip and URL generators:

```
public XYAreaRenderer(int type, XYToolTipGenerator labelGenerator,
XYURLGenerator urlGenerator);
Creates a new renderer with the specified tool tip generator and URL
generator.
```

### 33.13.3 Methods

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.
```

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.
```

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).
```

### 33.13.4 Notes

Some points to note:

- this class extends `AbstractXYItemRenderer`;

- instances of this class are cloneable and serializable;
- this class uses code copied from the [StandardXYItemRenderer](#) class, and that some additional work is required to eliminate the duplication. One option (still under consideration) for a future version of JFreeChart is to merge the two classes.

#### See Also

[AreaRenderer](#).

## 33.14 XYBarRenderer

### 33.14.1 Overview

This renderer can be used within an [XYPlot](#) to draw bar charts with data from an [IntervalXYDataset](#)—see figure 33.7 for an example.

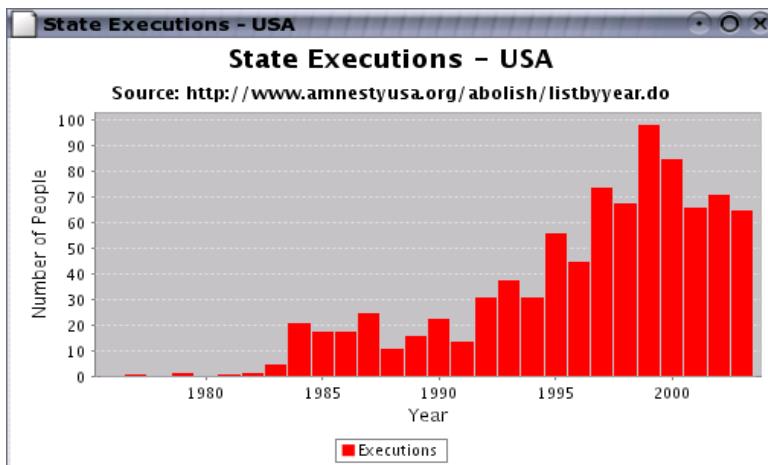


Figure 33.7: A chart generated with an *XYBarRenderer*.

Related to this class is [ClusteredXYBarRenderer](#).

### 33.14.2 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).

### 33.14.3 Methods

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.

```
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.

This renderer supports the use of `GradientPaint` for any series color by using a transformer:

```
public GradientPaintTransformer getGradientPaintTransformer();
```

Returns the transformer used for `GradientPaint` instances.

```
public void setGradientPaintTransformer(GradientPaintTransformer transformer);
```

Sets the transformer used for `GradientPaint` instances.

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.

To clone the renderer:

```
public Object clone() throws CloneNotSupportedException;
    Returns a clone of the renderer.
```

### 33.14.4 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.

## 33.15 XYBoxAndWhiskerRenderer

### 33.15.1 Overview

A renderer that is used to create a box-and-whisker chart using data from an [XYBoxAndWhiskerDataset](#). A sample chart is shown in Figure 33.8.

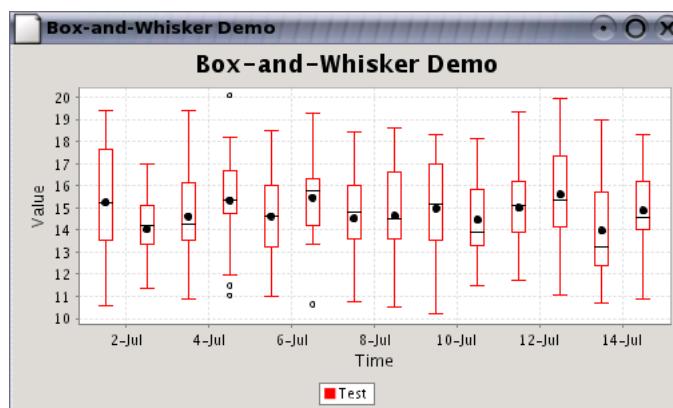


Figure 33.8: A chart generated with an [XYBoxAndWhiskerRenderer](#).

### 33.15.2 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.
```

### 33.15.3 Notes

Some points to note:

- for tool tips, you can use the [BoxAndWhiskerXYToolTipGenerator](#) class;
- there is a demo ([XYBoxAndWhiskerDemo1.java](#)) included in the JFreeChart Premium Demo distribution.

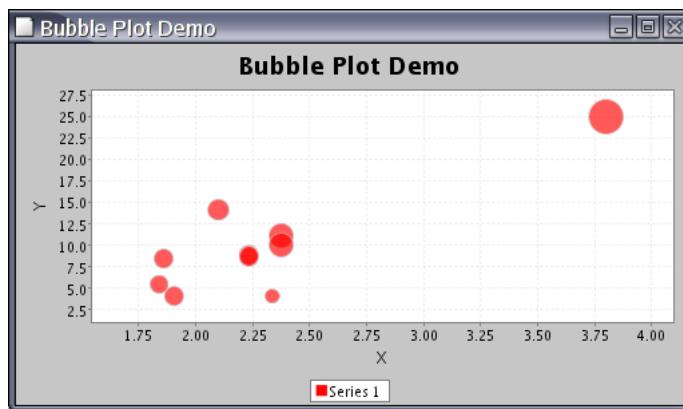
**See Also**

[BoxAndWhiskerRenderer](#).

## 33.16 XYBubbleRenderer

### 33.16.1 Overview

An *XY bubble renderer* displays items from an [XYZDataset](#) by drawing a bubble at each  $(x, y)$  point.



### 33.16.2 Notes

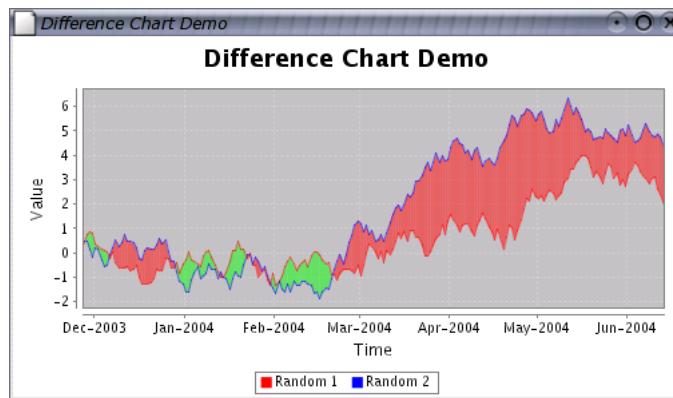
Some notes:

- this class implements the [XYItemRenderer](#) interface and extends the [AbstractXYItemRenderer](#) class.
- the [BubblePlotDemo](#) application (included in the JFreeChart Premium Demo distribution) provides a demonstration of this renderer.

## 33.17 XYDifferenceRenderer

### 33.17.1 Overview

A renderer that displays the difference between two series.



The `DifferenceChartDemo1.java` application (included in the JFreeChart Premium Demo distribution) provides an example of this renderer being used.

## 33.18 XYDotRenderer

### 33.18.1 Overview

A renderer that can be used by an `XYPlot` to display items from an `XYDataset`. The renderer draws a pixel-sized dot at each  $(x, y)$  point—see figure 33.9 for an example.

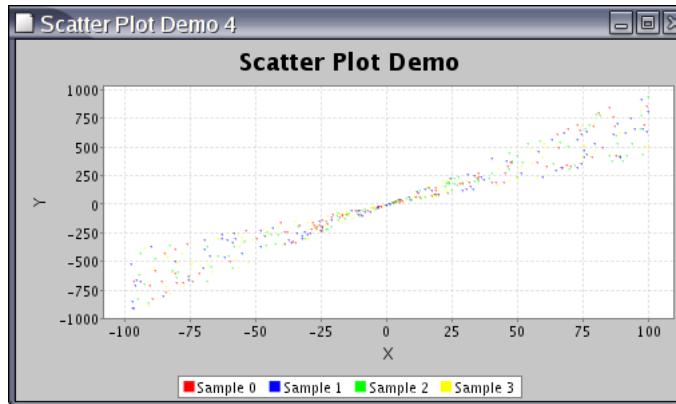


Figure 33.9: A chart generated with an `XYDotRenderer`.

This class implements the `XYItemRenderer` interface.

### 33.18.2 Constructor

The default constructor is the only constructor available:

```
public XYDotRenderer();
Creates a new renderer.
```

### 33.18.3 Methods

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.

### 33.18.4 Notes

Some points to note:

- this class extends the `AbstractXYItemRenderer` class;
- tooltips, item labels and URLs are NOT generated by this renderer (these features may be added in a future release);
- this class implements the `PublicCloneable` interface;
- instances of this class are `Serializable`;
- a demo application (`ScatterPlotDemo4.java`) is included in the JFreeChart Premium Demo distribution.

## 33.19 XYItemRenderer

### 33.19.1 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 33.10 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 33.1.

Class:	Description:
<code>HighLowRenderer</code>	High-low-open-close 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>XYBubbleRenderer</code>	Bubble charts.
<code>XYDifferenceRenderer</code>	Difference charts.
<code>XYDotRenderer</code>	Scatter plots.
<code>XYStepRenderer</code>	Step charts.
<code>YIntervalRenderer</code>	Interval charts.

Table 33.1: Classes that implement the `XYItemRenderer` interface

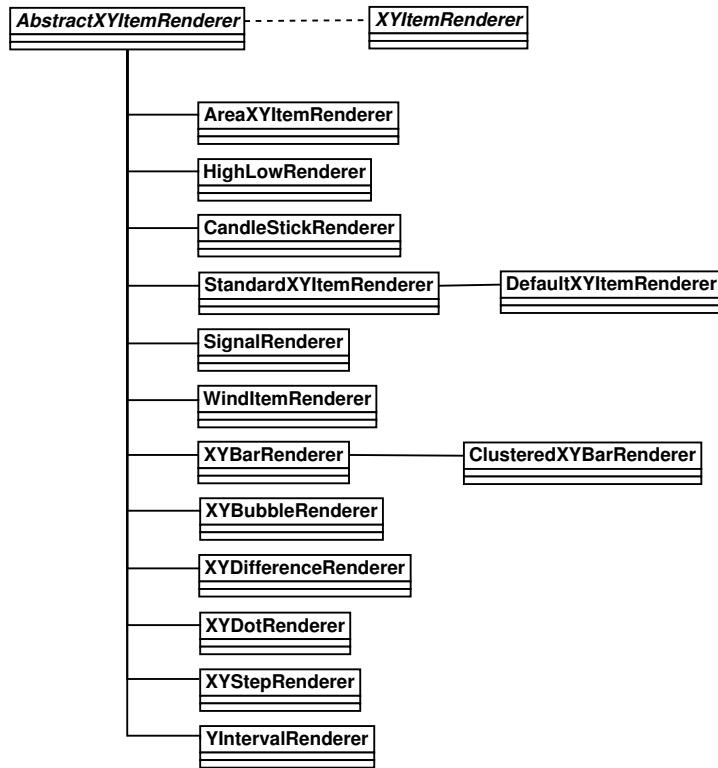


Figure 33.10: Renderer hierarchy

### 33.19.2 Core Methods

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 data, ChartRenderingInfo info);
Initialises the renderer. If possible, a renderer will pre-calculate any values
that help to improve the performance of the drawItem() method.
  
```

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 data, int series, int item, CrosshairInfo info);
Draws a single data item on behalf of XYPlot.
  
```

You can set your own *tooltip generator* and *URL generator* for the renderer.

### 33.19.3 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.

### 33.19.4 Notes

Some renderers require a dataset that is a specific extension of `XYDataset`. For example, the `HighLowRenderer` requires a `HighLowDataset`.

#### See Also

`AbstractXYItemRenderer`, `XYPlot`.

## 33.20 XYItemRendererState

### 33.20.1 Overview

To be documented.

## 33.21 XYLineAndShapeRenderer

### 33.21.1 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. 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.

### 33.21.2 Usage

Often this renderer is used as a replacement for the default renderer in a time series chart. In the following code, a new renderer is created and used to replace an existing renderer:

```
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.

### 33.21.3 Constructor

There is a single constructor for this class:

```
public XYLineAndShapeRenderer();
Creates a new renderer. By default, the renderer will draw lines and filled
shapes for all series in the dataset.
```

### 33.21.4 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.
```

To determine whether or not a line is drawn for an item (connecting the current item with the previous item):

```
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.
```

To determine whether or not lines are drawn for the items in ALL series:

```
public Boolean getLinesVisible();
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.
```

```
public void setLinesVisible(Boolean visible)
Sets the flag that controls whether or not lines are drawn for the items
in ALL series. You can set this flag to null if you prefer to use the “per
series” flags.
```

```
public void setLinesVisible(boolean visible)
Sets the flag that controls whether or not lines are drawn for the items in
ALL series.
```

To determine whether or not lines are drawn for the items in one series (this requires the flag above to be set to null):

```
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. If this is set to null, then the default value will apply.

public void setSeriesLinesVisible(int series, boolean visible);
Sets a flag that controls whether or not lines are drawn for the items in
the specified series.
```

The flags are stored as `Boolean` objects—if the flag is `null` for a series, then the default value is returned. You can set the default value using:

```
public void setDefaultLinesVisible(boolean flag);
Sets the default flag that controls whether or not the renderer draws lines
between the (x, y) items in a series.
```

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`.

The settings that control whether or not shapes are drawn and filled follow a very similar pattern. There are default values that can be overridden on a *per series* basis.

### 33.21.5 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;
- there is a demo (`XYLineAndShapeRendererDemo1.java`) included in the JFreeChart Premium Demo distribution.

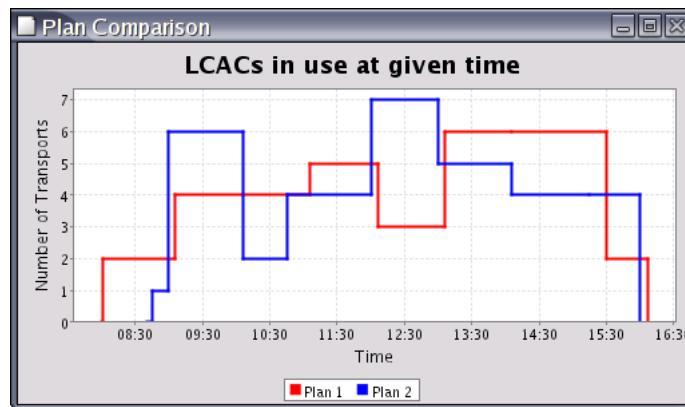
## 33.22 XYStepRenderer

### 33.22.1 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.

### 33.22.2 Usage

A demo (`XYStepChartDemo1.java`) is included in the JFreeChart Premium Demo distribution.

Figure 33.11: A sample chart using `XYStepRenderer`

## 33.23 XYStepAreaRenderer

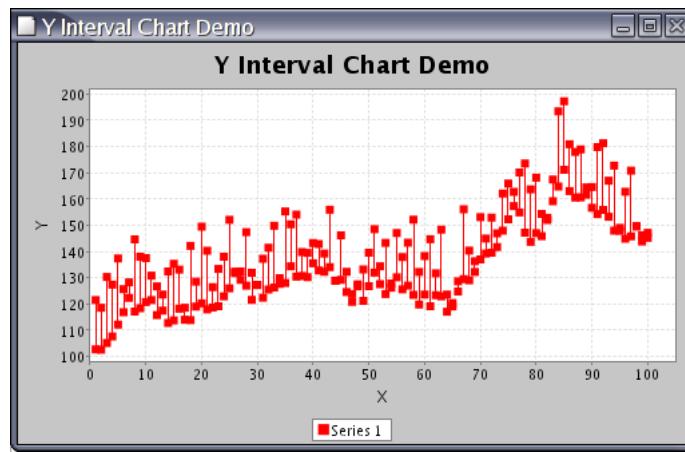
### 33.23.1 Overview

To be documented.

## 33.24 YIntervalRenderer

### 33.24.1 Overview

An `XYItemRenderer` that draws lines between the starting and ending y values from an `IntervalXYDataset`.

Figure 33.12: A sample chart using `YIntervalRenderer`

This renderer is designed for use with the `XYPlot` class.

### 33.24.2 Notes

The `YIntervalChartDemo` class in the `org.jfree.chart.demo` package provides an example of this renderer in use.

# Chapter 34

## Package: `org.jfree.chart.servlet`

### 34.1 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)

### 34.2 ChartDeleter

#### 34.2.1 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`.

### 34.3 DisplayChart

#### 34.3.1 Overview

A servlet that displays a chart image from the temporary directory.

### 34.4 ServletUtilities

#### 34.4.1 Overview

A utility class for performing operations in a servlet environment.

#### 34.4.2 Methods

To save a chart in the temporary directory:

```
public static String saveChartAsPNG(JFreeChart chart, int width, int height,  
ChartRenderingInfo info, HttpSession session);
```

Saves a chart to a PNG image file in the temporary directory. 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. Note that the temporary file name prefix can be set using the `setTempFilePrefix()` method.

# Chapter 35

## Package: org.jfree.chart.title

### 35.1 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.

### 35.2 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 [ChartChangeListeners](#). If the chart is displayed in a [ChartPanel](#), the panel will receive a [ChartChangeEvent](#) and respond by repainting the chart.

### 35.3 DateTitle

#### 35.3.1 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.

#### 35.3.2 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.

### 35.3.3 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 `Title` class.

## 35.4 ImageTitle

### 35.4.1 Overview

A chart title that displays an image (extends `Title`).

### 35.4.2 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.

### 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.

## 35.5 LegendTitle

### 35.5.1 Overview

This class is ultimately intended to make the legend behave in the same way as all other chart titles, but is currently incomplete.

## 35.6 TextTitle

### 35.6.1 Overview

A chart title that displays a text string (extends `Title`).

### 35.6.2 Constructors

To create a text title for a chart:

```
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`.

### 35.6.3 Methods

To set the title string:

```
public void setText(String text);
```

Sets the text for the title and sends a `TitleChangeEvent` to all registered listeners.

To set the font for the title:

```
public void setFont(Font font);
```

Sets the font for the title and sends a `TitleChangeEvent` to all registered listeners.

To set the color of the title:

```
public void setPaint(Paint paint);
```

Sets the paint used to display the title text and sends a `TitleChangeEvent` to all registered listeners.

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.

### 35.6.4 Notes

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.ccss.de/slovo/unifonts.htm>

## 35.7 Title

### 35.7.1 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.

### 35.7.2 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.
```

### 35.7.3 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.
```

### 35.7.4 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.
```

### 35.7.5 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.
```

### 35.7.6 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.
```

### 35.7.7 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 36

## Package: org.jfree.chart.ui

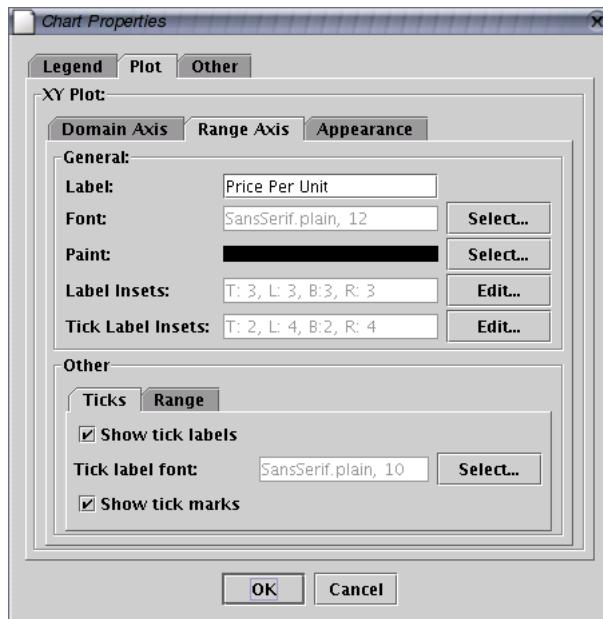
### 36.1 Introduction

This package contains user interface classes that can be used to modify chart properties. These classes are optional—they are used in the demonstration application, but you do not need to include this package in your own projects if you do not want to.

### 36.2 AxisPropertyEditPanel

#### 36.2.1 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 the property editors before JFreeChart 1.0.0 is released.

### 36.3 ChartPropertyEditPanel

#### 36.3.1 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 four sub-panels:

- a `TitlePropertyEditPanel`;
- a `LegendPropertyEditPanel`;
- a `PlotPropertyEditPanel`;
- 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 `TitlePropertyEditPanel` and is used if and when a sub-dialog is required for editing titles.

### 36.4 ColorBarPropertyEditPanel

#### 36.4.1 Overview

A panel for editing the properties of a `ColorBar`.

### 36.5 ColorPalette

#### 36.5.1 Overview

The abstract base class for the color palettes used by the `ContourPlot` class.

### 36.6 GreyPalette

#### 36.6.1 Overview

A grey palette (extends `ColorPalette`).

### 36.7 LegendPropertyEditPanel

#### 36.7.1 Overview

A panel for displaying and editing the properties of a chart legend. 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 the property editors before JFreeChart 1.0.0 is released.

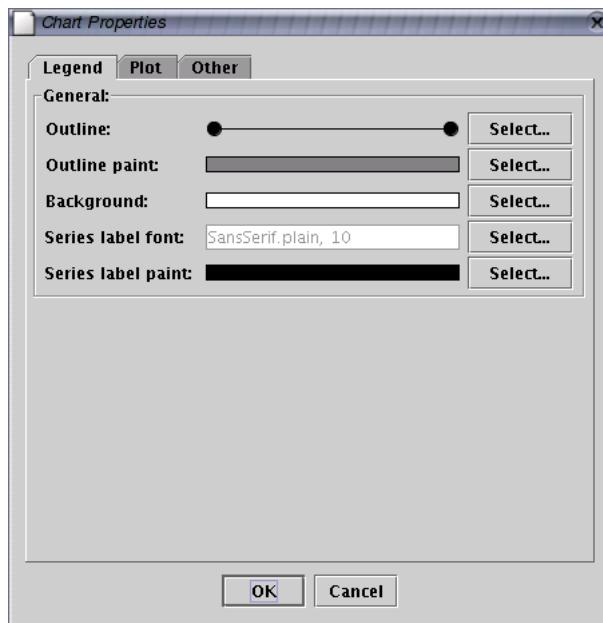


Figure 36.1: The legend property editor

## 36.8 NumberAxisPropertyEditPanel

### 36.8.1 Overview

A panel for displaying and editing the properties of a [NumberAxis](#).

## 36.9 PaletteChooserPanel

### 36.9.1 Overview

A panel for selecting a color palette.

## 36.10 PaletteSample

### 36.10.1 Overview

To be documented.

## 36.11 PlotPropertyEditPanel

### 36.11.1 Overview

A panel for displaying and editing the properties of a plot.

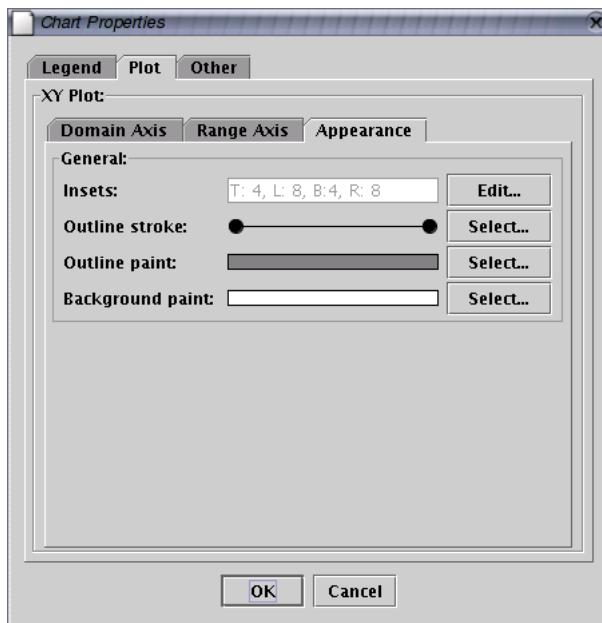


Figure 36.2: The plot property editor

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 the property editors before JFreeChart 1.0.0 is released.

## 36.12 RainbowPalette

### 36.12.1 Overview

A rainbow palette (extends [ColorPalette](#)).

## 36.13 TitlePropertyEditPanel

### 36.13.1 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 the property editors before JFreeChart 1.0.0 is released.

# Chapter 37

## Package: org.jfree.chart.urls

### 37.1 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.

### 37.2 CategoryURLGenerator

#### 37.2.1 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.

#### 37.2.2 Methods

This method returns a URL for a specific data item:

```
public String generateURL(CategoryDataset data, 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.
```

### 37.2.3 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.

## 37.3 CustomPieURLGenerator

### 37.3.1 Overview

To be documented.

## 37.4 CustomXYURLGenerator

### 37.4.1 Overview

A URL generator that uses custom strings as the URL for each item in an `XYDataset`. This class implements the `XYURLGenerator` interface.

## 37.5 PieURLGenerator

### 37.5.1 Overview

A *pie URL generator* is used by a `PiePlot` to generate URLs for use in HTML image maps.

### 37.5.2 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.

### 37.5.3 Notes

Some points to note:

- the `StandardPieURLGenerator` class is the only implementation of this interface provided in the JFreeChart class library.
- the `ChartUtilities` class contains methods for writing HTML image maps.

## 37.6 StandardCategoryURLGenerator

### 37.6.1 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>
```

This class implements the `CategoryURLGenerator` interface.

### 37.6.2 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.

### 37.6.3 Constructors

To create a new generator:

```
public StandardCategoryURLGenerator(String prefix,
String seriesParameterName, String categoryParameterName);
Creates a new generator with the specified attributes.
```

### 37.6.4 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 data, int series, int category)
Returns a string that will be used as the URL for the specified data item.
```

### 37.6.5 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.

## 37.7 StandardPieURLGenerator

### 37.7.1 Overview

A default URL generator for use when creating HTML image maps for pie charts. This class implements the [PieURLGenerator](#) interface.

### 37.7.2 Constructor

To create a new generator:

```
public StandardPieURLGenerator(String prefix, String categoryParameterName);  
Creates a new generator.
```

## 37.8 StandardXYURLGenerator

### 37.8.1 Overview

A default URL generator for creating HTML image maps. This class implements the [XYURLGenerator](#) interface.

## 37.9 StandardXYZURLGenerator

### 37.9.1 Overview

A URL generator that creates URLs for the items in an [XYZDataset](#).

## 37.10 TimeSeriesURLGenerator

### 37.10.1 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.

## 37.11 XYURLGenerator

### 37.11.1 Overview

An *XY URL generator* is used by a [XYItemRenderer](#) to generate URLs for use in HTML image maps.

### 37.11.2 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.
```

### 37.11.3 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.

## 37.12 XYZURLGenerator

### 37.12.1 Overview

An *XYZ URL generator* is used by a `XYItemRenderer` to generate URLs for use in HTML image maps.

### 37.12.2 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.
```

### 37.12.3 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.

# Chapter 38

## Package: org.jfree.data

### 38.1 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.

### 38.2 DefaultKeyValue

#### 38.2.1 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.

#### 38.2.2 Usage

This class is typically used to represent individual data items in a larger collection, such as [DefaultKeyedValues](#).

#### 38.2.3 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.
```

#### 38.2.4 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.
```

### 38.2.5 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.

## 38.3 DefaultKeyedValues

### 38.3.1 Overview

A collection of (*key, value*) data items, where the key is an instance of `Comparable` and the value is an instance of `Number`.

### 38.3.2 Notes

Some points to note:

- this class provides a default implementation of the `KeyedValues` interface;
- the `DefaultPieDataset` class uses an instance of this class to store its data.

## 38.4 DefaultKeyedValues2D

### 38.4.1 Overview

A storage structure for a table of values that are associated with keys. This class provides a default implementation of the `KeyedValues2D` interface.

### 38.4.2 Notes

The `DefaultCategoryDataset` class uses an instance of this class to store its data.

## 38.5 DomainInfo

### 38.5.1 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.

### 38.5.2 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.

### 38.5.3 Notes

It is not mandatory for a dataset to implement this interface.

#### See Also

[RangeInfo](#), [DatasetUtilities](#).

## 38.6 DomainOrder

### 38.6.1 Overview

An enumeration of the order of the domain values in a dataset—see table 38.1 for a list of the defined values.

ID:	Description:
<code>DomainOrder.ASCENDING</code>	Ascending order.
<code>DomainOrder.DESCENDING</code>	Descending order.
<code>DomainOrder.NONE</code>	No order.

Table 38.1: Constants defined by `DomainOrder`

## 38.7 KeyedObject

### 38.7.1 Overview

Not yet documented.

## 38.8 KeyedObjects

### 38.8.1 Overview

Not yet documented.

## 38.9 KeyedObjects2D

### 38.9.1 Overview

Not yet documented.

## 38.10 KeyedValue

### 38.10.1 Overview

A *keyed value* is a value (`Number`) that is associated with a key (`Comparable`).

### 38.10.2 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.
```

### 38.10.3 Notes

The `DefaultKeyValue` class provides one implementation of this interface.

## 38.11 KeyValueComparator

### 38.11.1 Overview

This class is used to compare two `KeyValue` objects, either by key or by value.

## 38.12 KeyValueComparatorType

### 38.12.1 Overview

Used to represent the two comparison types—*by key* or *by value*—used by the `KeyValueComparator` class.

## 38.13 KeyedValues

### 38.13.1 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.

### 38.13.2 Methods

To access the key associated with a value:

```
public Comparable getKey(int index);
>Returns the key associated with an item in the collection.
```

To convert a key into an item index:

```
public int getIndex(Comparable key);
>Returns the item index for a key.
```

To get a list of all keys in the collection:

```
public List getKeys();
>Returns a 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.
```

### 38.13.3 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.

## 38.14 KeyedValues2D

### 38.14.1 Overview

A table of values that can be accessed using a *row key* and a *column key*. This interface extends the `Values2D` interface.

### 38.14.2 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).

### 38.14.3 Notes

The [DefaultKeyedValues2D](#) class provides one implementation of this interface.

## 38.15 KeyToGroupMap

### 38.15.1 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.

### 38.15.2 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”.

### 38.15.3 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).

### 38.15.4 Notes

Some points to note:

- an instance of this class is used by the `GroupedStackedBarRenderer` class.

## 38.16 Range

### 38.16.1 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.

### 38.16.2 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.

### 38.16.3 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.

### 38.16.4 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.

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.

### 38.16.5 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.
```

### 38.16.6 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).

### 38.16.7 Notes

Some points to note:

- the `DateRange` class extends this class to support a date range.

## 38.17 RangeInfo

### 38.17.1 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.

### 38.17.2 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.
```

To get the maximum value in the dataset's range:

```
public double getRangeUpperBound(boolean includeInterval);
Returns the upper bound for the dataset's range.
```

To get the range of values in the dataset's range:

```
public Range getRangeBounds(boolean includeInterval);  
Returns the bounds for the dataset's range.
```

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.

### 38.17.3 Notes

It is not mandatory for a dataset to implement this interface.

#### See Also

[DomainInfo](#).

## 38.18 Value

### 38.18.1 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.

### 38.18.2 Methods

The interface defines a single method for accessing the value:

```
public Number getValue();  
Returns the value (possibly null).
```

### 38.18.3 Notes

Some notes:

- the `KeyedValue` interface extends this interface.
- the `DefaultKeyedValue` class provides one implementation of this interface.

## 38.19 Values

### 38.19.1 Overview

An interface for accessing a collection of values.

### 38.19.2 Methods

To get the number of items in the collection:

```
public int getItemCount();  
Returns the number of items in the collection.
```

To get a value from the collection:

```
public Number getValue(int item);  
Returns a value from the collection (possibly null).
```

### 38.19.3 Notes

Some notes:

- the `KeyedValues` interface extends this interface.
- the `DefaultKeyedValues` class provides one implementation of this interface.

## 38.20 Values2D

### 38.20.1 Overview

An interface for accessing a table of values by row and column index.

### 38.20.2 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.
```

### 38.20.3 Notes

Some points to note:

- the `KeyedValues2D` interface extends this interface.
- the `DefaultKeyedValues2D` class provides one implementation of this interface.

# Chapter 39

## Package: `org.jfree.data.category`

### 39.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

### 39.2 CategoryDataset

#### 39.2.1 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.

#### 39.2.2 Methods

This interface adds no additional methods to those defined in the `KeyedValues2D` and `Dataset` interfaces.

#### 39.2.3 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](#).

## 39.3 CategoryToPieDataset

### 39.3.1 Overview

A utility class that presents one row or column of data from a [CategoryDataset](#) via the [PieDataset](#) interface.

### 39.3.2 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.

### 39.3.3 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.

## 39.4 DefaultCategoryDataset

### 39.4.1 Overview

A default implementation of the [CategoryDataset](#) interface.

### 39.4.2 Constructors

The default constructor creates a new, empty dataset:

```
public DefaultCategoryDataset();  
Creates a new dataset.
```

The [DatasetUtilities](#) class has static methods for creating instances of this class using array data.

### 39.4.3 Methods

To add a value to the dataset:

```
public.addValue(Number value, Comparable rowKey, Comparable columnKey)  
Adds a value to the dataset. The value can be null (to indicate missing  
data). If there is already a value for the given keys, it is overwritten.
```

A similar method accepts a `double` value and converts it to a `Number` object before storing it.

Identical `setValue()` methods are also provided. These function in exactly the same way as the `addValue()` methods.

#### 39.4.4 Notes

This class uses an instance of `DefaultKeyedValues2D` to store its data.

### 39.5 DefaultIntervalCategoryDataset

#### 39.5.1 Overview

A default implementation of the `IntervalCategoryDataset` interface.

### 39.6 IntervalCategoryDataset

#### 39.6.1 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).

#### 39.6.2 Methods

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.

### 39.6.3 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;

# Chapter 40

## Package: org.jfree.data.contour

### 40.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

### 40.2 ContourDataset

#### 40.2.1 Overview

The dataset used by the [ContourPlot](#) class.

#### 40.2.2 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[] getIXIndices();
>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](#).

## 40.3 DefaultContourDataset

### 40.3.1 Overview

A default implementation of the [ContourDataset](#) interface.

**See Also**

[ContourPlot](#)

## 40.4 NonGridContourDataset

### 40.4.1 Overview

A dataset for use with the [ContourPlot](#) class.

# Chapter 41

## Package: org.jfree.data.function

### 41.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

### 41.2 Function2D

#### 41.2.1 Overview

A simple interface for a 2D function. Implementations of this interface include:

- [LineFunction2D](#);
- [PowerFunction2D](#).

It is a simple matter to implement your own functions.

#### 41.2.2 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.
```

#### 41.2.3 Notes

The [DatasetUtilities](#) class provides a method for creating an [XYDataset](#) by sampling the values of a function.

#### See Also

[LineFunction2D](#), [PowerFunction2D](#).

## 41.3 LineFunction2D

### 41.3.1 Overview

A simple function of the form  $y = a + bx$ .

### 41.3.2 Constructor

To construct a new line function:

```
public LineFunction2D(double a, double b);  
Creates a new line function with the given coefficients.
```

### 41.3.3 Methods

```
public double getValue(double x);  
Returns the value of the function for a given input.
```

### 41.3.4 Notes

This class implements the [Function2D](#) interface.

The [RegressionDemo1](#) application provides an example of this class being used.

#### See Also

[PowerFunction2D](#).

## 41.4 NormalDistributionFunction2D

### 41.4.1 Overview

To be documented.

## 41.5 PowerFunction2D

### 41.5.1 Overview

A function of the form  $y = ax^b$ .

### 41.5.2 Constructor

To construct a new power function:

```
public PowerFunction2D(double a, double b);  
Creates a new power function with the given coefficients.
```

### 41.5.3 Methods

```
public double getValue(double x);  
Returns the value of the function for a given input.
```

#### 41.5.4 Notes

This class implements the [Function2D](#) interface.

The `RegressionDemo1` application provides an example of this class being used.

#### See Also

[LineFunction2D](#).

# Chapter 42

## Package: org.jfree.data.gantt

### 42.1 Introduction

This package contains classes used to represent the dataset for a Gantt chart.

### 42.2 GanttCategoryDataset

#### 42.2.1 Overview

An extension of the [IntervalCategoryDataset](#) interface that is intended for creating Gantt charts.

#### 42.2.2 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.
```

### 42.2.3 Notes

The [GanttRenderer](#) class expects to find a dataset of this type.

## 42.3 Task

### 42.3.1 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](#).

### 42.3.2 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.
```

### 42.3.3 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.
```

### 42.3.4 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.
```

### 42.3.5 Notes

Some points to note:

- this class is `Cloneable` and `Serializable`;
- tasks can be added to a [TaskSeries](#).

## 42.4 TaskSeries

### 42.4.1 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*.

### 42.4.2 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).

### 42.4.3 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.

### 42.4.4 Notes

Some points to note:

- the `TaskSeriesCollection` class is used to create collections of one or more task series.

## 42.5 TaskSeriesCollection

### 42.5.1 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*.

### 42.5.2 Constructor

To create a new collection:

```
public TaskSeriesCollection();
Creates a new collection, initially empty.
```

### 42.5.3 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.
```

### 42.5.4 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).
```

#### 42.5.5 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.
```

#### 42.5.6 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 43

## Package: org.jfree.data.general

### 43.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

### 43.2 AbstractDataset

#### 43.2.1 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.

#### 43.2.2 Constructors

The default constructor:

```
protected AbstractDataset();  
Allocates storage for the registered change listeners.
```

#### 43.2.3 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).
```

#### 43.2.4 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.

#### 43.2.5 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.

#### 43.2.6 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](#).

### 43.3 AbstractSeriesDataset

#### 43.3.1 Overview

A useful base class for implementing the `SeriesDataset` interface (or extensions). This class extends `AbstractDataset`.

#### 43.3.2 Constructors

This class is never instantiated directly, so the constructor is protected:

```
protected AbstractSeriesDataset();
```

Simply calls the constructor of the superclass.

### 43.3.3 Methods

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.

Two abstract methods are declared:

```
public abstract int getSeriesCount();
```

Returns the number of series in the dataset—to be implemented by subclasses.

```
public abstract String getSeriesName(int series);
```

Returns the name of a series in the dataset—to be implemented by subclasses.

### 43.3.4 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](#).

## 43.4 CombinationDataset

### 43.4.1 Overview

An interface that defines the methods that should be implemented by a *combination dataset*.

### 43.4.2 Notes

This interface is implemented by the [CombinedDataset](#) class.

## 43.5 CombinedDataset

### 43.5.1 Overview

A dataset that can combine other datasets.

#### 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](#).

## 43.6 Dataset

### 43.6.1 Overview

The base interface for datasets. Not useful in its own right, this interface is further extended by [PieDataset](#), [CategoryDataset](#) and [SeriesDataset](#).

### 43.6.2 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.
```

### 43.6.3 Notes

This interface is not intended to be used directly, you should use an extension of this interface such as [PieDataset](#), [CategoryDataset](#) or [XYDataset](#).

## 43.7 DatasetChangeEvent

### 43.7.1 Overview

An event that is used to provide information about changes to datasets.

### 43.7.2 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.
```

### 43.7.3 Methods

To get a reference to the `Dataset` that generated the event:

```
public Dataset getDataset();  
Returns the dataset which generated the event.
```

### 43.7.4 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](#).

## 43.8 DatasetChangeListener

### 43.8.1 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.

### 43.8.2 Methods

The interface defines a single method:

```
public void datasetChanged(DatasetChangeEvent event);
```

Receives notification of a change to a dataset.

### 43.8.3 Notes

The [Plot](#) class implements this interface in order to receive notification of changes to its dataset(s).

#### See Also

[DatasetChangeEvent](#).

## 43.9 DatasetGroup

### 43.9.1 Overview

A *dataset group* provides a mechanism for grouping related datasets. At present, this is not used.

### 43.9.2 Constructor

This class has a single constructor:

```
public DatasetGroup();
```

Creates a new group.

### 43.9.3 Methods

The only method in this class creates a clone of the group:

```
public Object clone() throws CloneNotSupportedException;
```

Returns a clone of the group.

### 43.9.4 Notes

As mentioned in the overview, this class currently serves no real purpose.

## 43.10 DatasetUtilities

### 43.10.1 Overview

A collection of utility methods for working with datasets.

### 43.10.2 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 ap-
pending 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 ob-
jects.

public static CategoryDataset createCategoryDataset(Comparable[] rowKeys,
Comparable[] columnKeys, double[][] data);
As for the preceding methods, except that row and column keys are ex-
plicitly 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, String seriesName);
Creates a new XYDataset by sampling values in a specified range for the
Function2D.
```

#### 43.10.3 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.
```

#### 43.10.4 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 im-
plements 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);
```

### 43.10.5 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 de-
fined 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, deter-
mined 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, deter-
mined 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 iteratin-
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.

```

### 43.10.6 Other Methods

```

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](#).

## 43.11 DataUtilities

### 43.11.1 Overview

This class contains utility methods that relate to general data classes.

### 43.11.2 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 per-  
centage values from the specified data.
```

## 43.12 DefaultKeyValueDataset

### 43.12.1 Overview

A dataset that contains a single (*key, value*) data item. This class implements the `KeyValueDataset` interface.

### 43.12.2 Usage

This class does not get used by JFreeChart.

## 43.13 DefaultKeyedValuesDataset

### 43.13.1 Overview

A dataset that implements the `KeyedValuesDataset` interface.

### 43.13.2 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.

## 43.14 DefaultKeyedValues2DDataset

### 43.14.1 Overview

A default implementation of the `KeyedValues2DDataset` interface.

## 43.15 DefaultPieDataset

### 43.15.1 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).

### 43.15.2 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.
```

### 43.15.3 Methods

```
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.

public Comparable getKey(int item);
Returns the key for the given item index.

public int getIndex(Comparable key);
Returns the index for the given key, or -1 if the key is not recognised.

public Number getValue(int item);
Returns the value (possibly null) for the given item.
```

To get the value associated with a key:

```
public Number getValue(Comparable key);
Returns the value associated with a key (possibly null).
```

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.
```

#### 43.15.4 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 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](#).

#### 43.15.5 Notes

The dataset can contain `null` values.

#### See Also

[PieDataset](#), [PiePlot](#).

### 43.16 DefaultValueDataset

#### 43.16.1 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.

#### 43.16.2 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).
```

#### 43.16.3 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.
```

#### 43.16.4 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](#).

### 43.17 KeyedValueDataset

#### 43.17.1 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](#).

#### 43.17.2 Methods

This interface extends the [KeyedValue](#) and [Dataset](#) interfaces, and adds no additional methods.

#### 43.17.3 Notes

There are currently no charts that specifically require this type of dataset.

### 43.18 KeyedValuesDataset

#### 43.18.1 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.

#### 43.18.2 Methods

This interface adds no methods to those it inherits from the [KeyedValues](#) and [Dataset](#) interfaces.

## 43.19 KeyedValues2DDataset

### 43.19.1 Overview

Equivalent to the [CategoryDataset](#) interface.

## 43.20 PieDataset

### 43.20.1 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.

### 43.20.2 Methods

This interface adds no methods to those it inherits from the [KeyedValues](#) and [Dataset](#) interfaces.

### 43.20.3 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](#).

## 43.21 Series

### 43.21.1 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.

### 43.21.2 Constructor

The constructor is [protected](#) since you do not create a [Series](#) directly, but via a subclass:

```
protected Series(String name, String description);  
Creates a new series.
```

### 43.21.3 Methods

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.
```

To deregister a change listener:

```
public void removeChangeListener(SeriesChangeListener listener);  
Deregisters the listener.
```

If you have a lot of changes to make to a series, sometimes it can be a problem that *every* change generates a `SeriesChangeEvent` which is sent to all listeners. You can temporarily disable the event notification using:

```
public void setNotify(boolean notify);  
Turns the event notification on or off. When you turn this off then on again, a change event is sent immediately.
```

#### See Also

[AbstractSeriesDataset](#), [TimeSeries](#), [XYSeries](#).

## 43.22 SeriesChangeEvent

### 43.22.1 Overview

An event class that is passed to a `SeriesChangeListener` to notify it concerning a change to a `Series`.

## 43.23 SeriesChangeListener

### 43.23.1 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`).

### 43.23.2 Methods

This interface defines a single method:

```
public void seriesChanged(SeriesChangeEvent event);  
Receives notification when a series changes.
```

### 43.23.3 Notes

The `AbstractSeriesDataset` class implements this interface—it will generate a `DatasetChangeEvent` every time it receives notification of a `SeriesChangeEvent`.

## 43.24 SeriesDataset

### 43.24.1 Overview

A base interface that defines a dataset containing zero, one or many data series.

### 43.24.2 Methods

To find out how many series there are in a dataset:

```
public int getSeriesCount();  
Returns the number of series in the dataset.
```

To get the name of a series:

```
public String getSeriesName(int series);  
Returns the name of the series with the specified index (zero based).
```

### 43.24.3 Notes

This interface is extended by [CategoryDataset](#) and [XYDataset](#).

## 43.25 SeriesException

### 43.25.1 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](#).

## 43.26 SubSeriesDataset

A specialised dataset implementation written by Bill Kelemen. To be documented.

## 43.27 ValueDataset

### 43.27.1 Overview

A *value dataset* stores a single value ([Number](#) object).

### 43.27.2 Methods

This interface extends the [Value](#) and [Dataset](#) interfaces, and adds no new methods.

### 43.27.3 Notes

This dataset is used by the [ThermometerPlot](#) class.

## 43.28 WaferMapDataset

### 43.28.1 Overview

A dataset that can be used with the [WaferMapPlot](#) class.

# Chapter 44

## Package: org.jfree.data.jdbc

### 44.1 Introduction

This package contains interfaces and classes for the datasets used by JFreeChart.

### 44.2 JDBC`CategoryDataset`

#### 44.2.1 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.

#### 44.2.2 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.
```

### 44.2.3 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](#).

## 44.3 JDBC PieDataset

### 44.3.1 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.

### 44.3.2 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.
```

### 44.3.3 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](#).

## 44.4 JDBCXYDataset

### 44.4.1 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.

### 44.4.2 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.

### 44.4.3 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.

#### 44.4.4 Notes

There is a demo application `JDBCXYChartDemo` in the JFreeChart Premium Demo distribution that illustrates the use of this class.

##### See Also

[XYDataset](#).

# Chapter 45

## Package: `org.jfree.data.statistics`

### 45.1 Introduction

This package contains interfaces and classes for representing statistical datasets.

### 45.2 BoxAndWhiskerCalculator

#### 45.2.1 Overview

A utility class for calculating the statistics required for a box-and-whisker plot.

#### 45.2.2 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.
```

To calculate the mean of a list of values:

```
public static double calculateMean(List values)  
Returns the mean of a list of numbers. Items in the list that are not  
instances of the Number class are ignored. Likewise, null items are ignored.
```

To calculate the median of a list of values:

```
public static double calculateMedian(List values);  
Returns the median of a list of values. This method REQUIRES the list  
of values to be in ascending order.
```

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.
```

## 45.3 BoxAndWhiskerCategoryDataset

### 45.3.1 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.

### 45.3.2 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.
```

## 45.4 BoxAndWhiskerItem

### 45.4.1 Overview

A small class that holds the statistics and values required for a box-and-whisker item:

- a mean;
- a median;
- a first quartile boundary value;
- a third quartile boundary value;
- a minimum regular value;
- a maximum regular value;
- a minimum outlier;
- a maximum outlier;
- a list of outlier values;

This class is immutable.

#### 45.4.2 Notes

The `BoxAndWhiskerCalculator` class returns instances of this class from one of its methods.

### 45.5 BoxAndWhiskerXYDataset

#### 45.5.1 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.

#### 45.5.2 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 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).
```

## 45.6 DefaultBoxAndWhiskerCategoryDataset

### 45.6.1 Overview

A basic implementation of the [BoxAndWhiskerCategoryDataset](#) interface.

### 45.6.2 Methods

To add an item to the dataset:

```
public void add(final BoxAndWhiskerItem item, final Comparable rowKey,
final Comparable columnKey);
>Adds an item to the dataset using the specified row and column keys (the
row corresponds to the series and the column corresponds to the category).
```

For convenience, you can create a new item from a list of raw data values:

```
public void add(final List list, final Comparable rowKey, final Comparable
columnKey);
>Adds an item to the dataset that summarises the raw data in the list.
```

### 45.6.3 Notes

There is a demo (`BoxAndWhiskerDemo1.java`) included in the JFreeChart Premium Demo distribution.

## 45.7 DefaultBoxAndWhiskerXYDataset

### 45.7.1 Overview

A basic implementation of the [BoxAndWhiskerXYDataset](#) interface.

### 45.7.2 Notes

The `XYBoxAndWhiskerDemo1` (included in the JFreeChart Premium Demo distribution) provides an example of this class being used.

## 45.8 DefaultStatisticalCategoryDataset

### 45.8.1 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.

### 45.8.2 Constructors

This class has just one constructor:

```
public DefaultStatisticalCategoryDataset();
Creates a new instance containing no data.
```

### 45.8.3 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.
```

### 45.8.4 Accessing Data

To access the values in the dataset:

```
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.
```

```
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.
```

```
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.
```

### 45.8.5 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 int getColumnIndex(Comparable key);
```

```
public Comparable getColumnKey(int column);
```

```

public List getColumnKeys();

public int getRowIndex(Comparable key);

public Comparable getRowKey(int row);

public List getRowKeys();

```

#### 45.8.6 Other Methods

```

public Range getRangeBounds(boolean includeInterval);

public double getRangeLowerBound(boolean includeInterval);

public double getRangeUpperBound(boolean includeInterval);

```

### 45.9 HistogramBin

#### 45.9.1 Overview

This class is used to represent a bin for the [HistogramDataset](#) class.

### 45.10 HistogramDataset

#### 45.10.1 Overview

A dataset that can be used with the [XYPlot](#) class to display a histogram.

#### 45.10.2 Constructors

The default constructor creates an empty dataset:

```

public HistogramDataset();
Creates an empty dataset with a type of HistogramType.FREQUENCY.

```

#### 45.10.3 Methods

To set the type of histogram:

```

public void setType(HistogramType type);
Sets the histogram type and sends a DatasetChangeEvent to all registered
listeners.

```

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 the 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).
```

For both of the above methods, values that fall on a bin boundary will be allocated to the *lower* bin (except in the case of the *minimum* value which is assigned to the first bin).

#### 45.10.4 Notes

Some points to note:

- the dataset is `Cloneable` and `Serializable`;
- a demo (`HistogramDemo1.java`) is included in the JFreeChart Premium Demo distribution.

### 45.11 HistogramType

#### 45.11.1 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.

#### 45.11.2 Usage

These values are normally used in the `getType()` and `setType()` methods of the `HistogramDataset` class.

### 45.12 MeanAndStandardDeviation

#### 45.12.1 Overview

A simple class that records the mean and standard deviation for some data. The base data is not known to this class, so the mean and standard deviation values have to be calculated by external code.

### 45.12.2 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).

### 45.12.3 Methods

To access the mean value:

```
public Number getMean();
```

Returns the mean, which may be `null`.

```
public Number getStandardDeviation();
```

Returns the standard deviation, which may be `null`.

```
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.

### 45.12.4 Notes

This class is used in the `DefaultStatisticalCategoryDataset` implementation.

## 45.13 Regression

### 45.13.1 Overview

This class provides some utility methods for calculating regression co-efficients. Two regression types are supported:

- linear (OLS) regression;
- power regression.

### 45.13.2 Methods

To calculate the OLS regression for an array of data values:

```
public static double[] getOLSRegression(double[][] data);
```

Performs an ordinary least squares regression on the data. The result is an array containing two values, the intercept and the slope.

To calculate a power regression for an array of data values:

```
public static double[] getPowerRegression(double[][] data);
```

Performs a power regression on the data.

## 45.14 StatisticalCategoryDataset

### 45.14.1 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.

### 45.14.2 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 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 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 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 for one cell in the table.

### 45.14.3 Notes

The [DefaultStatisticalCategoryDataset](#) class implements this interface.

## 45.15 Statistics

### 45.15.1 Overview

Provides some static utility methods for calculating statistics.

### 45.15.2 Methods

To calculate the average of an array of `Number` objects:

```
public static double getAverage(Number[] data);
```

Returns the average of an array of numbers.

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.

To calculate a least squares regression line through an array of data:

```
public static double[] getLinearFit(Number[] x_data, Number[] y_data);
```

Returns the intercept (`double[0]`) and slope (`double[1]`) of the linear regression line.

To calculate the slope of a least squares regression line:

```
public static double getSlope(Number[] x_data, Number[] y_data);
```

Returns the slope of the linear regression line.

To calculate the slope of a least squares regression line:

```
public static double getCorrelation(Number[] data1, Number[] data2);
```

Returns the correlation between two sets of numbers.

### 45.15.3 Notes

This class was written by Matthew Wright.

# Chapter 46

## Package: org.jfree.data.time

### 46.1 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.

### 46.2 DateRange

#### 46.2.1 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`.

#### 46.2.2 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).
```

#### 46.2.3 Notes

Instances of this class are immutable and `Serializable`.

## 46.3 Day

### 46.3.1 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`.

### 46.3.2 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.

### 46.3.3 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.
```

### 46.3.4 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.
```

### 46.3.5 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.

## 46.4 DynamicTimeSeriesCollection

### 46.4.1 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.

### 46.4.2 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>

#### 46.4.3 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.

```
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:

---

<sup>1</sup>It would probably make sense to refactor the class so that the x-values are initialised in the constructor.

```
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.
```

#### 46.4.4 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).

## 46.5 FixedMillisecond

### 46.5.1 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`.

### 46.5.2 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 a `FixedMillisecond` instance based on a `java.util.Date` instance:

```
public FixedMillisecond(Date time);
Creates a new FixedMillisecond instance representing the same millisec-
ond 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.
```

### 46.5.3 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.
```

### 46.5.4 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.

## 46.6 Hour

### 46.6.1 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`.

### 46.6.2 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.

### 46.6.3 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.
```

### 46.6.4 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.
```

### 46.6.5 Notes

The `Hour` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

## 46.7 Millisecond

### 46.7.1 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`.

### 46.7.2 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.
```

### 46.7.3 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.
```

#### 46.7.4 Notes

The `Millisecond` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

### 46.8 Minute

#### 46.8.1 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.

#### 46.8.2 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.
```

#### 46.8.3 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.
```

#### 46.8.4 Notes

The `Minute` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

## 46.9 Month

### 46.9.1 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`.

### 46.9.2 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.
```

### 46.9.3 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.
```

#### 46.9.4 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.

### 46.10 MovingAverage

#### 46.10.1 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.

#### 46.10.2 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 46.1.

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 46.1: 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.

### 46.10.3 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.
```

### 46.10.4 Notes

The `MovingAverageDemo1` class in the JFreeChart Premium Demo distribution provides one example of how to use this class.

## 46.11 Quarter

### 46.11.1 Overview

A calendar quarter—this class extends `RegularTimePeriod`.

### 46.11.2 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);
```

### 46.11.3 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.
```

### 46.11.4 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.
```

### 46.11.5 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.

## 46.12 RegularTimePeriod

### 46.12.1 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`.

### 46.12.2 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>

### 46.12.3 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.

### 46.12.4 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.
```

---

<sup>2</sup>For example, an accountant might be adding up sales for all the subsidiaries of a multi-national 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 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.

```
public abstract long getLastMillisecond(Calendar calendar);
```

Returns the last millisecond of the time period, evaluated using the supplied calendar (which incorporates a timezone).

### 46.12.5 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`.

## 46.13 Second

### 46.13.1 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`.

### 46.13.2 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.
```

### 46.13.3 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.
```

### 46.13.4 Notes

The `Second` class is immutable. This is a requirement for all `RegularTimePeriod` subclasses.

## 46.14 SimpleTimePeriod

### 46.14.1 Overview

This class represents a fixed period of time with millisecond precision (implements the `TimePeriod` interface).

### 46.14.2 Constructor

To create a new instance:

```
public SimpleTimePeriod(Date start, Date end);  
Creates a new time period with the specified start and end.
```

### 46.14.3 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.
```

### 46.14.4 Notes

Some points to note:

- instances of this class are immutable;
- implements the `Serializable` interface;

## 46.15 TimePeriod

### 46.15.1 Overview

A period of time defined by two `java.util.Date` instances representing the start and end of the time period.

### 46.15.2 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.
```

### 46.15.3 Notes

This interface is implemented by:

- the `SimpleTimePeriod` class;
- the `RegularTimePeriod` base class and all its subclasses.

## 46.16 TimePeriodAnchor

### 46.16.1 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.

## 46.17 TimePeriodFormatException

### 46.17.1 Overview

An exception that can be thrown by the methods used to convert time periods to strings, and vice versa.

## 46.18 TimePeriodValue

### 46.18.1 Overview

An object that represents a time period with an associated value, used to represent each item in a `TimePeriodValues` collection.

### 46.18.2 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.
```

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.
```

### 46.18.3 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).

## 46.19 TimePeriodValues

### 46.19.1 Overview

A collection of `TimePeriodValue` objects. The objects are maintained in the order they are added. This class is used to represent one data series in a `TimePeriodValuesCollection`.

## 46.20 TimePeriodValuesCollection

### 46.20.1 Overview

A collection of `TimePeriodValues` objects.

### 46.20.2 Usage

The `TimePeriodValuesDemo1` application, included in the JFreeChart Premium Demo distribution, provides an example of how to use this class.

### 46.20.3 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.
```

### 46.20.4 Methods

To add a new series to the collection:

```
public void addSeries(TimePeriodValues series);  
Adds a series to the collection. A DatasetChangeEvent is sent to all  
registered listeners.
```

### 46.20.5 Notes

This class implements the `DomainInfo` interface.

## 46.21 TimeSeries

### 46.21.1 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.

### 46.21.2 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 Premium Demo distribution.

### 46.21.3 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).
```

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.
```

#### 46.21.4 Attributes

Each instance of `TimeSeries` has the following attributes:

Attribute:	Description:
<code>name</code>	The name of the series (inherited from <code>Series</code> ).
<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.

#### 46.21.5 Methods

To find out how many data items are in a series:

```
public int getItemCount()
Returns the number of data items in the series.
```

To retrieve a particular value from a series by the index of the item:

```
public TimeSeriesDataItem getDataItem(int item)
Returns a data item. The item argument is a zero-based index.
```

To retrieve a particular value from a series by time period:

```
public TimeSeriesDataItem getDataItem(linkRegularTimePeriod period)
Returns the data item (if any) for the specified time period.
```

To add a value to a time series:

```
public void add(RegularTimePeriod period, Number value)
throws SeriesException;
Adds a new value (null permitted) to the time series. Throws an exception
if the time period is not unique within the series.
```

You can create a time series that automatically discards “old” data. This is done by specifying a `historyCount` attribute:

```
public void setHistoryCount(int count);
Sets the historyCount attribute, which is the number of time periods in the
“history” for the time series. When a new data value is added, any data
that is more than historyCount periods old is automatically discarded.
```

#### 46.21.6 Notes

You can calculate the moving average of a time series using the `MovingAverage` utility class.

#### See Also

[TimePeriod](#), [TimeSeriesCollection](#).

## 46.22 TimeSeriesCollection

### 46.22.1 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).

### 46.22.2 Usage

A demo (`TimeSeriesDemo.java`) is included in the premium demo collection.

### 46.22.3 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.

### 46.22.4 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.
```

#### 46.22.5 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.
```

#### 46.22.6 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.

#### 46.22.7 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.

### 46.22.8 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.

### 46.22.9 Notes

Points to note:

- this class extends `AbstractSeriesDataset` to provide some of the basic series information.
- this class implements the `XYDataset` and `IntervalXYDataset` interfaces.

## 46.23 TimeSeriesDataItem

### 46.23.1 Overview

This class associates a `Number` with a `RegularTimePeriod`, and is used by the `TimeSeries` class to record individual data items.

### 46.23.2 Usage

You won't normally use this class directly—the `TimeSeries` class will create instances as required.

### 46.23.3 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.
```

#### 46.23.4 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).
```

#### 46.23.5 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.

### 46.24 TimeSeriesTableModel

An initial attempt to display a time series in a `JTable`.

### 46.25 TimeTableXYDataset

#### 46.25.1 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.

#### 46.25.2 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.
```

### 46.25.3 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.
```

### 46.25.4 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.

```

#### See Also:

[StackedXYAreaRenderer](#) and [StackedXYBarRenderer](#).

## 46.26 Week

### 46.26.1 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.

### 46.26.2 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.
```

### 46.26.3 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.
```

#### 46.26.4 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](#).

### 46.27 Year

#### 46.27.1 Overview

A class that represents a calendar year (for example, “2003”). This class extends `RegularTimePeriod`.

#### 46.27.2 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));
```

#### 46.27.3 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.

#### 46.27.4 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.

#### 46.27.5 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 47

## Package: org.jfree.data.xml

### 47.1 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.

### 47.2 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 Premium Demo distribution:

- `XMLBarChartDemo.java`
- `XMLPieChartDemo.java`

### 47.3 CategoryDatasetHandler

#### 47.3.1 Overview

A SAX handler that creates a `CategoryDataset` by processing the elements in an XML document.

#### 47.3.2 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 Premium Demo distribution.

#### 47.3.3 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. --&gt;
&lt;CategoryDataset&gt;

&lt;Series name = "Series 1"&gt;
&lt;Item&gt;
&lt;Key&gt;Category 1&lt;/Key&gt;
&lt;Value&gt;15.4&lt;/Value&gt;
&lt;/Item&gt;
&lt;Item&gt;
&lt;Key&gt;Category 2&lt;/Key&gt;
&lt;Value&gt;12.7&lt;/Value&gt;
&lt;/Item&gt;
&lt;Item&gt;
&lt;Key&gt;Category 3&lt;/Key&gt;
&lt;Value&gt;5.7&lt;/Value&gt;
&lt;/Item&gt;
&lt;Item&gt;
&lt;Key&gt;Category 4&lt;/Key&gt;
&lt;Value&gt;9.1&lt;/Value&gt;
&lt;/Item&gt;
&lt;/Series&gt;

&lt;Series name = "Series 2"&gt;
&lt;Item&gt;
&lt;Key&gt;Category 1&lt;/Key&gt;
&lt;Value&gt;45.4&lt;/Value&gt;
&lt;/Item&gt;
&lt;Item&gt;
&lt;Key&gt;Category 2&lt;/Key&gt;
&lt;Value&gt;73.7&lt;/Value&gt;
&lt;/Item&gt;
&lt;Item&gt;
&lt;Key&gt;Category 3&lt;/Key&gt;
&lt;Value&gt;23.7&lt;/Value&gt;
&lt;/Item&gt;
&lt;Item&gt;
&lt;Key&gt;Category 4&lt;/Key&gt;
&lt;Value&gt;19.4&lt;/Value&gt;
&lt;/Item&gt;
&lt;/Series&gt;

&lt;/CategoryDataset&gt;
</pre>

```

The `<CategoryDataset>` element can contain any number of `<Series>` elements, and each `<Series>` element can contain any number of `<Item>` elements.

#### 47.3.4 Notes

This class delegates work to the [CategorySeriesHandler](#) class.

### 47.4 CategorySeriesHandler

#### 47.4.1 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.

## 47.5 DatasetReader

### 47.5.1 Overview

This class contains utility methods for reading datasets from XML files. In the current release, support is included for [PieDataset](#) and [CategoryDataset](#).

### 47.5.2 Usage

Two applications ([XMLPieChartDemo](#) and [XMLBarChartDemo](#)) that demonstrate how to use this class are included in the JFreeChart Premium Demo distribution.

## 47.6 DatasetTags

### 47.6.1 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 47.1: Attributes for the *DatasetTags* interface

## 47.7 ItemHandler

### 47.7.1 Overview

A SAX handler that reads a *key/value* pair.

### 47.7.2 Usage

You should not need to use this class directly. Work is delegated to this handler by the [PieDatasetHandler](#) class.

### 47.7.3 Notes

This class delegates some work to the [KeyHandler](#) class.

## 47.8 KeyHandler

### 47.8.1 Overview

A SAX handler that reads a *key* element from an XML file.

### 47.8.2 Usage

You should not need to use this class directly. Work is delegated to this class by the [ItemHandler](#) class.

### 47.8.3 Notes

A key can be any instance of [Comparable](#), but the handler always uses the [String](#) class to represent keys.

## 47.9 PieDatasetHandler

### 47.9.1 Overview

A SAX handler for reading a [PieDataset](#) from an XML file.

### 47.9.2 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 Premium Demo distribution.

### 47.9.3 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.

### 47.9.4 Notes

This class delegates some work to the [ItemHandler](#) class.

## 47.10 RootHandler

### 47.10.1 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.

## 47.11 ValueHandler

### 47.11.1 Overview

A SAX handler that processes numerical values.

# Chapter 48

## Package: org.jfree.data.xy

### 48.1 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.

### 48.2 AbstractIntervalXYDataset

#### 48.2.1 Overview

A base class that can be used to implement an `IntervalXYDataset` (extends `AbstractXYDataset`).

#### 48.2.2 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.

### 48.3 AbstractXYDataset

#### 48.3.1 Overview

A base class that can be used to implement an `XYDataset`.

### 48.3.2 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.

## 48.4 AbstractXYZDataset

### 48.4.1 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.

### 48.4.2 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.
```

## 48.5 CategoryTableXYDataset

### 48.5.1 Overview

A dataset that implements the [TableXYDataset](#) interface, so that it can be used with the [StackedXYAreaRenderer](#) and [StackedXYBarRenderer](#) classes.

### 48.5.2 Constructor

To create a new dataset:

```
public CategoryTableXYDataset();
Creates a new dataset.
```

### 48.5.3 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.
```

#### 48.5.4 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).
```

#### 48.5.5 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.

```

#### 48.5.6 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.

```

### 48.6 DefaultHighLowDataset

#### 48.6.1 Overview

A default implementation of the `OHLCDataset` interface. There is some duplication between this class and the `DefaultOHLCDataset` interface.

### 48.7 DefaultOHLCDataset

#### 48.7.1 Overview

A simple implementation of the `OHLCDataset` interface that supports only one series. There are no methods to support updating the dataset at present.

### 48.7.2 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).

### 48.7.3 Methods

To get the series name:

```
public String getSeriesName(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, since this dataset supports only one series.

```
public Date getXDate(int series, int item);
```

Returns the x-value for the specified item as a `Date`. The `series` argument is ignored, since this dataset supports only one series.

```
public Number getY(int series, int item);
```

Returns the closing price for the specified item. This method is required by the `XYDataset` interface.

```
public Number getHigh(int series, int item);
```

Returns the high value for the specified item. The `series` argument is ignored, since this dataset supports only one series.

```
public double getHighValue(int series, int item);
```

Returns the high value for the specified item as a `double`.

```
public Number getLow(int series, int item);
```

Returns the low value for the specified item. The `series` argument is ignored, since this dataset supports only one series.

```
public double getLowValue(int series, int item);
```

Returns the low value for the specified item as a `double`.

```
public Number getOpen(int series, int item);
```

Returns the open value for the specified item. The `series` argument is ignored, since this dataset supports only one series.

```
public double getOpenValue(int series, int item);
```

Returns the open value for the specified item as a `double`.

```
public Number getClose(int series, int item);
```

Returns the close value for the specified item. The `series` argument is ignored, since this dataset supports only one series.

```
public double getCloseValue(int series, int item);
```

Returns the close value for the specified item as a `double`.

```
public Number getVolume(int series, int item);
Returns the volume value for the specified item. The series argument is
ignored, since this dataset supports only one series.

public double getVolumeValue(int series, int item);
Returns the volume value for the specified item as a double.
```

To sort the data items into date order:

```
public void sortDataByDate();
Sorts the array of data items into ascending order by date.
```

#### See Also

[OHLCDataSet](#).

## 48.8 DefaultTableXYDataset

### 48.8.1 Overview

An implementation of the `XYDataset` interface where all series share a common set of x-values. This dataset can be used to create stacked area charts.

### 48.8.2 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.
```

### 48.8.3 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 String getSeriesName(int series);
Returns the name of 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).
```

#### 48.8.4 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.
```

#### 48.8.5 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 regis-
tered 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.
```

#### 48.8.6 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.
```

#### 48.8.7 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.
```

### 48.9 DefaultWindDataset

#### 48.9.1 Overview

A default implementation of the `WindDataset` interface.

## 48.10 IntervalXYDataset

### 48.10.1 Overview

A dataset that returns an interval for each of the x and y dimensions. Extends the [XYDataset](#) interface.

### 48.10.2 Methods

To get the start value of the x-interval:

```
public Number getStartX(int series, int item);  
Returns the start value of the x-interval for an item within a series.  
  
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 Number getEndX(int series, int item);  
Returns the end value of the x-interval for an item within a series.  
  
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 Number getStartY(int series, int item);  
Returns the start value of the y-interval for an item within a series.  
  
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 Number getEndY(int series, int item);  
Returns the end value of the y-interval for an item within a series.  
  
public double getEndYValue(int series, int item);  
Returns the end value of the y-interval for an item within a series.
```

### 48.10.3 Notes

The [TimeSeriesCollection](#) class implements this interface.

#### See Also:

[XYDataset](#), [IntervalXYZDataset](#).

## 48.11 IntervalXYDelegate

### 48.11.1 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 [IntervalXYDataset](#).

### 48.11.2 Usage

This class is used internally by the JFreeChart Class Library. In general, you won't need to use this class directly.

### 48.11.3 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.
```

### 48.11.4 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.
```

#### 48.11.5 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.
```

#### 48.11.6 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.
```

#### 48.11.7 Notes

The class is used by the `CategoryTableXYDataset`, `DefaultTableXYDataset`, and `XYSeriesCollection` classes.

### 48.12 IntervalXYZDataset

#### 48.12.1 Overview

An extension of the `XYZDataset` interface, analogous to the `IntervalXYDataset` extension of the `XYDataset` interface.

#### 48.12.2 Notes

There are no classes that implement this interface at present.

## 48.13 MatrixSeries

### 48.13.1 Overview

To be documented.

## 48.14 MatrixSeriesCollection

### 48.14.1 Overview

To be documented.

## 48.15 NormalizedMatrixSeries

### 48.15.1 Overview

Not yet documented.

## 48.16 OHLCDataItem

### 48.16.1 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.

### 48.16.2 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.

### 48.16.3 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](#).

## 48.17 OHLCDataset

### 48.17.1 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.

### 48.17.2 Methods

To get the *high* value:

```
public Number getHighValue(int series, int item);
Returns the high value for an item within a series.
```

To get the *low* value:

```
public Number getLowValue(int series, int item);
Returns the low value for an item within a series.
```

To get the *open* value:

```
public Number getOpenValue(int series, int item);  
Returns the open value for an item within a series.
```

To get the *close* value:

```
public Number getCloseValue(int series, int item);  
Returns the close value for an item within a series.
```

To get the *volume*:

```
public Number getVolumeValue(int series, int item);  
Returns the volume value for an item within a series.
```

### 48.17.3 Notes

This dataset is implemented by the [DefaultOHLCDataset](#) class, and used by the [CandlestickRenderer](#) class.

#### See Also

[XYDataset](#), [DefaultOHLCDataset](#).

## 48.18 SignalsDataset

### 48.18.1 Overview

Not yet documented.

## 48.19 TableXYDataset

### 48.19.1 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](#)).

## 48.20 WindDataset

### 48.20.1 Overview

A *wind dataset* provides wind direction and intensity values observed at various points in time.

### 48.20.2 Notes

The [WindChartDemo1](#) application, included in the JFreeChart Premium Demo distribution, provides an example.

## 48.21 XisSymbolic

### 48.21.1 Overview

An interface that can be implemented by an `XYDataset` in order to link the (integer) x-values with symbols.

### 48.21.2 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.
```

### 48.21.3 Notes

None of the standard datasets implement this interface.

## 48.22 XYBarDataset

### 48.22.1 Overview

A dataset wrapper class that can convert any `XYDataset` into an `IntervalXYDataset`.

### 48.22.2 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.
```

## 48.23 XYDataItem

### 48.23.1 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.

### 48.23.2 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.
```

### 48.23.3 Methods

To access the x and y values:

```
public Number getX();
Returns the x-value (never null).

public Number getY();
Returns the y-value (possibly null).
```

To set the y-value:

```
public void setY(Number y);
Sets the y-value. Note that there is no corresponding method to set the
x-value.
```

### 48.23.4 Notes

Some notes:

- this class implements the `Comparable` interface, and implements ordering by x-values.
- this class parallels the `TimeSeriesDataItem` class.

## 48.24 XYDataset

### 48.24.1 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. 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` and `TimeSeriesCollection`. Extensions of this interface include: `IntervalXYDataset`, `HighLowDataset`, `XYZDataset` and `TableXYDataset`.

### 48.24.2 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`.

### 48.24.3 Methods

To get the number of items in a series:

```
public int getItemCount(int series);
>Returns the number of data items in a series.
```

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 (never null).
```

```
public double getXValue(int series, int item);
>Returns the x-value for an item within a series.
```

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 (possibly null, which
indicates a missing or unknown value).
```

```
public double getYValue(int series, int item);
>Returns the y-value for am item within a series. If this method re-
turns 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.
```

### 48.24.4 Notes

The interface allows `null` y-values but does not allow `null` x-values, because I couldn't think of a situation where `null` x-values are useful.

#### See Also:

[SeriesDataset](#), [IntervalXYDataset](#).

## 48.25 XYDatasetTableModel

### 48.25.1 Overview

A simple wrapper for an `XYDataset` that creates a read-only implementation of Swing's `TableModel` interface. The current implementation is somewhat broken, in that it doesn't account for the fact that each series in an `XYDataset` can have different x-values (and, in fact, different numbers of items).

### 48.25.2 Constructors

The default constructor creates an empty table model:

```
public XYDatasetTableModel();
Creates an empty table model.
```

To create a new table model:

```
XYDatasetTableModel(XYDataset dataset);
Creates a new table model for the specified dataset.
```

### 48.25.3 Methods

To set the dataset to be presented as a `TableModel`:

```
public void setModel(XYDataset dataset);
Sets the underlying dataset for the table model.
```

To get the row and column counts:

```
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.

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 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.
```

The table model is “read only”:

```
public boolean isCellEditable(int row, int column);
Returns false.

public void setValueAt(Object value, int row, int column);
Does nothing, since there is no general way to update the underlying dataset.
```

### 48.25.4 Notes

This class needs fixing.

## 48.26 XYSeries

### 48.26.1 Overview

A series of  $(x, y)$  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.

### 48.26.2 Constructors

To construct a series:

```
public XYSeries(String name);
```

Creates a new series (initially empty) with the specified name. By default, the data items will be sorted in ascending order of x-values, and duplicate x-values will be allowed.

To construct a series with control over sorting and whether or not duplicate x-values are permitted:

```
public XYSeries(String name,
    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 where duplicate x-values will be allowed or disallowed, as specified.

### 48.26.3 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.

### 48.26.4 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.
```

#### 48.26.5 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.
```

### 48.26.6 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.

### 48.26.7 Notes

Some points to note:

- this class extends `Series`, so you can register change listeners with the series;

## 48.27 XYSeriesCollection

### 48.27.1 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.

### 48.27.2 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 (more can be added).
```

### 48.27.3 Usage

A demo (`XYSeriesDemo.java`) is included in the premium demo collection.

### 48.27.4 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.
```

### 48.27.5 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).
```

### 48.27.6 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.

To get the name of a series:

```
public String getSeriesName(int series);
```

Returns the name of 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 x-value for an item within a series:

```
public Number getXValue(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 getStartXValue(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 getEndXValue(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 getYValue(int series, int item);
```

Returns the value of the specified item.

### 48.27.7 Notes

Some points to note:

- if the x-values in your dataset are time or date based, consider using the `TimeSeriesCollection` class instead.

## 48.28 XYZDataset

### 48.28.1 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.

### 48.28.2 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.

### 48.28.3 Notes

Some points to note:

- Known subclasses include `DefaultContourDataset` and `MatrixSeriesCollection`;
- JFreeChart doesn’t have support for three dimensional charts yet, but this interface still finds a use in the `XYBubbleRenderer` class.

## 48.29 YisSymbolic

### 48.29.1 Overview

An interface that can be implemented by an `XYDataset` in order to link the (integer) y-values with symbols.

### 48.29.2 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.

### 48.29.3 Notes

None of the standard datasets implement this interface.

# Appendix A

## JCommon

### A.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/index.html>

Selected JCommon classes are documented here because they are used extensively within JFreeChart.

### A.2 Align

#### A.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 A.1: Alignment codes*

#### A.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.
```

## A.3 PublicCloneable

### A.3.1 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.

### A.3.2 Methods

This interface declares a single method:

```
public Object clone() throws CloneNotSupportedException;
Creates a clone of the object.
```

## A.4 RectangleAnchor

### A.4.1 Overview

This class defines an enumeration of nine common anchor points within a rectangle. These points include the four corners of the rectangle, the four mid-points of each rectangle edge, and the center point:

ID:	Description:
<code>RectangleAnchor.TOP</code>	The midpoint of the rectangle's top edge.
<code>RectangleAnchor.BOTTOM</code>	The midpoint of the rectangle's bottom edge.
<code>RectangleAnchor.LEFT</code>	The midpoint of the rectangle's left edge.
<code>RectangleAnchor.RIGHT</code>	The midpoint of the rectangle's right edge.
<code>RectangleAnchor.TOP_LEFT</code>	The top-left corner of the rectangle.
<code>RectangleAnchor.TOP_RIGHT</code>	The top-right corner of the rectangle.
<code>RectangleAnchor.BOTTOM_LEFT</code>	The bottom-left corner of the rectangle.
<code>RectangleAnchor.BOTTOM_RIGHT</code>	The bottom-right corner of the rectangle.
<code>RectangleAnchor.CENTER</code>	The center of the rectangle.

Table A.2: Constants defined by `RectangleAnchor`

## A.5 RectangleEdge

### A.5.1 Overview

This class defines an enumeration of the four edges of a rectangle. It is used to specify the location of objects (for example, axes in a plot) relative to a rectangle:

ID:	Description:
RectangleEdge.TOP	The top edge.
RectangleEdge.BOTTOM	The bottom edge.
RectangleEdge.LEFT	The left edge.
RectangleEdge.RIGHT	The right edge.

Table A.3: Constants defined by *RectangleEdge*

## A.6 RectangleInsets

### A.6.1 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).

### A.6.2 Constructor

To create a new instance:

```
public RectangleInsets(double top, double bottom,
                      double left, double right);
Creates a new instance with the given insets as absolute units.

public RectangleInsets(final UnitType unitType,
                      final double top, final double bottom,
                      final double left, final 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.
```

### A.6.3 Methods

```
public UnitType getUnitType();
Returns the unit type (relative or absolute) for the insets.

public double getTop();

public double getBottom();

public double getLeft();

public double getRight();

public boolean equals(final Object obj);

public int hashCode();

public Rectangle2D createAdjustedRectangle(final Rectangle2D base, final
LengthAdjustmentType horizontal, final LengthAdjustmentType vertical);
```

```

public Rectangle2D createInsetRectangle(final Rectangle2D base);

public Rectangle2D createInsetRectangle(final Rectangle2D base, final boolean
horizontal, final boolean vertical);

public Rectangle2D createOutsetRectangle(final Rectangle2D base);

public Rectangle2D createOutsetRectangle(final Rectangle2D base, final
boolean horizontal, final boolean vertical);

public double calculateTopInset(final double height);

public double calculateTopOutset(final double height);

public double calculateBottomInset(final double height);

public double calculateBottomOutset(final double height);

public double calculateLeftInset(final double width);

public double calculateLeftOutset(final double width);

public double calculateRightInset(final double width);

public double calculateRightOutset(final double width);

public double trimWidth(double width);

public double extendWidth(double width);

public double trimHeight(double height);

public double extendHeight(double height);

public void trim(Rectangle2D area);

```

## A.7 Spacer

### A.7.1 Overview

This class is used to specify left, right, top and bottom margins 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).

### A.7.2 Constructor

To create a new `Spacer`:

```
public Spacer(int type, double left, double top, double right,
double left);
```

Creates a new spacer. The `type` can be `ABSOLUTE` or `RELATIVE`. The remaining arguments are interpreted as points (1/72 inch) for absolute spacing, or percentages for relative spacing.

### A.7.3 Methods

To get the amount of spacing for the left side:

```
public double getLeftSpace(double width);
>Returns the amount of spacing for the left side.
```

To get the amount of spacing for the right side:

```
public double getRightSpace(double width);
>Returns the amount of spacing for the right side.
```

In both of the above methods, the `width` argument refers to the width of a rectangle that the space calculation is relative to. It is ignored if the space is specified in absolute terms.

To get the amount of spacing for the top side:

```
public double getTopSpace(double height);
>Returns the amount of spacing for the top side.
```

To get the amount of spacing for the bottom side:

```
public double getBottomSpace(double height);
>Returns the amount of spacing for the top side.
```

In both of the above methods, the `height` argument refers to the height of a rectangle that the space calculation is relative to. It is ignored if the space is specified in absolute terms.

A given rectangle can be “shrunk” by a spacer object:

```
public void trim(Rectangle2D area);
>Reduces the dimensions of the specified area, according to the space settings.
```

### A.7.4 Notes

Throughout JFreeChart, the `Insets` class has been used to specify (absolute) padding information. This class is intended to replace the use of `Insets` to allow both absolute and relative settings.

## A.8 TextAnchor

### A.8.1 Overview

This class defines an enumeration of the anchor points relative to the bounds of a text string (see table A.4). It is used to specify an anchor point for text alignment and rotation.

ID:	Description:
TextAnchor.TOP_LEFT	The top left corner.
TextAnchor.TOP_CENTER	The center point on the top edge.
TextAnchor.TOP_RIGHT	The top right corner.
TextAnchor.CENTER_LEFT	The center point on the left edge.
TextAnchor.CENTER	The center point of the text.
TextAnchor.CENTER_RIGHT	The center point on the right edge.
TextAnchor.HALF_ASCENT_LEFT	The half ascent point on the left edge.
TextAnchor.HALF_ASCENT_CENTER	The center point along the half ascent line.
TextAnchor.HALF_ASCENT_RIGHT	The half ascent point on the right edge.
TextAnchor.BASELINE_LEFT	The baseline point on the left edge.
TextAnchor.BASELINE_CENTER	The center point along the half ascent line.
TextAnchor.BASELINE_RIGHT	The baseline point on the right edge.
TextAnchor.BOTTOM_LEFT	The bottom left corner.
TextAnchor.BOTTOM_CENTER	The center point on the bottom edge.
TextAnchor.BOTTOM_RIGHT	The bottom right corner.

Table A.4: Constants defined by *TextAnchor*

## A.9 UnitType

### A.9.1 Overview

This class defines tokens to indicate “relative” or “absolute” measurement units:

ID:	Description:
UnitType.ABSOLUTE	Absolute units.
UnitType.RELATIVE	Relative units.

Table A.5: Constants defined by *UnitType*

These tokens are used by the [RectangleInsets](#) class.

## Appendix B

# The GNU Lesser General Public License

### B.1 Introduction

JFreeChart is licensed under the terms of the GNU Lesser General Public License (LGPL). The full text of this license is reproduced in this appendix. You should read and understand this license 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 [B.3](#) first).

### B.2 The License

The following license 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!

## B.3 Frequently Asked Questions

### B.3.1 Introduction

Some of the most frequently asked questions about JFreeChart concern the license. I've published this FAQ to help developers understand my choice of license 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.

### B.3.2 Questions and Answers

1. *“Can I incorporate JFreeChart into a proprietary (closed-source) application?”*

Yes, the GNU Lesser General Public License (LGPL) is specifically designed to allow this.

2. *“Do I have to pay a license 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 license, 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 license 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 license (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 license;

- 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 2000-2004 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 License (GPL). It is a much simpler license, 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 license 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 license. 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 license?”*

The license 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 license, such as an "Apache-style" license?"*

Yes, a range of licenses was considered for JFreeChart, but now that the choice has been made there are no plans to change the license in the future.

A publication by Bruce Perens was especially helpful in comparing the available licenses:

<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 license, 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 licenses (for example the Apache Software License) allow open source software to be packaged and redistributed without source code. These licenses 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, 211  
AbstractCategoryItemLabelGenerator, 235  
AbstractCategoryItemRenderer, 305  
AbstractDataset, 394  
AbstractIntervalXYDataset, 462  
AbstractRenderers, 299  
AbstractSeriesDataset, 395  
AbstractXYDataset, 462  
AbstractXYItemLabelGenerator, 236  
AbstractXYItemRenderer, 329  
AbstractXYZDataset, 463  
acknowledgements, 17  
Acrobat PDF, 107  
adding chart change listeners, 155  
Align, 486  
annotations, 162  
    XYPlot, 297  
anti-aliasing, 66  
applets, 121  
AreaRenderer, 308  
AreaRendererEndType, 302  
Arrangement, 213  
Axis, 171  
axis  
    display integers only, 196  
AxisChangeEvent, 226  
AxisChangeListener, 226  
AxisCollection, 175  
AxisLocation, 176  
AxisSpace, 176  
AxisState, 177  
  
background image, 66  
BarRenderer, 308  
BarRenderer3D, 313  
Batik, 117  
Block, 213  
BlockBorder, 214  
BlockContainer, 214  
border, 64  
BorderArrangement, 215  
  
BoxAndWhiskerCalculator, 415  
BoxAndWhiskerCategoryDataset, 416  
BoxAndWhiskerItem, 417  
BoxAndWhiskerRenderer, 314  
BoxAndWhiskerToolTipGenerator, 238  
BoxAndWhiskerXYDataset, 418  
BoxAndWhiskerXYToolTipGenerator, 238  
bubble charts, 342  
  
CandlestickRenderer, 331  
catcode.com, 150  
category axis  
    margins, 178  
CategoryAnchor, 177  
CategoryAnnotation, 162  
CategoryAxis, 178  
CategoryAxis3D, 182  
CategoryDataset, 378  
CategoryItemEntity, 221  
CategoryItemRenderer, 315  
CategoryItemRendererState, 318  
CategoryLabelGenerator, 238  
CategoryLabelPosition, 182  
CategoryLabelPositions, 183  
CategoryLabelWidthType, 184  
CategoryPlot, 259  
CategoryStepRenderer, 318  
CategoryTableXYDataset, 463  
CategoryTextAnnotation, 162  
CategoryTick, 184  
CategoryToolTipGenerator, 239  
CategoryToPieDataset, 379  
Cewolf, 138  
chart  
    background color, 65  
    background image, 66  
    border, 64  
    subtitles, 65  
    title, 65  
chart border, 155  
chart change listeners, 155

chart entities, 221  
ChartChangeEvent, 227  
ChartChangeEventType, 228  
ChartChangeListener, 228  
ChartColor, 141  
ChartDeleter, 351  
ChartEntity, 222  
ChartFactory, 141  
ChartFrame, 144  
ChartMouseEvent, 144  
ChartMouseListener, 145  
ChartPanel, 145  
ChartProgressEvent, 228  
ChartProgressListener, 229  
ChartRenderingInfo, 149  
ChartUtilities, 149  
ClipPath, 151  
ClusteredXYBarRenderer, 332  
ColorBar, 184  
ColorBlock, 216  
ColumnArrangement, 216  
CombinationDataset, 396  
combined charts, 96  
CombinedDataset, 396  
CombinedDomainCategoryPlot, 264  
CombinedDomainXYPlot, 265  
CombinedRangeCategoryPlot, 267  
CombinedRangeXYPlot, 268  
comments and suggestions, 17  
CompassFormat, 185  
CompassPlot, 270  
compiling JFreeChart, 35  
ContourDataset, 382  
ContourEntity, 223  
ContourPlot, 270  
ContourPlotUtilities, 270  
ContourToolTipGenerator, 239  
ContourValuePlot, 270  
contributors, 17  
CrosshairState, 270  
CustomXYToolTipGenerator, 239  
CyclicNumberAxis, 185  
CyclicXYItemRenderer, 333  
Dataset, 397  
DatasetChangeEvent, 397  
DatasetChangeListener, 398  
DatasetGroup, 398  
DatasetRenderingOrder, 271  
DatasetUtilities, 398  
DataUtilities, 403  
DateAxis, 186  
DateRange, 426  
DateTick, 191  
DateTickMarkPosition, 191  
DateTickUnit, 191  
Day, 427  
DefaultBoxAndWhiskerCategoryDataset, 419  
DefaultBoxAndWhiskerXYDataset, 419  
DefaultCategoryDataset, 379  
DefaultCategoryItemRenderer, 319  
DefaultContourDataset, 383  
DefaultDrawingSupplier, 272  
DefaultHighLowDataset, 465  
DefaultIntervalCategoryDataset, 380  
DefaultKeyedValue, 367  
DefaultKeyedValueDataset, 403  
DefaultKeyedValues, 368  
DefaultKeyedValues2D, 368  
DefaultKeyedValues2DDataset, 403  
DefaultKeyedValuesDataset, 403  
DefaultOHLCDataset, 465  
DefaultPieDataset, 404  
DefaultPolarItemRenderer, 302  
DefaultStatisticalCategoryDataset, 419  
DefaultTableXYDataset, 467  
DefaultValueDataset, 405  
DefaultWindDataset, 469  
DefaultXYItemRenderer, 333  
demo  
    running, 35  
DialShape, 272  
disabling chart change events, 156  
DisplayChart, 351  
distribution  
    contents, 34  
domain axis, 171  
DomainInfo, 369  
DomainOrder, 369  
download, 33  
DrawableLegendItem, 151  
DrawingSupplier, 272  
dynamic charts, 70  
DynamicTimeSeriesCollection, 428  
Effect3D, 152  
EmptyBlock, 217  
entities, 221  
EntityCollection, 223  
events, 226

exporting charts  
    to JPEG, 150  
    to PDF, 107  
    to PNG, 150  
    to SVG, 117  
`ExtendedCategoryAxis`, 193  
`FastScatterPlot`, 273  
features, 15  
`FixedMillisecond`, 431  
`FlowArrangement`, 217  
Free Software Foundation, 492  
`Function2D`, 384  
`GanttCategoryDataset`, 387  
`GanttRenderer`, 319  
`GridArrangement`, 217  
gridlines  
    `XYPlot`, 297  
`GroupedStackedBarRenderer`, 320  
headless Java, 138  
`HighLowItemLabelGenerator`, 240  
`HighLowRenderer`, 333  
`HistogramBin`, 421  
`HistogramDataset`, 421  
`HistogramType`, 422  
home page, 16  
`Hour`, 432  
HTML image map, 151  
image loading, 138  
image maps, 151  
`ImageMapUtilities`, 232  
images  
    JPEG format, 150  
    PNG format, 150  
`IntervalBarRenderer`, 322  
`IntervalCategoryDataset`, 380  
`IntervalCategoryLabelGenerator`, 241  
`IntervalCategoryToolTipGenerator`, 241  
`IntervalMarker`, 274  
`IntervalXYDataset`, 470  
`IntervalXYDelegate`, 470  
`IntervalXYZDataset`, 472  
item labels, 79  
`ItemLabelAnchor`, 242  
`ItemLabelPosition`, 243  
`iText`, 107  
`Javadoc`, 35  
`JDBCCategoryDataset`, 411  
`JDBCPieDataset`, 412  
`JDBCXYDataset`, 413  
`JFreeChart`, 152  
`JFreeChart`  
    applets, 121  
    license, 492  
    overview and features, 15  
    sample charts, 18  
    servlets, 126  
JPEG, 150  
`JSP`, 138  
`KeyedObject`, 370  
`KeyedObjects`, 370  
`KeyedObjects2D`, 370  
`KeyedValue`, 370  
`KeyedValueComparator`, 370  
`KeyedValueComparatorType`, 370  
`KeyedValueDataset`, 406  
`KeyedValues`, 371  
`KeyedValues2D`, 371  
`KeyedValues2DDataset`, 407  
`KeyedValuesDataset`, 406  
`KeyToGroupMap`, 372  
`LabelBlock`, 218  
`LayeredBarRenderer`, 322  
`Legend`, 157  
`LegendChangeEvent`, 229  
`LegendChangeListener`, 229  
`LegendItem`, 157  
`LegendItemCollection`, 158  
`LegendItemEntity`, 224  
`LegendItemSource`, 159  
`LegendRenderingOrder`, 159  
`LengthConstraintType`, 219  
`LevelRenderer`, 322  
`LGPL`, 492  
license, 492  
    frequently asked questions, 499  
`LineAndShapeRenderer`, 323  
linear regression, 423  
`LineFunction2D`, 385  
`LogarithmicAxis`, 193  
mapping datasets to axes  
    `XYPlot`, 296  
`Marker`, 275  
`MarkerAxisBand`, 193

MatrixSeries, 473  
MatrixSeriesCollection, 473  
MeanAndStandardDeviation, 422  
MeterLegend, 159  
MeterPlot, 275  
Millisecond, 433  
MinMaxCategoryRenderer, 325  
Minute, 434  
ModuloAxis, 193  
Month, 435  
MovingAverage, 436  
MultiplePiePlot, 277  
  
NonGridContourDataset, 383  
NormalDistributionFunction2D, 385  
NormalizedMatrixSeries, 473  
NotOutlierException, 302  
NumberAxis, 195  
NumberAxis  
    display integers only, 196  
NumberAxis3D, 198  
NumberTick, 198  
NumberTickUnit, 198  
  
OHLCDataItem, 473  
OHLCDataSet, 474  
Outlier, 303  
OutlierList, 303  
OutlierListCollection, 303  
  
PeriodAxis, 199  
PeriodAxisLabel, 202  
PieDataset, 407  
PieLabelDistributor, 278  
PieLabelRecord, 278  
PiePlot, 278  
PiePlot3D, 283  
PiePlotState, 284  
PieSectionEntity, 224  
PieSectionLabelGenerator, 243  
PieToolTipGenerator, 244  
Plot, 284  
PlotChangeEvent, 229  
PlotChangeListener, 230  
PlotOrientation, 287  
PlotRenderingInfo, 287  
PlotState, 288  
PNG, 150  
PolarChartPanel, 159  
PolarItemRenderer, 303  
  
PolarPlot, 288  
power regression, 423  
PowerFunction2D, 385  
PublicCloneable, 487  
  
Quarter, 437  
  
Range, 373  
range axis, 171  
RangeInfo, 375  
real time charts, 71  
RectangleAnchor, 487  
RectangleConstraint, 219  
RectangleEdge, 487  
RectangleInsets, 488  
Regression, 423  
RegularTimePeriod, 439  
RendererChangeEvent, 230  
RendererChangeListener, 230  
RendererState, 303  
rendering hints, 66  
rendering order  
    CategoryPlot, 262  
    XYPlot, 295  
RingPlot, 288  
  
sample charts, 18  
Second, 441  
SegmentedTimeline, 204  
Series, 407  
    SeriesChangeEvent, 408  
    SeriesChangeListener, 408  
    SeriesDataset, 409  
    SeriesException, 409  
    servlets, 126  
        deploying, 136  
    ServletUtilities, 351  
    SimpleTimePeriod, 442  
    Spacer, 489  
    StackedAreaRenderer, 325  
    StackedBarRenderer, 326  
    StackedBarRenderer3D, 326  
    StackedXYAreaRenderer, 334  
    StackedXYBarRenderer, 334  
    StandardCategoryLabelGenerator, 244  
    StandardCategoryToolTipGenerator, 246  
    StandardContourToolTipGenerator, 247  
    StandardEntityCollection, 224  
    StandardLegend, 160  
    StandardPieItemLabelGenerator, 247

StandardTickUnitSource, 204  
StandardXYItemRenderer, 336  
StandardXYLabelGenerator, 248  
StandardXYZToolTipGenerator, 249  
StandardXYZToolTipGenerator, 250  
StatisticalBarRenderer, 327  
StatisticalCategoryDataset, 424  
Statistics, 424  
step charts, 348  
SubCategoryAxis, 204  
subtitles, 65  
suppressing chart change events, 156  
SVG, 117  
SymbolicAxis, 204  
SymbolicTickUnit, 204  
SymbolicXYItemLabelGenerator, 251  
  
TableXYDataset, 475  
Task, 388  
TaskSeries, 390  
TaskSeriesCollection, 391  
TextAnchor, 490  
TextAnnotation, 163  
ThermometerPlot, 289  
Tick, 204  
TickLabelEntity, 225  
TickUnit, 205  
TickUnits, 205  
TickUnitSource, 206  
time series  
    tool tips, 249  
Timeline, 206  
TimePeriod, 442  
TimePeriodAnchor, 443  
TimePeriodFormatException, 443  
TimePeriodValue, 443  
TimePeriodValues, 444  
TimePeriodValuesCollection, 444  
TimeSeries, 445  
TimeSeriesCollection, 447  
TimeSeriesDataItem, 450  
TimeSeriesTableModel, 451  
TimeTableXYDataset, 451  
title, 65  
TitleChangeEvent, 231  
TitleChangeListener, 231  
tool tips  
    time series, 249  
 tooltips, 76  
  
Unicode, 113  
UnitType, 491  
unpacking the JFreeChart distribution, 34  
  
Value, 376  
ValueAxis, 207  
ValueAxisPlot, 292  
ValueDataset, 409  
ValueMarker, 292  
Values, 376  
Values2D, 377  
ValueTick, 210  
  
WaferMapDataset, 410  
WaferMapPlot, 293  
WaferMapRenderer, 304  
WaterfallBarRenderer, 328  
Week, 453  
WindDataset, 475  
WindItemRenderer, 337  
  
X11, 138  
XisSymbolic, 476  
XYAnnotation, 163  
XYAreaRenderer, 337  
XYBarDataset, 476  
XYBarRenderer, 339  
XYBoxAndWhiskerRenderer, 341  
XYBubbleRenderer, 342  
XYDataItem, 476  
XYDataset, 477  
XYDatasetTableModel, 478  
XYDifferenceRenderer, 342  
XYDotRenderer, 343  
XYDrawableAnnotation, 164  
XYImageAnnotation, 164  
XYItemEntity, 225  
XYItemRenderer, 344  
XYItemRendererState, 346  
XYLabelGenerator, 251  
XYLineAndShapeRenderer, 346  
XYLineAnnotation, 165  
XYPlot, 293  
XYPointerAnnotation, 166  
XYPolygonAnnotation, 168  
XYSeries, 479  
XYSeriesCollection, 482  
XYShapeAnnotation, 169  
XYStepAreaRenderer, 349

XYStepRenderer, 348  
XYTextAnnotation, 169  
XYToolTipGenerator, 251  
XYZDataset, 484  
XYZToolTipGenerator, 252  
  
Year, 455  
YIntervalRenderer, 349  
YisSymbolic, 485