



# **JClass® DesktopViews 6.4.2**

## Chart Programmer's Guide



**© 2009 Quest Software, Inc.  
ALL RIGHTS RESERVED.**

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Quest Software, Inc.

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters

LEGAL Dept

5 Polaris Way

Aliso Viejo, CA 92656

Email: [legal@quest.com](mailto:legal@quest.com)

Refer to our Web site ([www.quest.com](http://www.quest.com)) for regional and international office information.

## **Trademarks**

Quest, Quest Software, the Quest Software logo, Aelita, Akonix, AppAssure, Benchmark Factory, Big Brother, ChangeAuditor, DataFactory, DeployDirector, ERDisk, Foglight, Funnel Web, GPOAdmin, iToken, I/Watch, Imceda, InLook, IntelliProfile, InTrust, Invertus, IT Dad, I/Watch, JClass, Jint, JProbe, LeccoTech, LiteSpeed, LiveReorg, MessageStats, NBSpool, NetBase, Npulse, NetPro, PassGo, PerformaSure, Quest Central, SharePlex, Sitraka, SmartAlarm, Spotlight, SQL LiteSpeed, SQL Navigator, SQL Watch, SQLab, Stat, StealthCollect, Tag and Follow, Toad, T.O.A.D., Toad World, vAnalyzer, vAutomator, vControl, vConverter, vDupe, vEssentials, vFoglight, vMigrator, vOptimizer Pro, VPackager, vRanger, vRanger Pro, vReplicator, vSpotlight, vToad, Vintela, Virtual DBA, VizionCore, Vizioncore vAutomation Suite, Vizioncore vEssentials, Xaffire, and XRT are trademarks and registered trademarks of Quest Software, Inc in the United States of America and other countries. Other trademarks and registered trademarks used in this guide are property of their respective owners.

## **Disclaimer**

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST'S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

## **Third Party Contributions**

See Third\_Party\_Contributions.htm in your Quest Software License Server installation directory.

**Chart Programmer's Guide  
January 2009  
Version 6.4.2**

# Table of Contents

<b>Preface</b> . . . . .	<b>13</b>
Introducing JClass Chart. . . . .	13
Assumptions . . . . .	14
Typographical Conventions . . . . .	14
Overview of this Guide . . . . .	15
API Documentation (Javadoc). . . . .	16
Licensing . . . . .	16
Related Documents . . . . .	16
About Quest Software, Inc.. . . . .	17
JClass Community . . . . .	18

## Part I: Using JClass Chart

<b>1 JClass Chart Basics</b> . . . . .	<b>21</b>
1.1 Chart Areas . . . . .	21
1.2 Chart Types . . . . .	22
1.3 Supported Development Environments . . . . .	25
1.4 Adding Data . . . . .	26
Underlying Data Model . . . . .	26
Targeted Data Model . . . . .	26
1.5 Setting and Getting Object Properties . . . . .	27
Setting Properties with Java Code . . . . .	27
Setting Properties Interactively at Run-Time . . . . .	27
Setting Properties with a Java IDE at Design-Time . . . . .	28
Setting Properties with HTML or XML . . . . .	28
1.6 Other Programming Basics . . . . .	28
1.7 JClass Chart Inheritance Hierarchy . . . . .	29
1.8 JClass Chart Object Containment . . . . .	30
1.9 JClass Chart Customizer . . . . .	31
Displaying JClass Chart Customizer at Run-Time . . . . .	32
Editing and Viewing Properties . . . . .	32
Saving Chart Properties to a File . . . . .	33
Closing JClass Chart Customizer . . . . .	33
1.10 Internationalization . . . . .	33

<b>2</b>	<b>Chart Programming Tutorial. . . . .</b>	<b>35</b>
2.1	Introduction . . . . .	35
2.2	A Basic Plot Chart . . . . .	35
2.3	Loading Data From a File . . . . .	38
2.4	Adding Header, Footer, and Labels . . . . .	39
2.5	Changing to a Bar Chart . . . . .	42
2.6	Inverting Chart Orientation . . . . .	43
2.7	Bar3d and 3d Effect . . . . .	44
2.8	End-User Interaction . . . . .	44
2.9	Get Started Programming with JClass Chart . . . . .	45
<b>3</b>	<b>Selecting a Chart Type . . . . .</b>	<b>47</b>
3.1	Plot and Scatter Plot Charts . . . . .	47
3.2	Area and Stacking Area Charts . . . . .	48
3.3	Bar and Stacking Bar Charts . . . . .	49
3.4	Financial Charts . . . . .	51
3.5	Pie Charts . . . . .	53
3.6	Polar Charts . . . . .	58
3.7	Radar Charts . . . . .	62
3.8	Area Radar Charts . . . . .	64
<b>4</b>	<b>Adding Data with the Underlying Data Model . . . . .</b>	<b>67</b>
4.1	Understanding the Underlying Data Model . . . . .	67
4.2	Pre-Built Chart DataSources . . . . .	68
4.3	Loading Data from a File . . . . .	68
4.4	Loading Data from a URL . . . . .	68
4.5	Loading Data from an Applet . . . . .	69
4.6	Loading Data from a Swing TableModel . . . . .	70
4.7	Loading Data from an XML Source . . . . .	70
	Specifying Data by Series . . . . .	71
	Specifying Data by Point . . . . .	72
	Labels and Other Parameters . . . . .	73
4.8	Text Data Formats . . . . .	73
	Formatted Data Examples . . . . .	75
	Explanation of Format Elements . . . . .	75

4.9	Making Your Own Chart Data Source . . . . .	77
	The Simplest Chart Data Source Possible . . . . .	77
	LabelledChartDataModel – Labelling Your Chart . . . . .	79
	EditableChartDataModel – Modifying Your Data . . . . .	80
	HoleValueChartDataModel – Specifying Hole Values . . . . .	81
4.10	Making an Updating Chart Data Source . . . . .	82
	Chart Data Source Support Classes . . . . .	82
<b>5</b>	<b>Adding Data with the Targeted Data Model. . . . .</b>	<b>85</b>
5.1	Overview of the Targeted Data Model . . . . .	86
5.2	Review of JDBC Result Sets . . . . .	86
5.3	Adding Data from a Result Set to a Chart . . . . .	87
	Creating a Data Set Implementation for a Result Set . . . . .	87
	Creating an Instance of DefaultDataModel . . . . .	89
	Setting the Data Model on Your Chart . . . . .	90
5.4	Result Set Data Set Implementations by Chart Type . . . . .	90
	BasicResultSetDataSet . . . . .	91
	BarResultSetDataSet . . . . .	93
	FinancialResultSetDataSet . . . . .	96
	PieResultSetDataSet . . . . .	99
	PolarResultSetDataSet . . . . .	102
	RadarResultSetDataSet . . . . .	105
5.5	Advanced Topics . . . . .	108
5.6	Creating a Custom Data Set Implementation . . . . .	112
	Understanding the Targeted Data Model . . . . .	112
	Creating a BasicDataSet Implementation . . . . .	114
	Summary of the Interfaces and Classes . . . . .	116
	A BarDataSet Implementation Example . . . . .	119
<b>6</b>	<b>Defining Axis Controls . . . . .</b>	<b>121</b>
6.1	Axis Labelling and Annotation Methods . . . . .	121
	Choosing an Annotation Method . . . . .	122
	Values Annotation . . . . .	122
	PointLabels Annotation . . . . .	124
	ValueLabels Annotation . . . . .	124
	TimeLabels Annotation . . . . .	125
	Custom Axes Labels . . . . .	128
6.2	Positioning Axes . . . . .	129

6.3	Chart Orientation and Axis Direction . . . . .	130
	Inverting Chart Orientation . . . . .	131
	Changing Axis Direction . . . . .	131
6.4	Setting Axis Bounds . . . . .	132
6.5	Customizing Origins . . . . .	132
6.6	Logarithmic Axes . . . . .	133
6.7	Titling Axes and Rotating Axis Elements . . . . .	133
6.8	Gridlines . . . . .	135
6.9	Adding a Second Y-Axis . . . . .	137

## **7 Defining Text and Style Elements . . . . . 139**

7.1	Header and Footer Titles . . . . .	139
7.2	Legends . . . . .	139
	Types of Legends . . . . .	140
	Customizing Legends . . . . .	141
	Creating Custom Legends with the JCLegend Toolkit . . . . .	143
7.3	Chart Labels . . . . .	151
	Label Implementation . . . . .	151
	Adding Labels to a Chart . . . . .	152
	Interactive Labels . . . . .	153
	Adding and Formatting Label Text . . . . .	154
	Positioning Labels . . . . .	154
	Adding Connecting Lines . . . . .	154
7.4	Chart Styles . . . . .	155
7.5	Outline Style . . . . .	157
7.6	Hole Styles . . . . .	158
	Example of Hole Styles . . . . .	158
	Hole Styles in Plot and Polar Charts . . . . .	159
	Hole Styles in Area Charts . . . . .	160
7.7	Borders . . . . .	161
7.8	Fonts . . . . .	161
7.9	Colors . . . . .	161
7.10	Positioning Elements on the Chart Object . . . . .	163
7.11	3D Effect . . . . .	164
7.12	Anti-Aliasing . . . . .	164

<b>8</b>	<b>Defining Markers and Thresholds . . . . .</b>	<b>167</b>
8.1	Markers . . . . .	167
	Creating X-axis Markers . . . . .	168
	Creating Y-axis Markers . . . . .	169
	Creating a Crosshair with Markers . . . . .	171
	Customizing Markers . . . . .	172
8.2	Thresholds . . . . .	177
	Creating X-axis Thresholds . . . . .	178
	Creating Y-axis Thresholds . . . . .	179
	Overlapping and Intersecting Thresholds . . . . .	181
	Customizing Thresholds . . . . .	182
<b>9</b>	<b>Advanced Chart Programming. . . . .</b>	<b>187</b>
9.1	Outputting JClass Charts . . . . .	187
	Encode method . . . . .	187
	Encode example . . . . .	188
	Code example . . . . .	188
9.2	Batching Chart Updates . . . . .	189
9.3	FastAction . . . . .	189
9.4	FastUpdate . . . . .	189
9.5	Programming End-User Interaction . . . . .	190
	Event Triggers . . . . .	190
	Valid Modifiers . . . . .	191
	Programming Event Triggers . . . . .	191
	Removing Action Mappings . . . . .	191
	Calling an Action Directly . . . . .	191
	Specifying Action Axes . . . . .	192
9.6	Map and Unmap . . . . .	192
9.7	Pick . . . . .	192
	Pick Methods for Polar and Radar Charts . . . . .	193
	Pick Methods for Area Radar Charts . . . . .	193
	Pick Focus . . . . .	193
9.8	Unpick . . . . .	194
9.9	Using Pick and Unpick . . . . .	194
9.10	Coordinate Conversion Methods . . . . .	199
9.11	Image-Filled Bar Charts . . . . .	201

## Part II: Supported Technologies

<b>10</b>	<b>Using JCChartFactory</b>	<b>205</b>
10.1	Overview of the JCChartFactory Class	205
10.2	Overview of the LoadProperties Class	206
	LoadProperties Class and the fileAccess Property	207
	When to Define a LoadProperties Object	207
10.3	Saving Data: The OutputDataProperties Class	208
10.4	Saving Image Information: The OutputProperties Class	209
	Constructing an OutputProperties Object	210
	Setting Output Properties on an Image	210
<b>11</b>	<b>Loading and Saving Charts Using HTML</b>	<b>211</b>
11.1	Overview of HTML for JClass Chart	211
11.2	Creating a Chart from HTML	212
	Specifying JClass Chart Properties Using HTML Tags	213
	Creating the Chart and Loading HTML-based Properties	214
11.3	Updating a Chart Using HTML	215
	Updating a Chart with New Chart Properties	215
	Updating a Chart with a New Data Set	216
11.4	Saving a Chart to HTML	217
	Saving Data When a Chart is Saved to HTML	218
	Saving Image Information to HTML	218
<b>12</b>	<b>Loading and Saving Charts Using XML</b>	<b>219</b>
12.1	Background XML Information	219
12.2	Overview of XML for JClass Chart	220
12.3	Creating a Chart Using XML	220
	Specifying JClass Chart Properties Using XML Elements	221
	Creating the Chart and Loading XML-based Properties	225
12.4	Updating a Chart Using XML	225
	Updating a Chart with New Chart Properties	225
	Updating a Chart with a New Data Set	226
12.5	Saving a Chart to XML	227
	Saving Data When a Chart is Saved to XML	228
	Saving Image Information to XML	228



12.6	Internationalizing Your XML-based Chart . . . . .	229
	Using Variables . . . . .	229
	Creating a Resource Bundle . . . . .	230
	Using Resource Bundles . . . . .	230
<b>13</b>	<b>JClass Chart Beans . . . . .</b>	<b>233</b>
13.1	Choosing the Right Bean . . . . .	233
	JClass Chart Beans . . . . .	234
	JClass Chart Beans and JCCart . . . . .	234
13.2	Standard Bean Properties . . . . .	234
	Axis Properties . . . . .	234
	Chart Types . . . . .	238
	Display Properties . . . . .	238
	Headers and Footers . . . . .	240
	Legends . . . . .	240
13.3	Data-Loading Methods . . . . .	241
	SimpleChart: Loading Data from a File . . . . .	242
	SimpleChart: Using Swing TableModel Data Objects . . . . .	243
<b>14</b>	<b>SimpleChart Bean Tutorial . . . . .</b>	<b>245</b>
14.1	Introduction to JavaBeans . . . . .	245
	Properties . . . . .	245
14.2	SimpleChart Bean Tutorial . . . . .	246
	Steps in this Tutorial . . . . .	246
<b>15</b>	<b>MultiChart Bean . . . . .</b>	<b>253</b>
15.1	Introduction to MultiChart . . . . .	253
	Multiple Axes . . . . .	253
	Multiple Data Views . . . . .	254
	Intelligent Defaults . . . . .	254
15.2	Getting Started with MultiChart . . . . .	254
15.3	MultiChart Property Reference . . . . .	255
	Axis Controls . . . . .	255
	Headers, Footers, and Legends . . . . .	264
	Data Source and Data View Controls . . . . .	266
	Appearance Controls . . . . .	270
	View3D . . . . .	273
	Event Controls . . . . .	273

## Part III: Reference Appendices

<b>A</b>	<b>Summary of Properties for JClass Chart Objects. . . . .</b>	<b>277</b>
A.1	ChartDataView . . . . .	277
A.2	ChartDataViewSeries . . . . .	280
A.3	ChartText . . . . .	281
A.4	JCAnno . . . . .	282
A.5	JCAreaChartFormat . . . . .	284
A.6	JCAxis . . . . .	284
A.7	JCAxisFormula . . . . .	290
A.8	JCAxisTitle . . . . .	290
A.9	JCBarChartFormat . . . . .	292
A.10	JCCandleChartFormat . . . . .	292
A.11	JCChart . . . . .	293
A.12	JCChartArea . . . . .	295
A.13	JCChartLabel . . . . .	296
A.14	JCChartLabelManager . . . . .	297
A.15	JCChartStyle . . . . .	297
A.16	JCFillStyle . . . . .	299
A.17	JCGrid . . . . .	299
A.18	JCGridLegend . . . . .	300
A.19	JCHiloChartFormat . . . . .	301
A.20	JCHLOCCChartFormat . . . . .	301
A.21	JCLegend . . . . .	302
A.22	JCLineStyle . . . . .	303
A.23	JCMarker . . . . .	304
A.24	JCMultiColLegend . . . . .	305
A.25	JCPieChartFormat . . . . .	306
A.26	JCPolarRadarChartFormat . . . . .	307
A.27	JCSymbolStyle . . . . .	308
A.28	JCThreshold . . . . .	309
A.29	JCValueLabel . . . . .	310
A.30	PlotArea . . . . .	310
A.31	SimpleChart . . . . .	311
<b>B</b>	<b>HTML Syntax . . . . .</b>	<b>315</b>
B.1	ChartDataView Properties . . . . .	316
B.2	ChartDataViewSeries Properties . . . . .	317

B.3	Header and Footer Properties . . . . .	318
B.4	JCAreaChartFormat Properties . . . . .	318
B.5	JCAnnoProperties . . . . .	319
B.6	JCAxis X-Axes and Y-Axes Properties . . . . .	319
B.7	JCBarChartFormat Properties . . . . .	321
B.8	JCCandleChartFormat Properties . . . . .	322
B.9	JCChart Properties . . . . .	322
B.10	JCChartArea Properties . . . . .	323
B.11	JCChartLabel Properties . . . . .	324
B.12	JCDataIndex Properties . . . . .	325
B.13	JCGrid Properties . . . . .	325
B.14	JCHiLoChartFormat Properties . . . . .	326
B.15	JCHLOCCChartFormat Properties . . . . .	326
B.16	JCLegend Properties . . . . .	326
B.17	JCMarker Properties . . . . .	327
B.18	JCMultiColLegend Properties . . . . .	328
B.19	JCPieChartFormat Properties . . . . .	329
B.20	JCPolarRadarChartFormat Properties . . . . .	329
B.21	JCThreshold Properties . . . . .	330
<b>C</b>	<b>XML DTD . . . . .</b>	<b>331</b>
C.1	Chart.dtd . . . . .	332
C.2	JCChartData.dtd . . . . .	376
<b>D</b>	<b>Distributing Applets and Applications . . . . .</b>	<b>381</b>



# Preface

*[Introducing JClass Chart](#) ■ [Assumptions](#) ■ [Typographical Conventions](#)  
[Overview of this Guide](#) ■ [API Documentation \(Javadoc\)](#) ■ [Licensing](#)  
[Related Documents](#) ■ [About Quest Software, Inc.](#)*

## Introducing JClass Chart

JClass Chart is a charting/graphing component written entirely in Java. The chart component displays data graphically in a window and can interact with a user.

The chart component can be used easily by all types of Java programmers:

- Component users, setting JClass Chart properties programmatically.
- OO developers, instantiating and extending JClass Chart objects.
- JavaBean developers, setting JClass Chart properties using a third-party Integrated Development Environment (IDE).

You can freely distribute Java applets and applications containing JClass components according to the terms of the License Agreement that appears during the installation.

## Feature Overview

You can set the properties of JClass Chart objects to determine how the chart will look and behave. You can control:

- Chart type (plot, scatter plot, area, stacking area, bar, stacking bar, pie, Hi-Lo, Hi-Lo-Open-Close, candle, polar, radar, and area radar)
- Header and footer positioning, border style, text, font, and color
- Number of data views, each having its own data, chart type, axes, and chart styles
- Flexible data loading from applets, files, URLs, input streams, and databases
- Chart styles: line color, fill color, point size, point style, and point color
- Legend positioning, orientation, border style, anchor, font, and color
- Chart positioning, border style, color, width, height, and 3D effect (bar, stacking bar, and pie charts only)
- Axis labelling using Point labels, Series labels, Value labels, or Time labels
- Number of x- or y-axes, each having its own minimum and maximum, axis numbering method, numbering and ticking increment, grid increment, font, origin, axis direction, and precision
- Control of user interaction with components including picking, mapping, Chart Customizer, rotation, scaling, and translation

- Chart labels that can appear anywhere on the chart, including automatic dwell labels for each point on the chart
- Marker lines and threshold areas

JClass Chart is compatible with JDK 1.4. If you are using JDK 1.4 and experience drawing problems, you may want to upgrade to the latest drivers for your video card from your video card vendor.

## Assumptions

This manual assumes that you have some experience with the Java programming language. You should have a basic understanding of object-oriented programming and Java programming concepts such as classes, methods, and packages before proceeding with this manual. See [Related Documents](#) later in this section of the manual for additional sources of Java-related information.

## Typographical Conventions

### Typewriter Font

- Java language source code and examples of file contents.
- JClass Chart and Java classes, objects, methods, properties, constants, and events.
- HTML documents, tags, and attributes.
- Commands that you enter on the screen.

### *Italic Text*

- Pathnames, filenames, URLs, programs, and method parameters.
- New terms as they are introduced, and to emphasize important words.
- Figure and table titles.
- The names of other documents referenced in this manual, such as *Java in a Nutshell*.

### **Bold**

- Keyboard key names and menu references.

## Overview of this Guide

### Part I – Using JClass Chart

Chapter 1, [JClass Chart Basics](#), provides a programmer's overview of JClass Chart. It covers class hierarchy, object containment, terminology, programming basics, and specific issues to be aware of before using JClass Chart.

Chapter 2, [Chart Programming Tutorial](#), introduces you to JClass Chart programming by compiling and running an example program. It includes examples of common chart programming tasks.

Chapter 3, [Selecting a Chart Type](#), outlines the special features of the JClass Chart chart types.

Chapter 4, [Adding Data with the Underlying Data Model](#), how to use different pre-built data sources and outlines how to use the data source toolkit to create your own.

Chapter 5, [Adding Data with the Targeted Data Model](#), describes how to use the targeted data model to add data in a JDBC result set to a chart.

Chapter 6, [Defining Axis Controls](#), covers JClass Chart properties used when first setting up your chart, concentrating on axis properties.

Chapter 7, [Defining Text and Style Elements](#), covers JClass Chart properties used to customize the appearance of a chart, including header/footer, legend, and chart styles.

Chapter 8, [Defining Markers and Thresholds](#), describes how to use markers and thresholds to highlight data values in your chart.

Chapter 9, [Advanced Chart Programming](#), looks at programming more advanced aspects of the chart.

### Part II – Supported Technologies

Chapter 10, [Using JCChartFactory](#), introduces the factory that is used when creating, updating, and saving charts using HTML or XML.

Chapter 11, [Loading and Saving Charts Using HTML](#), describes how to create a chart using HTML syntax, how to update the chart, and how to save it.

Chapter 12, [Loading and Saving Charts Using XML](#), describes how to create a chart from XML, how to update it, and how to save it. It also contains procedures to internationalize your chart.

Chapter 13, [JClass Chart Beans](#), is a guide to the different JClass Chart Beans. It illustrates all of the properties available, including the different data loading methods.

Chapter 14, [SimpleChart Bean Tutorial](#), introduces basic Bean concepts, and guides you through developing a chart application in an IDE.

Chapter 15, [MultiChart Bean](#), is a user's guide for MultiChart, an advanced charting Bean.

### Part III – Reference Appendices

Appendix A, [Summary of Properties for JClass Chart Objects](#), summarizes the properties contained in all of the JClass Chart objects.

Appendix B, [HTML Syntax](#), lists the HTML syntax to use to define JClass Chart properties in HTML.

Appendix C, [XML DTD](#), lists the XML elements to use to define JClass Chart properties in XML.

Appendix D, [Distributing Applets and Applications](#), is an overview of how to deploy applets and applications.

## API Documentation (Javadoc)

The *JClass DesktopViews API Documentation* (Javadoc) is installed automatically when you install JClass Chart and is found in the `JCLASS_HOME/docs/api/` directory.

## Licensing

In order to use JClass Chart, you need a valid license. Complete details about licensing are outlined in the *JClass DesktopViews Installation Guide*, which is automatically installed when you install JClass Chart.

## Related Documents

The following is a sample of useful references to Java and JavaBeans programming:

- “*Java Platform Documentation*” at <http://java.sun.com/docs/index.html> and the “*Java Tutorial*” at <http://java.sun.com/docs/books/tutorial/index.html> from Sun Microsystems
- For an introduction to creating enhanced user interfaces, see “*Creating a GUI with JFC/Swing*” at <http://java.sun.com/docs/books/tutorial/uiswing/index.html>
- *Java in a Nutshell, 2nd Edition* from O'Reilly & Associates Inc. See the O'Reilly Java Resource Center at <http://java.oreilly.com>.
- Resources for using JavaBeans are at <http://java.sun.com/beans/resources.html>

These documents are not required to develop applications using JClass Chart, but they can provide useful background information on various aspects of the Java programming language.



## About Quest Software, Inc.

Quest Software, Inc. delivers innovative products that help organizations get more performance and productivity from their applications, databases and Windows infrastructure. Through a deep expertise in IT operations and a continued focus on what works best, Quest helps more than 18,000 customers worldwide meet higher expectations for enterprise IT. Quest Software can be found in offices around the globe and at [www.quest.com](http://www.quest.com).

### Contacting Quest Software

Email	<a href="mailto:info@quest.com">info@quest.com</a>
Mail	Quest Software, Inc. World Headquarters 5 Polaris Way Aliso Viejo, CA 92656 USA
Web site	<a href="http://www.quest.com">www.quest.com</a>

Refer to our web site for regional and international office information.

### Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a commercial version and have a valid maintenance contract. Quest Support provides around the clock coverage with SupportLink, our web self-service. Visit SupportLink at: <http://support.quest.com>

From SupportLink, you can do the following:

- Quickly find thousands of solutions (Knowledgebase articles/documents).
- Download patches and upgrades.
- Seek help from a Support engineer.
- Log and update your case, and check its status.

View the *Global Support Guide* for a detailed explanation of support programs, online services, contact information, and policy and procedures. The guide is available at: [http://support.quest.com/pdfs/Global Support Guide.pdf](http://support.quest.com/pdfs/Global%20Support%20Guide.pdf)

Please note that many of the initial questions you may have will concern basic installation or configuration issues. Consult this product's [readme file](#) and the [JClass Desktop Views Installation Guide](#) (available in HTML and PDF formats) for help with these types of problems.

## **JClass Community**

For the latest product information, helpful resources, and discussions with the JClass Quest team and other community members, join the JClass community at <http://jclass.inside.quest.com/>.

# *Part I*

## *Using JClass Chart*



# JClass Chart Basics

*Chart Areas ■ Chart Types ■ Supported Development Environments ■ Adding Data  
Setting and Getting Object Properties ■ Other Programming Basics  
JClass Chart Inheritance Hierarchy ■ JClass Chart Object Containment  
JClass Chart Customizer ■ Internationalization*

This chapter covers concepts and vocabulary used in JClass Chart programming, and provides an overview of the JClass Chart class hierarchy.

## 1.1 Chart Areas

The following illustration shows the terms used to describe chart areas:

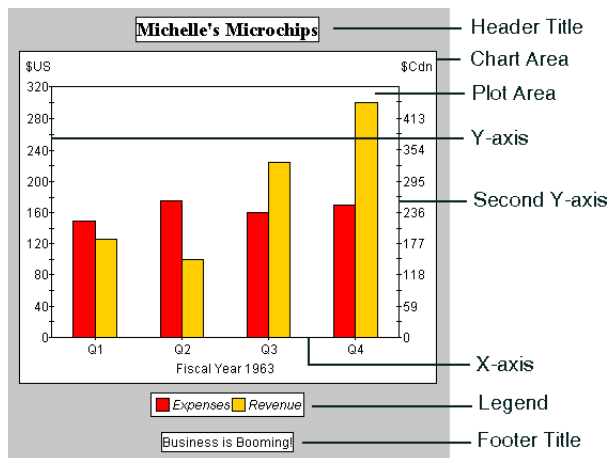


Figure 1 Elements contained in a typical chart.

## 1.2 Chart Types

JClass Chart can display data in the following basic chart types: plot, scatter plot, area, stacking area, bar, stacking bar, pie, Hi-Lo, Hi-Lo-Open-Close, candle, polar, radar, and area radar. You can also create more specialized charts by building on one of the basic types.

Use the `ChartType` property to set the chart type for one `ChartDataView`. Each data view managed by the chart has its own chart type. The following table lists basic information about each chart type, including the enumeration that sets that type and the data layouts it can display (see the next section for an introduction to data).

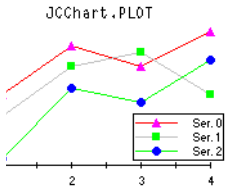
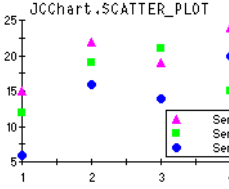
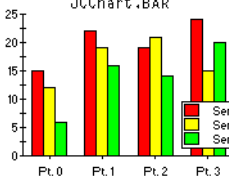
Chart Type	Single X-series	Multiple X-series	Notes
	✓	✓	<p><b>Plot</b></p> <p>Draws each series as connected points of data.</p> <ul style="list-style-type: none"> <li>Series appearance determined by chart style line color, symbol shape, size, and color properties.</li> </ul>
	✓	✓	<p><b>Scatter Plot</b></p> <p>Draws each series as unconnected points of data.</p> <ul style="list-style-type: none"> <li>Series appearance determined by chart style symbol shape, size, and color properties.</li> </ul>
	✓		<p><b>Bar</b></p> <p>Draws each series as a bar in a cluster. The number of clusters is the number of points in the data. Each cluster displays the <i>n</i>th point in each series.</p> <ul style="list-style-type: none"> <li>x-axis generally annotated using Point labels.</li> <li>Series appearance determined by chart style fill color and image properties.</li> <li>3D effect available using depth, elevation, and rotation properties.</li> </ul>

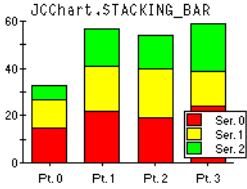
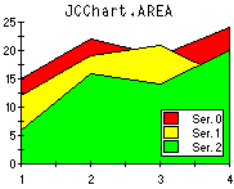
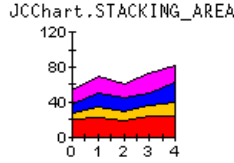
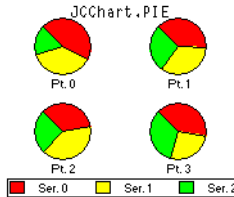
Chart Type	Single X-series	Multiple X-series	Notes
	✓		<p><b>Stacking Bar</b></p> <p>Draws each series as a portion of a stacked bar cluster, the number of clusters being the number of data points. Each cluster displays the <math>n</math>th point in each series. Negative y-values are stacked below the x-axis.</p> <ul style="list-style-type: none"> <li>■ x-axis generally annotated using Point labels.</li> <li>■ Series appearance determined by chart style fill color property.</li> <li>■ 3D effect available using depth, elevation, and rotation properties.</li> </ul>
	✓		<p><b>Area</b></p> <p>Draws each series as connected points of data, filled below the points. Each series is layered over the preceding series.</p> <ul style="list-style-type: none"> <li>■ Series appearance determined by chart style fill color property.</li> </ul>
	✓		<p><b>Stacking Area</b></p> <p>Draws each series as connected points of data, filled below the points. Places each y-series on top of the last one to show the area relationships between each series and the total.</p> <ul style="list-style-type: none"> <li>■ Series appearance determined by chart style fill color property.</li> </ul>
	✓		<p><b>Pie</b></p> <p>Draws each series as a slice of a pie. The number of pies is the number of points in the data (values below a certain threshold can be grouped into an <i>other</i> slice). Each pie displays the <math>n</math>th point in each series.</p> <ul style="list-style-type: none"> <li>■ Pies are annotated with Point labels only.</li> <li>■ Series appearance determined by chart style fill color property.</li> <li>■ 3D effect available using depth and elevation properties.</li> </ul>

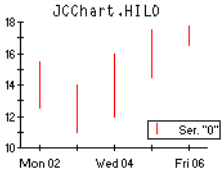
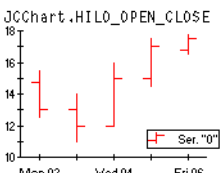
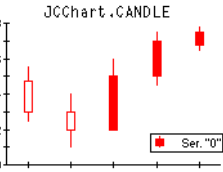
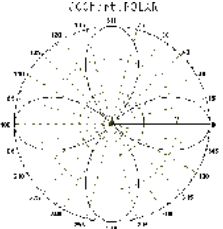
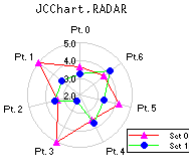
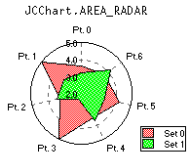
Chart Type	Single X-series	Multiple X-series	Notes
	✓		<p><b>Hi-Lo</b></p> <p>Draws <i>two</i> series together as a “high-low” bar. The points in each series define one portion of the bar:</p> <ul style="list-style-type: none"> <li>1st series – points are the “high” value</li> <li>2nd series – points are the “low” value</li> </ul> <ul style="list-style-type: none"> <li>■ Appearance determined by chart style line color property in the <b>first</b> series of each pair.</li> </ul>
	✓		<p><b>Hi-Lo-Open-Close</b></p> <p>Similar to Hi-Lo, but draws <i>four</i> series together as a “high-low-open-close” bar. The additional series’ points make up the other components of the bar:</p> <ul style="list-style-type: none"> <li>3rd series – points are the “open” value</li> <li>4th series – points are the “close” value</li> </ul> <ul style="list-style-type: none"> <li>■ Appearance determined by chart style line color and symbol size properties in the <b>first</b> series of each set.</li> </ul>
	✓		<p><b>Candle</b></p> <p>A special type of Hi-Lo-Open-Close chart; draws four series together as a “candle” bar.</p> <ul style="list-style-type: none"> <li>■ <i>Simple</i> candle appearance determined by chart style line color, fill color, and symbol size properties in the <b>first</b> series of each set.</li> <li>■ <i>Complex</i> candle appearance determined by different chart style properties from <b>each</b> series of each set.</li> </ul>
	✓	✓	<p><b>Polar</b></p> <p>Draws each series as connected points of data on a polar coordinate system (<i>theta</i>, <i>r</i>). <i>x</i>-values represent the amount of rotation and <i>y</i>-values are the distance from the origin.</p> <ul style="list-style-type: none"> <li>■ When using Array data, <i>x</i>-values are shared across series.</li> <li>■ <i>x</i>-axis bounds cannot be set; <i>y</i>-axis bounds cannot be set inside the data extents.</li> <li>■ Appearance determined by ChartStyles’ line and symbol properties of each series.</li> </ul>



Chart Type	Single X-series	Multiple X-series	Notes
	✓		<p><b>Radar</b></p> <p>Draws each series as connected points along radar “sticks” spaced equally apart. The <math>m</math>th stick charts the <math>y</math>-value of the <math>m</math>th point in each series.</p> <ul style="list-style-type: none"> <li>■ x-axis annotated with Point-labels or integer values.</li> <li>■ Appearance determined by ChartStyles’ line and symbol properties of each series.</li> </ul>
	✓		<p><b>Area Radar</b></p> <p>Draws each series as connected points of data, filled inside the points. The points are the same as they would be for a Radar chart. Each series is drawn “on top” of the preceding series.</p> <ul style="list-style-type: none"> <li>■ x-axis annotated with Point-labels or integer values.</li> <li>■ Appearance determined by ChartStyles’ fill and line properties.</li> </ul>

### 1.3 Supported Development Environments

Part I of this guide shows you how to create and customize a JClass Chart programmatically. You can, however, also add JClass Chart components to applications developed for the following environments:

- applets
- HTML
- XML
- JavaBeans

These environments are touched upon throughout this guide, but Part II concentrates on using JClass Chart with supported technologies:

- [Using JCChartFactory](#), in Chapter 10
- [Loading and Saving Charts Using HTML](#), in Chapter 11
- [Loading and Saving Charts Using XML](#), in Chapter 12
- [JClass Chart Beans](#), in Chapter 13
- [SimpleChart Bean Tutorial](#), in Chapter 14
- [MultiChart Bean](#), in Chapter 15

## 1.4 Adding Data

You add data to a chart using one of two data models: the *underlying data model* or the *targeted data model*. The underlying data model is universal, while the targeted data model can be used only for certain types of applications. The following sections summarize the data models.

**Tip:** If your data is stored as a JDBC result set and you are creating your chart programmatically, review the [Targeted Data Model](#) section first.

### 1.4.1 Underlying Data Model

The underlying data model is data-format dependent and chart-type independent. This data model requires that your data be stored as an array of doubles, which means that you may need to create a compliant data source. The chart type, however, can be changed by resetting a single property.

The underlying data model can be used to add data to any JClass Chart application. It is also the preferred model if your data is dynamic and needs to be updated frequently. For more information, see Chapter 4, [Adding Data with the Underlying Data Model](#).

### 1.4.2 Targeted Data Model

The targeted data model is data-format independent and chart-type dependent. This means that your data can be stored in any format and, as you match the data in your data source to chart elements, you do so with properties that are meaningful for the type of chart that you selected. Behind the scenes, JClass Chart transforms your data into an array of doubles and calls the underlying data model to do the actual work of creating the chart.

While the targeted data model is data-format independent, the *implementation* of the data model that ships with JClass Chart is designed specifically for JDBC result sets. The implementation includes pre-built result set data set classes; you instantiate the data set class designed for the chart type that you selected. For more information, see Chapter 5, [Adding Data with the Targeted Data Model](#).

The targeted data model is supported for applets and any other application where you have programmatic access to the chart. You may want to review the JClass Chart examples that implement the targeted data model to see how easy it is to add data to a chart using the chart-type specific interfaces.

**Note:** The targeted data model is not supported for JavaBeans and XML. You need to use the underlying data model.

## 1.5 Setting and Getting Object Properties

You can set and retrieve JClass Chart properties in the following ways:

- Calling property set and get methods in a Java program
- Using the JClass Chart Customizer at run-time
- Using a Java IDE with the JClass Chart beans
- Specifying properties using a markup language (HTML or XML)

Each method changes the same chart property. This manual therefore uses *properties* to discuss how features work, rather than referring to the method, JClass Chart Customizer tab, or HTML parameter you might use to set that property.

**Note:** In most cases, you need to understand the chart's object containment hierarchy to access its properties. Use the [Objects contained in a chart – traverse contained objects to access properties](#) diagram to determine how to access the properties of an object.

### 1.5.1 Setting Properties with Java Code

Every JClass Chart property has a set and get method associated with it. For example, to retrieve the value of the `AnnotationMethod` property of the first x-axis, the `getAnnotationMethod()` method is called:

```
method = c.getChartArea().getXAxis(0).getAnnotationMethod();
```

To set the `AnnotationMethod` property of the same axis:

```
c.getChartArea().getXAxis(0).setAnnotationMethod(  
    JCAxis.POINT_LABELS);
```

These statements navigate the objects contained in the chart by retrieving the values of successive properties, which are contained objects. In the code above, the value of the `ChartArea` property is a `JCChartArea` object. The chart area has an `XAxis` property, the value of which is a collection of `JCAxis` objects. The axis also has the desired `AnnotationMethod` property.

For detailed information on the properties available for each object, consult the online [API](#) reference documentation. The API is automatically installed when you install JClass and is found in the `JCLASS_HOME/docs/api/` directory.

### 1.5.2 Setting Properties Interactively at Run-Time

If enabled by the developer, end-users can manipulate property values on a chart running in your application. Clicking a mouse button launches the JClass Chart Customizer. The user can navigate through the tabbed dialogs and edit the properties displayed. For details on enabling and using JClass Chart Customizer, see Section 1.9, [JClass Chart Customizer](#).

### 1.5.3 Setting Properties with a Java IDE at Design-Time

A JClass Chart bean can be used with a Java Integrated Development Environment (IDE), and its properties can be manipulated at design-time. Consult your IDE's documentation for details on how to load third-party bean components into the IDE. You can also refer to the [JClass and Your IDE](#) chapter in the *Installation Guide*.

Most IDEs list a component's properties in a property sheet or dialog. Simply find the property you want to set in this list and edit its value. Again, consult your IDE's documentation for complete details.

### 1.5.4 Setting Properties with HTML or XML

You can use HTML or XML to set and save chart properties. This has the following benefits:

- **Speed** – You can see the effects of a set of property values quickly.
- **Flexibility** – You can use a single class to create many different kinds of charts simply by varying HTML/XML properties; end-users can modify properties to suit their own needs.
- **Repeatability** – You can save the values of chart properties to a file, which can serve as a useful testing and debugging tool.

To create or update a chart from HTML or XML, and to save a chart to HTML or XML, you use the `JCChartFactory` class. For more information, see Chapter 10, [Using JCChartFactory](#).

## 1.6 Other Programming Basics

### Working with Object Collections

Many chart objects are organized into collections. For example, the chart axes are organized into the `XAxis` collection and the `YAxis` collection. In JavaBean terminology, these objects are held in indexed properties.

To access a particular element of a collection, specify the index that uniquely identifies this element. For example, the following code changes the maximum value of the first x-axis to 25.1:

```
c.getChartArea().getAxis(0).setMax(25.1);
```

The index zero refers to the first element of a collection. By default, `JCChartArea` contains one element in `XAxis` and one in `YAxis`. For a polar, radar, and area radar chart, there can be only one y-axis and one x-axis.

## Calling Methods

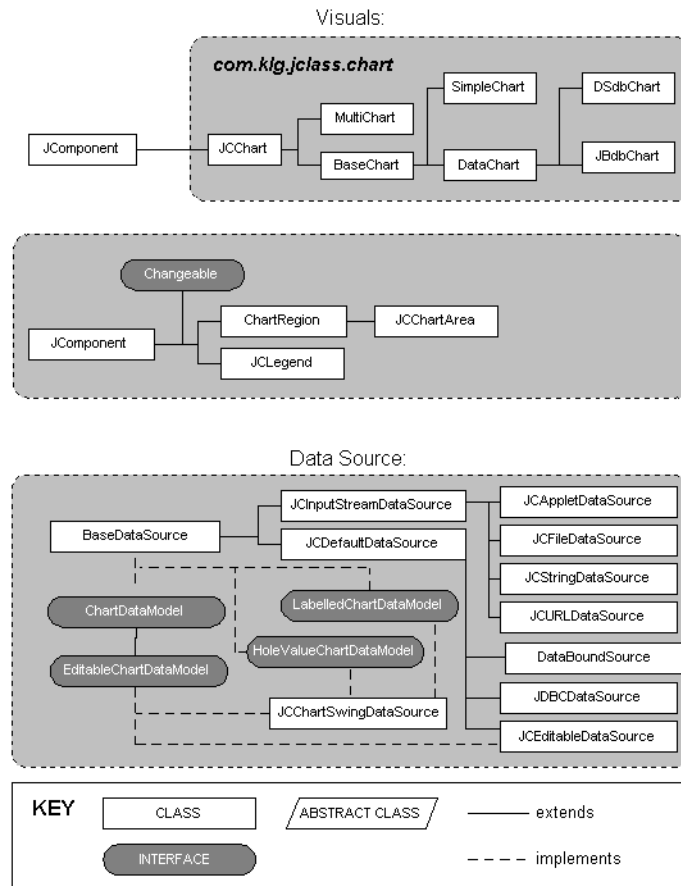
To call a JClass Chart method, access the object that defines the method. For example, the following statement uses the `coordToDataCoord()` method, defined by the `ChartDataView` collection, to convert the location of a mouse click event in pixels to their equivalent in data coordinates:

```
JCDataCoord dc = c.getDataView(0).coordToDataCoord(10,15);
```

Details on each method can be found in the [API documentation](#) for each class.

## 1.7 JClass Chart Inheritance Hierarchy

The following diagram provides an overview of class inheritance of JClass Chart.



## 1.8 JClass Chart Object Containment

When you create (or instantiate) a new chart, several other objects are also created. These objects are contained in and are part of the chart. Chart programmers need to traverse these objects to access the properties of a contained object. The following diagram shows the object containment for JClass Chart.

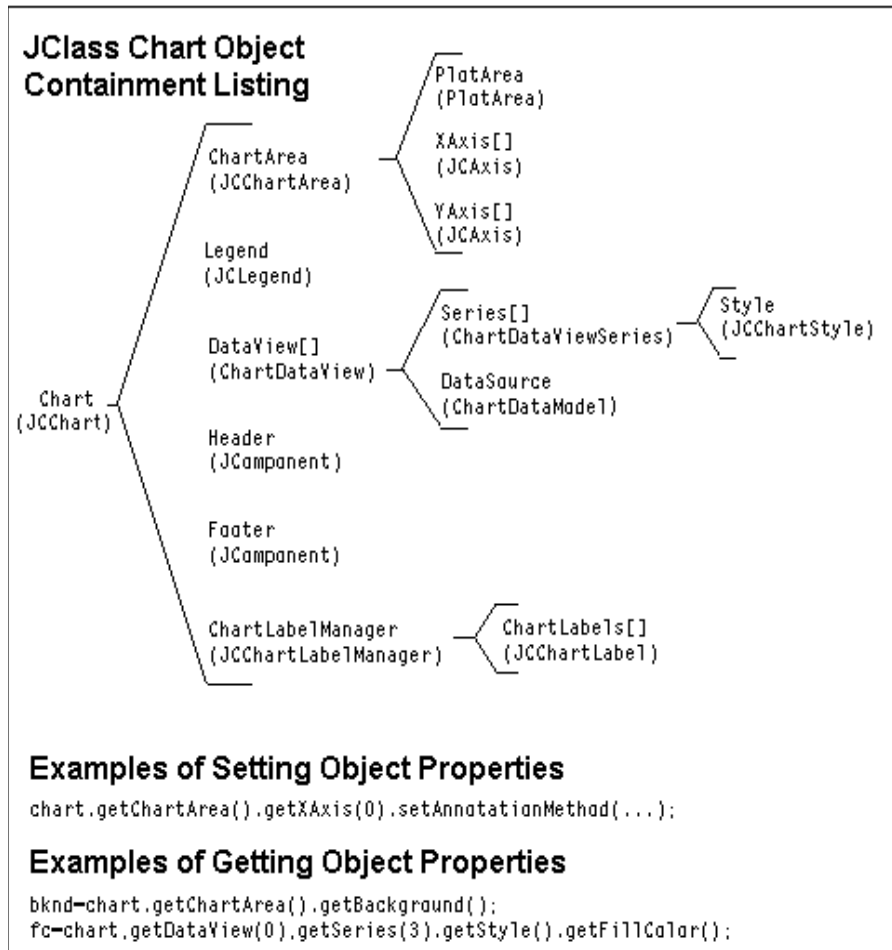


Figure 3 Objects contained in a chart – traverse contained objects to access properties.

JCChart (the top-level object) manages header and footer JComponent objects, a legend (JCLegend), and the chart area (JCChartArea). The chart also contains a collection of data

view (`ChartDataView`) objects and can contain the `ChartLabelManager` (`JCChartLabelManager`) which manages a collection of chart label (`JCChartLabel`) objects.

The chart area contains most of the chart's actual properties because it is responsible for charting the data. It also contains and manages a collection of x-axis (`JCAxis`) objects and y-axis (`JCAxis`) objects (one of each by default).

The data view collection contains objects and properties (like the chart type) that are tied to the data being charted. Each data view contains a collection of series (`ChartDataViewSeries`) objects, one for each series of data points, used to store the visual display style of each series (`JCChartStyle`).

Note that chart does not own the data itself, but instead merely views on the data. Each data view also contains a data source (`ChartDataModel`) object. The data is owned by the `DataSource` object. This is an object that your application creates and manages separately from the chart. For more information on JClass Chart's data source model, see [Data Sources](#).

## 1.9 JClass Chart Customizer

JClass Chart Customizer enables developers (or end-users if enabled by your program) to view and customize the properties of the chart as it runs.

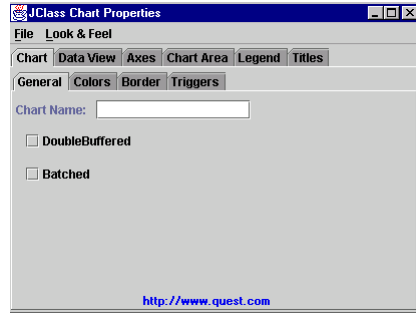


Figure 4 The JClass Chart Customizer.

JClass Chart Customizer can save developers a lot of time. Charts can be prototyped and shown to potential end-users without having to write any code. Developers can experiment with combinations of property settings, seeing results immediately in the context of a running application, greatly aiding chart debugging.

### 1.9.1 Displaying JClass Chart Customizer at Run-Time

By default, JClass Chart Customizer is disabled at run-time. To enable it, you need to set the chart's `AllowUserChanges` and `Trigger` properties, for example:

```
chart.setAllowUserChanges(true);  
chart.setTrigger(0, new EventTrigger(InputEvent.META_MASK,  
    EventTrigger.CUSTOMIZE));
```

To display JClass Chart Customizer once it has been enabled, move the mouse over the chart and click the *secondary* mouse button; that is, the button on your system that displays popup menus. (On Windows, the secondary button is usually the right mouse button, while on UNIX systems this can be the middle mouse button.)

### 1.9.2 Editing and Viewing Properties

1. Select the tab that corresponds to the chart element that you want to edit. Tabs contain one or more inner tabs that group related properties together. Select inner tabs to narrow down the type of property you want to edit.
2. If you are editing an indexed property, select the specific object to edit from the lists displayed in the tabs. The fields in the tab update to display the current values.
3. Select a property and edit its value.

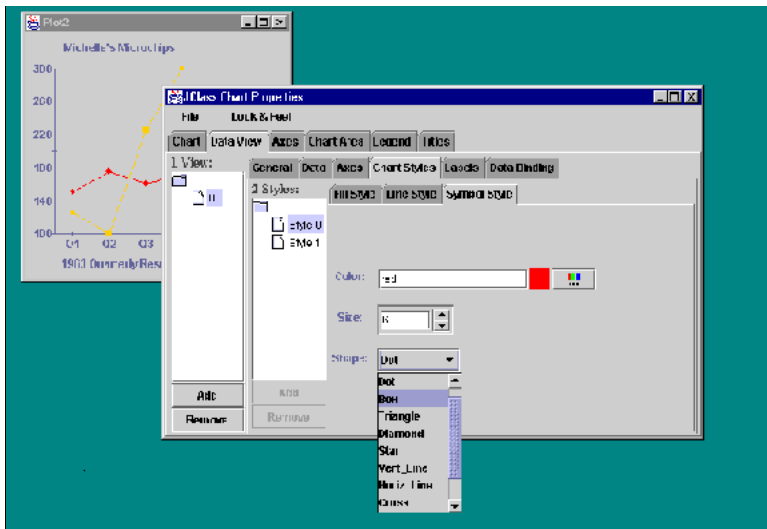


Figure 5 Editing a sample chart.

As you change property values, the changes are immediately applied to the chart and displayed. You can make further changes without leaving JClass Chart Customizer. However, once you have changed a property the only way to “undo” the change is to manually change the property back to its previous value.



### 1.9.3 Saving Chart Properties to a File

When you have finished designing your chart, you can choose to save the chart properties to an HTML file or XML file. You can then use that file to update your chart. For more information, see Chapter 10, [Using JCCChartFactory](#), Chapter 11, [Loading and Saving Charts Using HTML](#), and Chapter 12, [Loading and Saving Charts Using XML](#).

To save chart properties to a file, select **File > Save As HTML** or **File > Save As XML**.

### 1.9.4 Closing JClass Chart Customizer

To close JClass Chart Customizer, close its window.

## 1.10 Internationalization

*Internationalization* is the process of making software that is ready for adaptation to various languages and regions without engineering changes. JClass DesktopViews products have been internationalized.

*Localization* is the process of making internationalized software run appropriately in a particular environment.

In JClass DesktopViews, all Strings that may be seen by a typical user have been internationalized and are ready for localization. These Strings are in resource bundles in every package that requires them. You need to create additional resource bundles for each of the locales that you want to support.

**Note:** Localizations that are built into the Java platform – such as number and date formatting – are handled by JClass Chart, without the need for you to do any extra work.

To localize your JClass Chart, you need the JClass Chart source code (requires a source code license). The packages that require localization have a *resources* subdirectory that contains the resource bundles, called *LocaleInfo* (or some similar variation, such as *LocaleBeanInfo*). You may want to perform an automated search of the package structure to find all the resource bundles.

To create a new resource bundle, copy the *LocaleInfo.java* file (staying within the same *resources* directory) and change its name to include standard language and country identifiers for the locale that you want to support. For example, if you want to support French as spoken in France, rename the copy of *LocaleInfo.java* to *LocaleInfo\_fr\_FR.java*. You can then replace the Strings in the copied file with the French translations.

To use a localized resource bundle, you pass the language and country identifiers to the `setLocale()` method. For example, `setLocale(new Locale(fr, FR))` means that the Strings will be read from *LocaleInfo\_fr\_FR.java*.

For more information, including standard language and country identifiers, see <http://java.sun.com/j2se/1.4.2/docs/guide/intl/index.html>.

If you are creating XML-based charts, either manually or via JClass Chart Customizer, you can internationalize the text on the charts using variables and a resource bundle. For more information, see [Internationalizing Your XML-based Chart](#), in Chapter 12.

# Chart Programming Tutorial

[Introduction](#) ■ [A Basic Plot Chart](#) ■ [Loading Data From a File](#) ■ [Adding Header, Footer, and Labels](#)  
[Changing to a Bar Chart](#) ■ [Inverting Chart Orientation](#) ■ [Bar3d and 3d Effect](#)  
[End-User Interaction](#) ■ [Get Started Programming with JClass Chart](#)

## 2.1 Introduction

This tutorial shows you how to start using JClass Chart by compiling and running an example program. It is different from the SimpleChart Bean tutorial because it focuses on programmatic use of JClass Chart. For a Bean tutorial, see Chapter 14, [SimpleChart Bean Tutorial](#). This program, *Plot1.java*, will graph the 1963 Quarterly Expenses and Revenues for “Michelle’s Microchips”, a small company a little ahead of its time.

The following table shows the data to be displayed:

	Q1	Q2	Q3	Q4
Expenses	150.0	175.0	160.0	170.0
Revenue	125.0	100.0	225.0	300.0

## 2.2 A Basic Plot Chart

When *Plot1.java* is compiled and run, the window shown below is displayed:

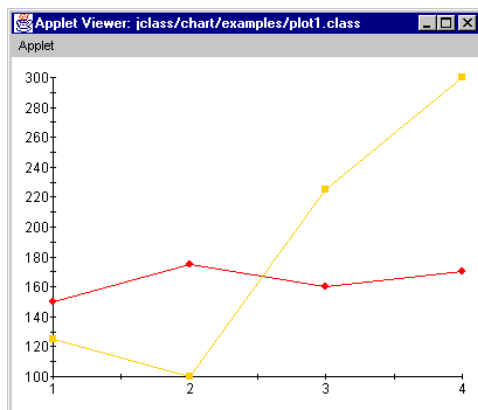


Figure 6 The *Plot1.java* program displayed.

The following listing displays the program *Plot1.java*. This is a minimal Java program that creates a new chart component and loads data into it from a file. It can be run as an applet or a standalone application.

Line	Source
1	<code>package examples.chart.intro;</code>
2	
3	<code>import java.awt.GridLayout;</code>
4	<code>import javax.swing.JPanel;</code>
5	<code>import com.klg.jclass.chart.JCChart;</code>
6	<code>import com.klg.jclass.chart.ChartDataView;</code>
7	<code>import com.klg.jclass.chart.data.JCFileDataSource;</code>
8	<code>import com.klg.jclass.util.swing.JCExitFrame;</code>
9	
10	<code>import demos.common.FileUtil;</code>
11	
12	<code>/**</code>
13	<code> * Basic example of Chart use. Load data from</code>
14	<code> * a file and displays it as a simple plot chart.</code>
15	<code> */</code>

Line	Source
16	public class Plot1 extends JPanel {
17	
18	/**
19	* Default constructor for this class. Loads data and
20	* sets up chart.
21	*/
22	public Plot1() {
23	setLayout(new GridLayout(1,1));
24	
25	// Create new chart instance.
26	JCChart chart = new JCChart();
27	// Load data for chart
28	try {
29	// Use JCFileDataSource to load data from specified file
30	String fname = FileUtil.getFullFileName(
31	"examples.chart.intro","chart1.dat");
32	chart.getDataView(0).setDataSource(new JCFileDataSource
33	(fname));
34	}
35	catch (Exception e) {
36	e.printStackTrace(System.out);
37	}
38	// Add chart to panel for display.
39	add(chart);
40	}
41	
42	public static void main(String args[]) {
43	JCExitFrame f = new JCExitFrame("Plot1");

Line	Source
44	<code>Plot1 p = new Plot1();</code>
45	<code>f.getContentPane().add(p);</code>
46	<code>f.setSize(200, 200);</code>
47	<code>f.setVisible(true);</code>
48	<code>}</code>
49	
50	<code>}</code>
51	

Most of the code in *Plot1.java* should be familiar to Java programmers. The first few lines (3–10) import the classes necessary to run *Plot1.java*. In addition to the standard AWT `GridLayout` class and Swing `JPanel` class, three classes in the `jclass.chart` package are needed: `JCChart` (the main chart class), `ChartDataView` (the data view object), and `JCFileDataSource` (a stock data source). This example also makes use of the `JCExitFrame` from `JClass Elements`, which is a part of the `JClass DesktopViews` suite. Line 16 provides the class definition for this program, a subclass of `JPanel`.

Lines 22–40 define the constructor. The `Layout` property on line 23 lays out a simple grid structure to display the components it holds. A new chart is then instantiated on line 26. Lines 30–31 load data from a file named *chart1.dat* into a new data source object (`JCFileDataSource`) and tell the chart to display this data.

Lines 42–48 define the `main()` method needed when the program is run as a standalone Java application.

## 2.3 Loading Data From a File

A common task in any `JClass Chart` program is to load the chart data into a format that the chart can use. `JClass Chart` uses a “model view/control” (MVC) architecture to handle data in a flexible and efficient manner. The data itself is stored in a object that implements the `ChartDataModel` interface created and controlled by your application. The chart has a `ChartDataView` object that controls a view on this data source, providing properties that control which data source to use, and how to display the data.

`JClass Chart` includes several stock (built-in) data sources that you can use (or you can define your own). This program uses the data source that reads data from a file: `JCFileDataSource`. With this understanding we can look more closely at lines 32–33:

```
chart.getDataView(0).setDataSource(new JCFileDataSource
    (fname));
```

Two things are happening here: a new `JCFileDataSource` object is instantiated, with the name of the data file passed as a parameter in the constructor, and the `DataSource` property of the chart's first (default) data view is being set to use this data source.

The following shows the contents of the *chart1.dat* file:

```
ARRAY 2 4
# X-values
1.0 2.0 3.0 4.0
# Y-values
150.0 175.0 160.0 170.0
# Y-values set 2
125.0 100.0 225.0 300.0
```

This file is in the format understood by `JCFileDataSource`. Lines beginning with a '#' symbol are treated as comments. The first line tells the `FileDataSource` object that the data that follows is in Array layout and is made up of two series containing four points each. The X-values are used by all series.

There are two types of data: Array and General. Use Array layout when the series of Y-values share common X-values. Use General when the Y-values do not share common X-values, or when all series do not have the same number of values.

Note that for data arrays in Polar charts,  $(x, y)$  coordinates in each data set will be interpreted as  $(\theta, r)$ . For array data, the X-array will represent a fixed theta value for each point.

In Radar and Area Radar charts, only array data can be used.  $(x, y)$  points will be interpreted in the same way as for Polar charts (above), except that the theta (that is, x) values will be ignored. The circle will be split into `nPoints` segments with `nSeries` points drawn on each radar line.

For complete details on using data with JClass Chart, see Chapter 4, [Adding Data with the Underlying Data Model](#) and Chapter 5, [Adding Data with the Targeted Data Model](#).

## 2.4 Adding Header, Footer, and Labels

The plot displayed by *Plot1.java* is not very useful to an end-user. There is no header, footer, or legend, and the X-axis numbering is not very meaningful.

The chart below displays various changes that can be made to a chart to make it more useful. The changes made to this chart are listed below. Full source code can be found in the *plot2.java* program, located in the `JCLASS_HOME/examples/chart/intro` directory.

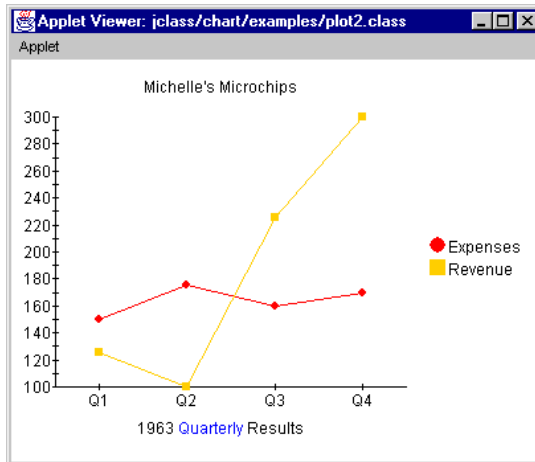


Figure 7 The program created by Plot2.java.

JClass Chart will always try to produce a reasonable chart display, even if very few properties have been specified. JClass Chart will use intelligent defaults for all unspecified properties.

All properties for a particular chart may be specified when the chart is created. Properties may also be changed as the program runs by calling the property's `set` method. A programmer can also ask for the current value of any property by using the property's `get` method.

### Adding Headers and Footers

To display a header or footer, we need to set properties of the Header and Footer objects contained in the chart. For example, the following code sets the `Text` and `Visible` properties for the footer:

```
// Make footer visible
chart.getFooter().setVisible(true);
// By default, footer is a JLabel - set its Text property
((JLabel)chart.getFooter()).setText("1963 Quarterly Results");
```

`Visible` displays the header/footer. `Text` specifies the text displayed in the header/footer.

By default, headers and footers are `JLabels`, although they can be any `Swing JComponent`. `JLabels` support the use of HTML tags. The use of HTML tags overrides the default `Font` and `Color` properties of the label.

Please note that HTML labels may not work with PDF, PS, or PCL encoding.



## Adding a Legend and Labelling Points

A legend clarifies the chart by showing an identifying label for each series in the chart. We would also like to display more meaningful labels for the points along the X-axis. Both types of information can be easily specified in the data file itself. The following lists *chart2.dat*, a modified version of the previous data file that includes series labels (for the legend), and point labels (for the X-axis):

```
ARRAY `` 2 4
# Point Labels
'Q1' 'Q2' 'Q3' 'Q4'
# X-values, with a blank series label (``) -- a blank series
# label is required if the Y-values have series labels
`` 1.0 2.0 3.0 4.0
# Y-values, with Series label (in this case, Expenses)
'Expenses' 150.0 175.0 160.0 170.0
# Y-values set 2, with Series label (in this case, Revenue)
'Revenue' 125.0 100.0 225.0 300.0
```

Lines beginning with a '#' symbol are treated as comments.

As noted in the comments within the above code, if series labels are being used for the Y-values, then the X-data must be preceded by a blank series label (''). This blank label will not show up on the chart. The third line specifies the point labels (for instance, "Q1"). Subsequent lines of data begin with a Y-data series label ("Expenses" and "Revenue").

This data file now provides the labels that we want to use, but to actually display them in the chart, we need to set the Legend object's *Visible* property and change the *AnnotationMethod* property of the X-axis to annotate the axis with the point labels in the data.

These and the previous changes are combined; now the chart is created with code that looks like this:

```
// Create new chart instance.
chart = new JCChart();
// Load data for chart
try {
    // Use JCFileDataSource to load data from specified file
    String fname = FileUtil.getFullFileName("examples.chart.intro",
        "chart2.dat");
    chart.getDataView(0).setDataSource(new
        JCFileDataSource(fname));
}
catch (Exception e) {
    e.printStackTrace(System.out);
}
// Make header visible, and add some text
chart.getHeader().setVisible(true);
// By default, header is a JLabel -- set its Text property
((JLabel)chart.getHeader()).setText("Michelle's Microchips");
// Make footer visible
chart.getFooter().setVisible(true);
// By default, footer is a JLabel -- set its Text property
((JLabel)chart.getFooter()).setText("1963 Quarterly Results");
```

```
// Make legend visible
chart.getLegend().setVisible(true);

// Make X-axis use point labels instead of default value labels.
chart.getChartArea().getXAxis(0).setAnnotationMethod
(JCAxis.POINT_LABELS);

// Add chart to panel for display.
add(chart);
```

Because we are accessing a variable defined in `JCAxis`, we need to add that to the classes imported by the program:

```
import jclass.chart.JCAxis;
```

In the line that sets the annotation method, notice that `XAxis` is a collection of `JCAxis` objects. A single chart can display several X- and Y-axes.

## 2.5 Changing to a Bar Chart

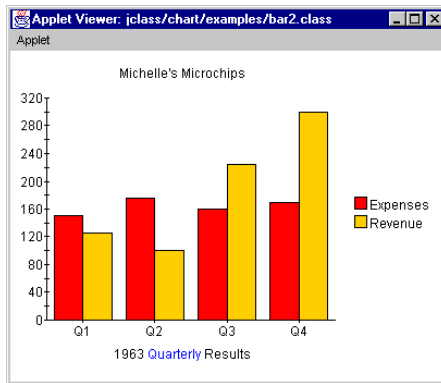


Figure 8 The `bar2.java` program displayed.

A powerful feature of `JClass Chart` is the ability to change the chart type independently of any other property. (Although there are interdependencies between some properties, most properties are completely orthogonal.) For example, to change the `Plot2` chart to a bar chart, the following code can be used:

```
c.getDataView(0).setChartType(JCChart.BAR);
```

This sets the `ChartType` property of the data view. Alternately, you can set the chart type when you instantiate a new chart, for example:

```
JCChart c = new JCChart(JCChart.BAR);
```

The full code for this program (`Bar2.java`) can be found in with the other examples. `JClass Chart` can display data in other chart types. For more information, see Chapter 3, [Selecting a Chart Type](#).

## 2.6 Inverting Chart Orientation

Most graphs display the X-axis horizontally and the Y-axis vertically. It is often appropriate, however, to invert the sense of the X- and Y-axis. This is easy to do, using the `Inverted` property of the data view object.

In a plot, inverting causes the Y-values to be plotted against the horizontal axis, and the X-values to be plotted against the vertical. In a bar chart, it causes the bars to be displayed horizontally instead of vertically.

When programming JClass Chart, try not to assume that the X-axis is always the horizontal axis. Determining which axis is vertical and which horizontal depends on the value of the `Inverted` property.

To invert, set the data view object's `Inverted` property to `true`. By default it is `false`.

```
c.getDataView(0).setInverted(true);
```

The following shows the windows created by *Plot2.java* and *Bar2.java* when inverted:

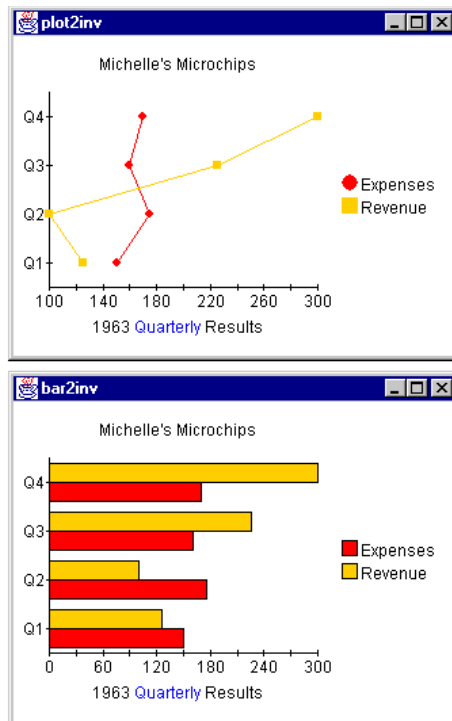


Figure 9 *Plot2* and *Bar2* windows with `Inverted` set to `true`.

Full code for these examples is in the `JCLASS_HOME/examples/chart/intro` directory.

## 2.7 Bar3d and 3d Effect

Chart 3D effects can be added to bar and stacking bar charts. Three properties affect the display of 3D information: Depth, Elevation, and Rotation. Modifying these properties will alter the 3D effects displayed. Depth and at least one of Elevation or Rotation must be non-zero to see any 3D effects. The properties can be set as follows:

```
chart.getChartArea().setElevation(20);
chart.getChartArea().setRotation(30);
chart.getChartArea().setDepth(10);
```

Function call Header for the function	Description
setDepth()  public void setDepth( int newDepth)	Controls the apparent depth of the chart; the parameter <code>newDepth</code> represents the depth as a percentage of the width; valid values are 0 to 500.
setElevation()  public void setElevation( int newElevation)	Controls the distance above the X-axis for the 3D effect; the parameter <code>newElevation</code> is the number of degrees above the X-axis that the chart is to be positioned; valid values are between -45 and 45.
setRotation()  public void setRotation( int newRotation)	Controls the position of the eye relative to the Y-axis for the 3D effect; the parameter <code>newRotation</code> is the number of degrees to the right of the Y-axis the chart is to be positioned; valid values are between -45 and 45.

## 2.8 End-User Interaction

More than simply a display tool, JClass Chart is an interactive component. Programmers can explicitly add functions that enable an end-user to directly interact with a chart. The following end-user interactions are possible:

- Translation – users can move a graph or a series of graphs along the X- and/or Y- axes.
- Rotate – users can change the vantage point of a chart type, to better view information contained with a JClass Chart component.
- Zoom – users can zoom in or out of a JClass Chart component to better view information contained within it.
- Depth – users can change the apparent depth of a 3D chart.

- Edit – users can change the placement of data points within a chart.
- Customize – users can alter the other display features of a chart, (such as color, label names, or the numerical value of data points) that comprise a chart display.
- Pick – users can determine the position of data points displayed on a chart.

## 2.9 Get Started Programming with JClass Chart

The following suggestions should help you become productive with JClass Chart as quickly as possible:

- Check out the sample code – the example and demo programs included with JClass Chart are useful in showing what JClass Chart can do, and how to do it. Run them and examine the source code. They can all be found in the *JCLASS\_HOME/demos/chart* and *JCLASS\_HOME/examples/chart* directories.
- Browse the JClass Chart [API documentation](#) – complete reference documentation on the API is available online in HTML format. The properties, methods, and events for each component are documented.



## Selecting a Chart Type

*Plot and Scatter Plot Charts* ■ *Bar and Stacking Bar Charts*  
*Area and Stacking Area Charts* ■ *Financial Charts* ■ *Pie Charts*  
*Polar Charts* ■ *Radar Charts* ■ *Area Radar Charts*

Most of the chart types share a set of common properties, such as a data view, data series, data points, axes, headers, footers, legends, and other style elements. These shared properties are described in other chapters. The purpose of this chapter is to highlight the differences among the chart types and describe how to use the properties special to one or more of the various chart types.

### 3.1 Plot and Scatter Plot Charts

In the plot-type charts, the (x,y) data points for each data series are plotted on a rectangular chart. By default, the x-axis is horizontal and the y-axis is vertical. In a plot chart, the data points in a series are connected by lines. In a scatter plot chart, the points remain unconnected.

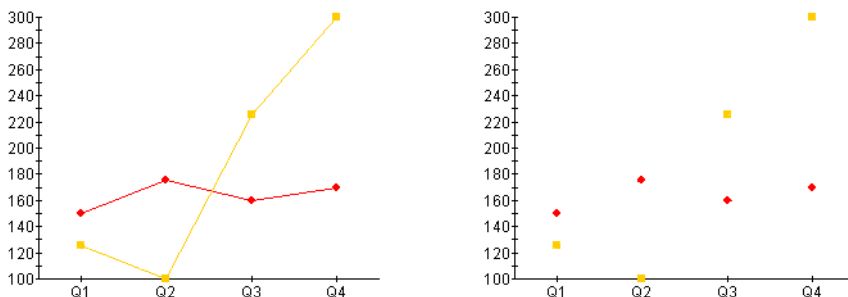


Figure 10 Sample plot and scatter plot charts.

To specify a plot chart, you use the following syntax:

```
dataView.setChartType(JCChart.PLOT);
```

To specify a scatter plot chart, you use the following syntax:

```
dataView.setChartType(JCChart.SCATTER_PLOT);
```

### Inverted Plot Chart

You can change the axes so that the y-axis is horizontal and the x-axis is vertical. To invert the axes, set the `Inverted` property of the `ChartDataView` object to `true`. To invert charts with multiple data views, set the `Inverted` property of each `ChartDataView` object.

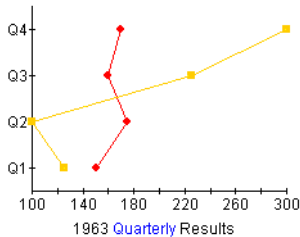


Figure 11 Inverted plot chart.

## 3.2 Area and Stacking Area Charts

In area charts, the (x,y) data points for each data series are plotted on a rectangular chart. The data points are connected with a line and the region below the line is filled in with a color or pattern. Each data series is drawn on top of the previously drawn series, with the last data series drawn appearing on top.

In a stacking area chart, the data series are stacked above one another. The stacking is accomplished by adding the y-value of the data point for the current series to the corresponding y-values of all the previously drawn data series.

The following figure shows the same data mapped on an area chart and a stacking area chart. You may notice that the blue data series visible in the stacking area chart is hidden behind other data series in the area chart.



Figure 12 Sample area chart and stacking area chart

To specify an area chart, you use the following syntax:

```
dataView.setChartType(JCChart.AREA);
```

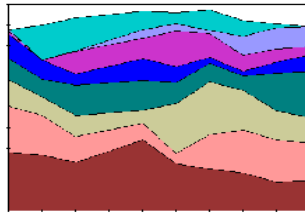
To specify a stacking area chart, you use the following syntax:

```
dataView.setChartType(JCChart.STACKING_AREA);
```



### 100-Percent Stacking Area Charts

When `100Percent` property is set to `true`, the y-axis display as an area percentage of the total. The top of the chart is 100% (the total of all y-values).



100-Percent Stacking Area Chart

Figure 13 Stacking area chart with `100Percent=true`

For example:

```
((JCAreaChartFormat)dataView.getChartFormat()).set100Percent(true)
```

## 3.3 Bar and Stacking Bar Charts

In bar charts, the  $(x,y)$  data points in a data series are represented as bars, with each series assigned a unique color or pattern. All the data series in the data view share the same set of x-values, which results in *clusters* of bars at each x-axis value. By default, the x-axis is horizontal and the y-axis is vertical. You can customize the overlap and width of clusters, as described later in this section.

In a stacking bar chart, instead of clusters of bars, the data series are stacked above one another. The stacking is accomplished by adding the y-value of the data point for the current series to the corresponding y-values of all the previously drawn data series.

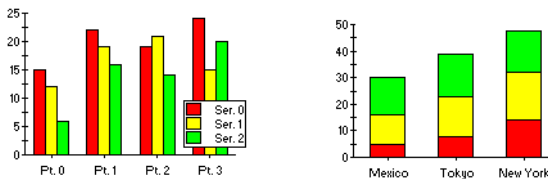


Figure 14 Sample bar and stacking bar charts.

To specify a bar chart, you use the following syntax:

```
dataView.setChartType(JCChart.BAR);
```

To specify a stacking bar chart, you use the following syntax:

```
dataView.setChartType(JCChart.STACKING_BAR);
```

## Inverted Bar Chart

If you want the bars to be presented horizontally instead of vertically, you can invert the axes. To invert the axes, set the `Inverted` property of the `ChartDataView` object to `true`. To invert charts with multiple data views, you need to set the `Inverted` property for each `ChartDataView` object.

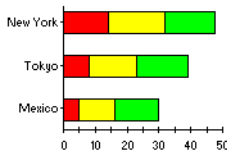


Figure 15 Inverted stacking bar chart.

## Cluster Overlap

Use the bar `ClusterOverlap` property to set the amount that bars in a cluster overlap each other. The default value is 0. The value represents the percentage of bar overlap. Negative values add space between bars and positive values cause bars to overlap. Valid values are between -100 and 100. The syntax is as follows:

```
((JCBarChartFormat)dataView.getChartFormat()).setClusterOverlap(50)
```

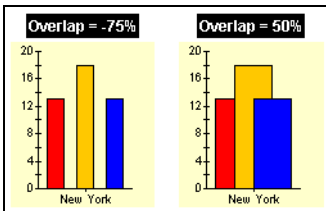


Figure 16 Negative and positive bar cluster overlap.

## Cluster Width

Use the bar `ClusterWidth` property to set the space used by each bar cluster. The default value is 80. The value represents the percentage of available space, with valid values between 0 and 100. The syntax is as follows:

```
((JCBarChartFormat)dataView.getChartFormat()).setClusterWidth(100)
```

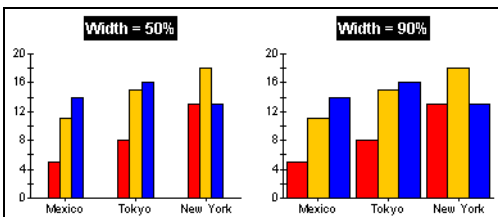


Figure 17 Setting different bar cluster widths.

### 100-Percent Stacking Bar Charts

The y-axes of stacking bar charts can display a percentage interpretation of the bar data using the `100Percent` property. When set to `true`, each stacked bar's total y-values represents 100%. The y-value of each bar is interpreted as its percentage of the total. This property has no effect on bar charts. The syntax is as follows:

```
((JCBarChartFormat)dataView.getChartFormat()).set100Percent(true)
```

## 3.4 Financial Charts

The financial charts are the Hi-Lo, Hi-Lo-Open-Close, and candle chart types. The financial chart types use the y-values in multiple data series to construct a bar. The data series share the same set of x-axis values. For a Hi-Lo chart, *two* data series are used. The first data series contains the high values while the second series represents the low values.

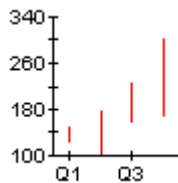


Figure 18 Sample Hi-Lo chart using a set of two data series.

In the Hi-Lo-Open-Close and candle charts, *four* data series are needed. The first two series are the high and low values respectively, the third series contains the open values, and the fourth series contains the close value.

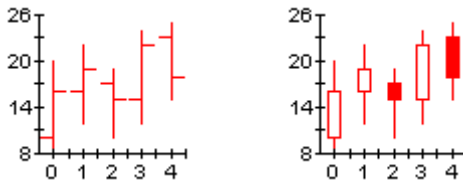


Figure 19 Sample Hi-Lo-Open-Close and candle charts using a set of four data series.

To specify a Hi-Lo chart, you use the following syntax:

```
dataView.setChartType(JCChart.HILO);
```

To specify a Hi-Lo-Open-Close chart, you use the following syntax:

```
dataView.setChartType(JCChart.HILO_OPEN_CLOSE);
```

To specify a candle chart, you use the following syntax:

```
dataView.setChartType(JCChart.CANDLE);
```

### Hi-Lo-Open-Close Charts

When the chart type is `JCChart.HILO_OPEN_CLOSE`, several properties defined in `JCHLOCCChartFormat` control how open and close ticks are displayed:

<code>ShowingOpen</code>	Displays or hides open tick marks.
<code>ShowingClose</code>	Displays or hides close tick marks.
<code>OpenCloseFullWidth</code>	Displays open/close ticks across both sides of the bar. This is useful for creating error bar charts.

### Customizing ChartStyles

Because these chart types use multiple series for each “row” of Hi-Lo or candle bars, it is difficult to determine which chart style specifies the display attributes of a particular row of bars. To make programming the chart styles of financial charts easier, `JClass Chart` provides several methods that retrieve and set the style for a *logical* series. These methods are defined in the `JCHiloChartFormat`, `JCHLOCCChartFormat` and `JCCandleChartFormat` classes. Each `get` method returns the `JCChartStyle` object used for the logical series you specify. You can customize the properties in this returned object and then use the appropriate `set` method to apply them to the same logical series in the chart.

Most of the financial chart types use only one or two `JCChartStyle` properties. The following table lists the properties used by each chart type (see [Chart Styles](#), in Chapter 7 for more information on chart styles):

	LineColor	SymbolSize
Hi-Lo	✓	
Hi-Lo-Open-Close	✓	✓
Candle (simple)	✓	✓
Candle (complex)	see below	

For every financial chart type except complex candle, the actual chart style used is that of the *first* series.

### Simple and Complex Candle Charts

You can choose between a simple and complex candle chart display using the `Complex` property defined in `JCCandleChartFormat`.

When set to `false`, the chart style from just *one* series (the first) determines the appearance of the candle. The table above shows the properties used. A rising stock price is indicated by making the candle transparent. A falling stock price displays in the color specified by `FillColor`.

Complex candle charts (`Complex` is `true`), use elements of the chart styles of all *four* series, providing complete control over every visual aspect of the candles. The convenience methods defined in `JCCandleChartFormat` make it easy to retrieve/set the style that controls the appearance of a particular aspect of the candles.

The following lists the `JCChartStyle` properties that control each aspect of a complex candle, along with which of the four chart styles is used:

- Hi-Lo line – `LineColor` property (first chart style)
- Rising price candle color and width – `FillColor` and `SymbolSize` properties (second chart style)
- Falling price candle color and width – `FillColor` and `SymbolSize` properties (third chart style)
- Candle outline – `LineColor` property (fourth chart style)

### Example Code

The following code sets the rising and falling candle styles of a complex candle chart:

```
JCChartStyle chartStyle;
JCCandleChartFormat candleFormat;

// Set candle to complex type so we can change colors
candleFormat=(JCCandleChartFormat)chart.getDataView(1).getChartFormat();
candleFormat.setComplex(true);

// Change rising candle color
chartStyle = candleFormat.getRisingCandleStyle(0);
chartStyle.setLineColor(Color.green);
chartStyle.setFillColor(Color.red);

// Change falling candle color
chartStyle = candleFormat.getFallingCandleStyle(0);
chartStyle.setLineColor(Color.green);
chartStyle.setFillColor(Color.yellow);
```

Two demo programs included with `JClass Chart` illustrate creating financial charts: the stock demo, located in `JCLASS_HOME/demos/chart/stock/`, and the financial demo, located in `JCLASS_HOME/demos/chart/financial`.

## 3.5 Pie Charts

Pie charts are quite different from the other chart types. They do not have the concept of a two-dimensional grid or axes. Data points in a data series are represented as slices in a pie. Each data series in the chart data view is captured in a separate pie.



Figure 20 Sample pie charts displaying one data series and many data series.

To specify a pie chart, you use the following syntax:

```
dataView.setChartType(JCChart.PIE);
```

You can customize your pie charts with the properties of `JCPieChartFormat`. The following code snippet shows the syntax for setting `JCPieChartFormat` properties:

```
JCPieChartFormat pcf = (JCPieChartFormat) arr.getChartFormat();
pcf.setOtherLabel("Other Bands");
pcf.setThresholdValue(10.0);
pcf.setThresholdMethod(JCPieChartFormat.PIE_PERCENTILE);
pcf.setSortOrder(JCPieChartFormat.DATA_ORDER);
pcf.setStartAngle(90.0);
```

### Building the “Other” Slice

Pie charts introduce a special category called “Other”, into which all data values below a certain threshold can be grouped.

Pie charts are often more effective if unimportant values are grouped into an “Other” category. Use the `ThresholdMethod` property to select the grouping method to use. `SLICE_CUTOFF` is useful when you know the data value that should be grouped into the “Other” slice. `PIE_PERCENTILE` is useful when you want a certain percentage of the pie to be devoted to the “Other” slice.

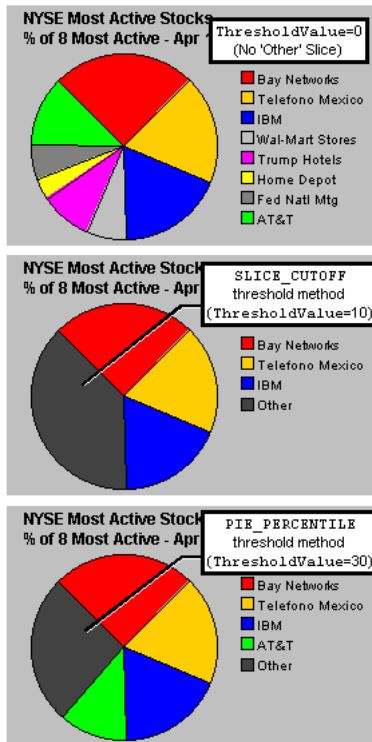


Figure 21 Three JClass Charts illustrating how the “Other” slice can be used.

Use the `MinSlices` property to fine-tune the number of slices displayed before the “Other” slice. For example, when set to 5, the chart tries to display 5 slices before grouping data into the “Other” slice.

### “Other” Slice Style and Label

The `OtherStyle` property allows access to the `ChartStyle` used to render the “Other” slice. Use `FillStyle`’s `Pattern` and `Color` properties to define the appearance of the Other slice.

Use the `OtherLabel` property to change the label of the “Other” slice.

### Pie Ordering

Use the `SortOrder` property to specify whether to display slices largest-to-smallest, smallest-to-largest, or the order they appear in the data.

## Start Angle

The position in the pie chart where the first pie slice is drawn can be specified with the `StartAngle` property. A value of zero degrees represents a horizontal line from the center of the pie to the right-hand side of the pie chart; a value of 90 degrees represents a vertical line from the center of the pie to the top-most point of the pie chart; a value of 180 degrees represents a horizontal line from the center of the pie to the left-hand side of the pie chart; and so on. Slices are drawn clockwise from the specified angle. Values must lie in the range from zero to 360 degrees. The default value is 135 degrees.

## Exploded Pie Slices

It is possible to have individual slices of a pie “explode” (that is, detach from the rest of the pie). Exploded slices can be used in both 2D and 3D pie charts. Two properties of `JCPieChartFormat` are responsible for this function: `ExplodeList` and `ExplodeOffset`.

`ExplodeList` specifies a list of exploded pie slices in the pie charts. It takes *pts* as a parameter, which is composed of an array of `Point` objects. Each point object contains the data point index (pie number) in the x-value and the series number (slice index) in the y-value, specifying the pie slice to explode. To explode the “other” slice, the series number should be `OTHER_SLICE`. If null, no slices are exploded.

`ExplodeOffset` specifies the distance a slice is exploded from the center of a pie chart. It takes off as a parameter, which is the explode offset value.

The following code sample shows how `ExplodeList` and `ExplodeOffset` can be used to set the list of exploded slices.

```
Point[] exList = new Point[3];
exList[0] = new Point(0, 0);
exList[1] = new Point(1, 5);
exList[2] = new Point(2, JCPieChartFormat.OTHER_SLICE);
pcf.setExplodeList(exList);
pcf.setExplodeOffset(10);
```

The following code sample shows how to set up a pick listener such that when a user clicks on an individual pie slice, that slice explodes (and then implodes if the user clicks on it again):

```
public void pick(JCPickEvent e)
{
    JCDataIndex di = e.getPickResult();
    if (di == null) return;
    Object obj = di.getObject();
    ChartDataView vw = di.getDataView();
    ChartDataViewSeries srs = di.getSeries();
    int slice = di.getSeriesIndex();
    int pt = di.getPoint();
    int dist = di.getDistance();
```



```

    if (vw != null && slice != -1) {
        JCPieChartFormat pcf = (JCPieChartFormat)vw.getChartFormat();
        Point[] exList = pcf.getExplodeList();
        if (exList == null) return;
        // implode existing exploded slices
        for (int i = 0; i < exList.length; i++) {
            if ((exList[i].x == pt) && (exList[i].y == slice)) {
                Point[] newList = new Point[exList.length - 1];
                for (int j = 0; j < i; j++)
                    newList[j] = exList[j];
                for (int j = i; j < newList.length; j++)
                    newList[j] = exList[j + 1];
                pcf.setExplodeList(newList);
                return;
            }
        }
        // explode new slice
        Point[] newList = new Point[exList.length + 1];
        for (int j = 0; j < exList.length; j++)
            newList[j] = exList[j];
        newList[exList.length] = new Point(pt, slice);
        pcf.setExplodeList(newList);
    }
}

```

The full code for this program can be found in *JCLASS\_HOME/examples/chart/interactions/*. For more information on pick, see [Using Pick and Unpick](#), in Chapter 9.

### Saving and Loading Exploding Pie Slices

Exploded pie slice properties can be saved or loaded to or from both XML and HTML. This is done by passing JCPieChartFormat's `setExplodeList()` method an array of `Point` objects which correspond to the exploded series and points. For each `Point` object in the array, the *X* value represents the pie (or point) number, while the *Y* value represents the slice (or series) number. To specify all of the points or series, use the `ALL` integer; to specify that the “other” slice should be exploded, use `other` as the *Y* value.

For more information on using this feature in XML, see [pie-format](#) in Appendix C, [XML DTD](#).

In HTML, the code should resemble the following:

```

<APPLET CODEBASE="../../../" ARCHIVE="lib/jcchart.jar" WIDTH=450 HEIGHT=300
CODE="com/klg/jclass/chart/applet/JCChartApplet.class">
<PARAM name="dataFile" value="sample_1.dat">
<PARAM name="data.chartType" value="PIE">
<PARAM name="data.pie.explodeList" value="0,all|all,1|3,3|4,other">
</APPLET>

```

where all slices on the first pie are exploded (0,ALL), the slices corresponding to the first dataseries are exploded on all pies (ALL,1), the slice corresponding to the third dataseries is exploded on the fourth pie (3,3), and the fifth pie's “other” slice is also exploded (4,other). Note that the pie (or point) number starts at 0; therefore, the first pie is 0, the second is 1, and so on.

## 3.6 Polar Charts

In polar charts, the  $(x,y)$  data points for each series are drawn as  $(\theta, r)$ , where  $\theta$  is amount of rotation from the x-origin and  $r$  is the distance from the y-origin.  $\theta$  can be specified in degrees (default), radians, or gradians. Because the x-axis is a circle, the x-axis maximum and minimum values are fixed.

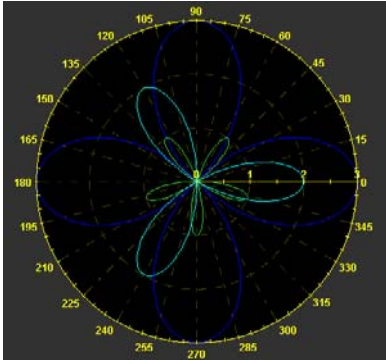


Figure 22 Sample polar chart.

To specify a polar chart, you use the following syntax:

```
dataView.setChartType(JCChart.POLAR);
```

### Background Information for the Polar Charts

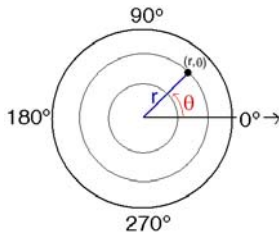
In order to work efficiently with polar charts, you should understand the following basic concepts.

#### Theta

Theta ( $\theta$ ), which is the angle from the x-axis origin, is measured in a counterclockwise direction. In cartesian (rectangular) X and Y plots, theta “translates” to the x-axis.

#### r value

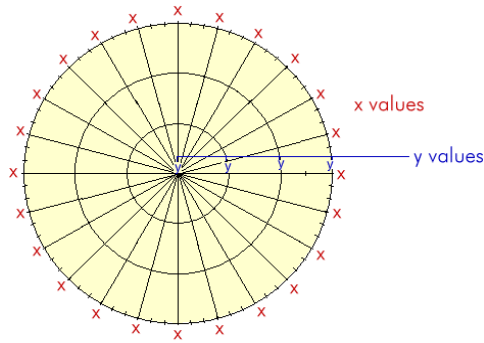
$r$  represents the distance from the y-axis origin. In cartesian (rectangular) X and Y plots,  $r$  “translates” to the y-axis. Multiple  $r$  values are allowed.



## Angles

Angles can be measured in degrees, radians, or gradients.

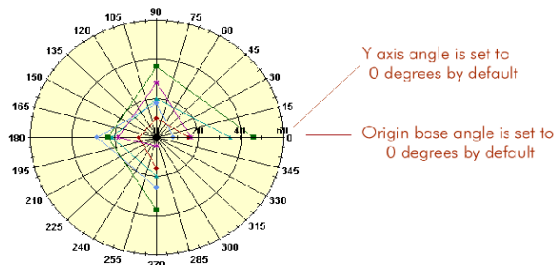
### X- and Y-values in Polar Charts



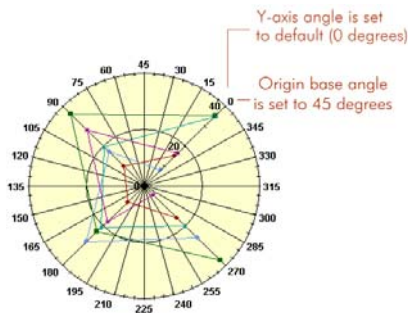
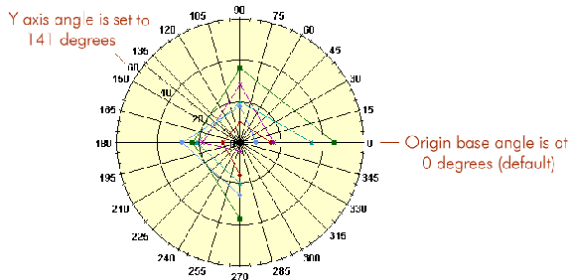
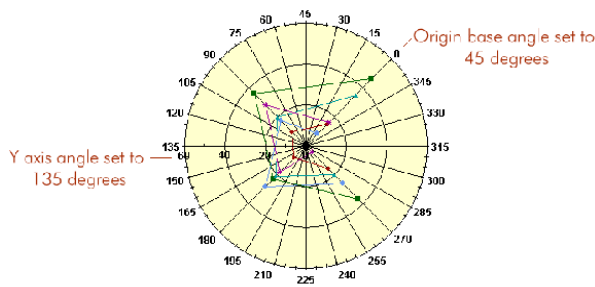
### Setting the Origin

All angles are relative to the origin base angle. The position of the x-axis origin is determined by the origin base angle. The `OriginBase` property is a value between 0 and 360 degrees (if the angle unit is degrees).

The y-axis angle is the angle that the y-axis makes with the origin base. The origin base angle is set to  $0^\circ$  by default. The y-axis angle is set to  $0^\circ$  to the origin base by default.



You can change the origin base angle, the y-axis angle, or both.



## Data Format

In the underlying data model, the data format for polar charts is either:

- general –  $(x,y)$  for every series; or
- array (only one x-value).

The x-array contains the theta values; the y-array contains the r-values. For array data, the x-array represents a fixed theta value for each point. For more information, see [Text Data Formats](#), in Chapter 4.

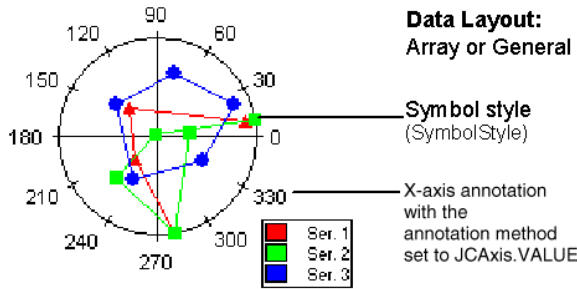


Figure 23 Data format for polar charts.

### PolarChartDraw Class

The `PolarChartDraw` class (which extends `ChartDraw`) is a drawable object for polar charts. This object is used for rendering a polar chart based on data contained in the `dataObject`.

The default constructor is `PolarChartDraw()`.

There are two key methods in this class:

- `recalc()` – recalculates the extents of related objects
- `draw()` – draws related objects and takes as its parameter the graphics context to use for drawing

### Full or Half-Range X-Axis

Use the `HalfRange` property to determine whether the x-axis is displayed as one full range from 0 to 360 degrees (*default*) or two half-ranges: from –180 degrees to zero degrees to 180 degrees. In interval notation the range would be `[0,360]` when `HalfRange` is `false` and `(-180, 180]` when `HalfRange` is `true`.

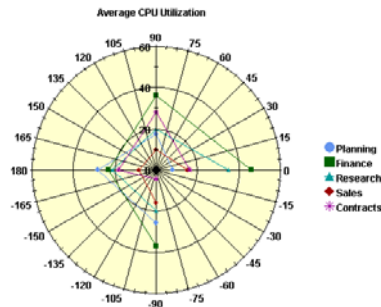


Figure 24 Half-range is true – values in the lower half of the chart are negative.

## Allowing Negative Values

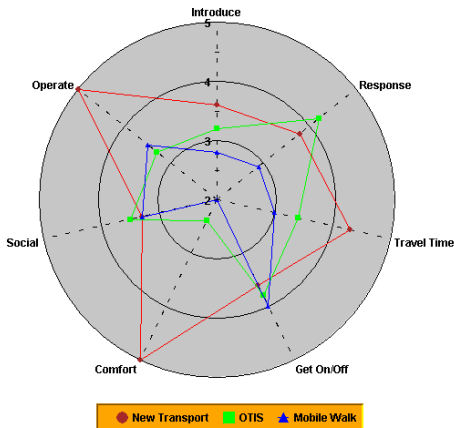
Polar charts do not allow negative values for the y-axis unless the y-axis is reversed. A negative radius is interpreted as a positive radius rotated 180 degrees.

Thus  $(\theta, r) = (\theta + 180, -r)$ .

## 3.7 Radar Charts

A radar chart plots data as a function of distance from a central point. A line connects the data points for each series, forming a polygon around the chart center.

A radar chart draws the y-value in each data set along a radar line (the x-value is ignored). If the data set has  $n$  points, then the chart plane is divided into  $n$  equal angle segments, and a radar line is drawn (representing each point) at  $360/n$  degree increments. By default, the radar line representing the first point is drawn horizontally (at 0 degrees).



Radar charts permit easy visualization of symmetry or uniformity of data, and are useful for comparing several attributes of multiple items. Although radar charts look as if they have multiple y-axes, they have only one; hence, you cannot change the scale of just one spoke.

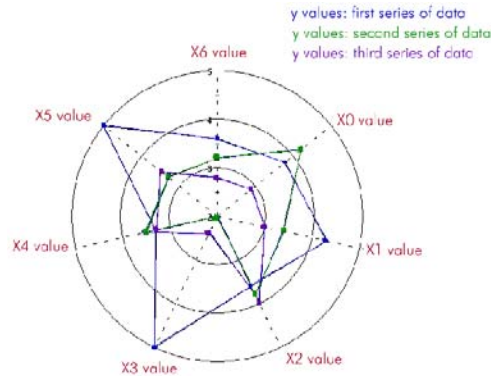
To specify a radar chart, you use the following syntax:

```
dataView.setChartType(JCChart.RADAR);
```

The `JCPolarRadarChartFormat` class provides methods to get or set properties specific to polar, radar, or area radar charts. Using `ChartStyles`, you can customize the symbol and line properties of each series. For more information, see [Chart Styles](#), in Chapter 7.

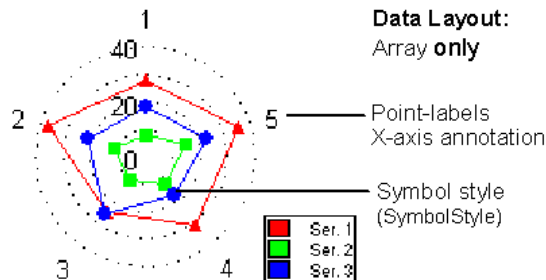
## Background Information for Radar Charts

An example of the x-values and y-values of a radar chart is shown below; in this case, there are seven x-values and three series of y-values.



## Data Format

In the underlying data model, the data format required for radar charts is array only. For more information, see [Text Data Formats](#), in Chapter 4.



## RadarChartDraw Class

The `RadarChartDraw` class (which extends `PolarChartDraw`) is a drawable object for radar charts. This object is used for rendering a radar chart based on data contained in the `dataObject`.

The default constructor is `RadarChartDraw()`.

There are two key methods in this class:

- `recalc()` – recalculates the extents of related objects; and
- `draw()` – draws related objects and takes as its parameter the graphics context to use for drawing.

## 3.8 Area Radar Charts

An area radar chart draws the y-value in each data set along a radar line (the x-value is ignored). If the data set has  $n$  points, the chart plane is divided into  $n$  equal angle segments, and a radar line is drawn (representing each point) at  $360/n$  degree increments. Each series is drawn “on top” of the preceding series.

Area radar charts are the same as radar charts, except that the area between the origin and the points is filled.

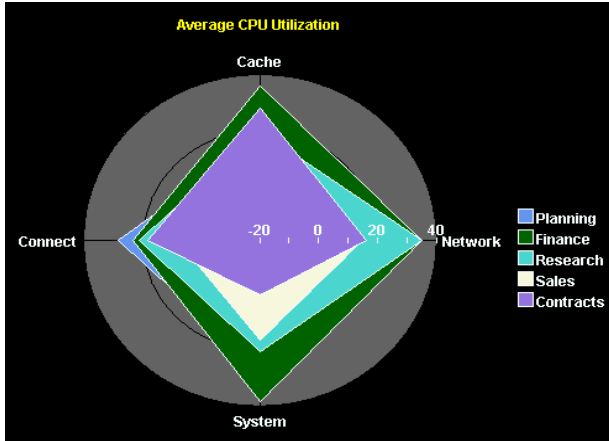


Figure 25 Sample area radar chart.

To specify an area radar chart, you use the following syntax:

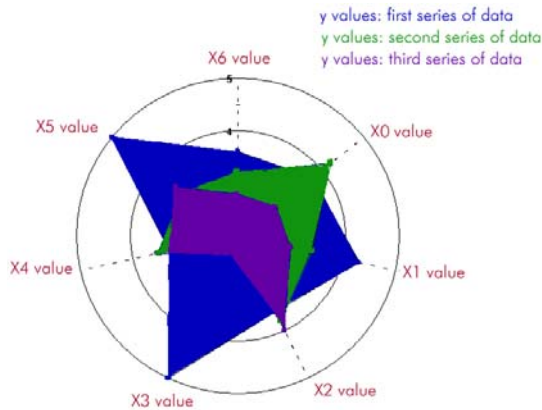
```
dataView.setChartType(JCChart.AREA_RADAR);
```

The `JCPolarRadarChartFormat` class provides methods to get or set properties specific to polar, radar, or area radar charts. Using `ChartStyles`, you can customize the fill and line properties of each series. For more information, see [Chart Styles](#), in Chapter 7.

### Background Information for Area Radar Charts

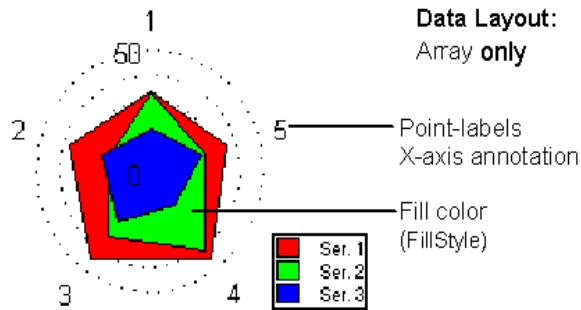
An example of the x- and y-values of an area radar chart is shown below; in this case, there are seven x-values and three series of y-values.





### Data Format

In the underlying data model, the data format required for area radar charts is array only. For more information, see [Text Data Formats](#), in Chapter 4.



### AreaRadarChartDraw Class

The `AreaRadarChartDraw` class (which extends `RadarChartDraw`) is a drawable object for area radar charts. This object is used for rendering an area radar chart based on data contained in the `dataObject`. The default constructor is `AreaRadarChartDraw()`.



# Adding Data with the Underlying Data Model

*Understanding the Underlying Data Model ■ Pre-Built Chart DataSources ■ Loading Data from a File  
Loading Data from a URL ■ Loading Data from an Applet ■ Loading Data from a Swing TableModel  
Loading Data from an XML Source ■ Text Data Formats  
Making Your Own Chart Data Source ■ Making an Updating Chart Data Source*

After you select the type of chart that you want to create, the next step is to add your data to the chart. The underlying data model can be used to add data to all JClass Chart applications. It is also the model of choice if your data is dynamic and needs to be updated frequently.

**Important:** If your data is stored as a JDBC result set and you are creating your chart programmatically, you should consider using the targeted data model instead. For more information, see [Adding Data](#), in Chapter 1 and Chapter 5, [Adding Data with the Targeted Data Model](#).

## 4.1 Understanding the Underlying Data Model

Data sources are added to JClass Chart using *data views*, which are encapsulated by the `ChartDataView` object. `ChartDataView` organizes data as a collection of `ChartDataViewSeries` objects, one `ChartDataViewSeries` for each series of data points.

In most cases, your charts will require only one data view. However, JClass Chart allows you to load data from multiple data sources at the same time, assigning each source to a separate data view. By default, all data views are showing, but each may be hidden or revealed depending on the needs of your application. Data views may be mapped to the same set of x-axes and y-axes, or to different axes.

**Note:** Radar, area radar, and pie charts do not support multiple data views.

## 4.2 Pre-Built Chart DataSources

JClass Chart provides pre-built `DataSource` objects that you can use to load data. They are located in the `com.klg.jclass.chart.data` package.

The following table summarizes the available `DataSource` objects:

DataSource name	Description
<code>BaseDataSource</code>	A very simple container for chart data.
<code>JCAppletDataSource</code>	Used to load data from an applet parameter tag.
<code>JCChartSwingDataSource</code>	Used to extract data from a Swing <code>TableModel</code> .
<code>JCDefaultDataSource</code>	An extension of <code>BasicDataSource</code> .
<code>JCEditableDataSource</code>	An editable version of <code>JCDefaultDataSource</code> .
<code>JCFileDataSource</code>	Used to load data from a file.
<code>JCInputStreamDataSource</code>	Used to load data from any stream.
<code>JCStringDataSource</code>	Used to load data from a <code>String</code> .
<code>JCURLDataSource</code>	Used to load data from a URL.
<code>JDBCDataSource</code>	Used to load data from a JDBC Result Set.

## 4.3 Loading Data from a File

An easy way to bring data into a chart is to load it from a formatted file using `JCFileDataSource`. To load data this way, you create a data file that follows JClass Chart's standard format, as outlined in [Section 4.8, Text Data Formats](#).

To finish, you instantiate a `JCFileDataSource` object and attach it to a view in your chart application. The following example shows how to instantiate and attach a `JCFileDataSource`:

```
chart.getDataView(0).setDataSource(new JCFileDataSource("file.dat"));
```

## 4.4 Loading Data from a URL

You can chart data from a URL address using `JCURLDataSource`. To load data this way, you create a data file that follows JClass Chart's standard format, as outlined in [Section 4.8, Text Data Formats](#).

Then, you instantiate `JCURLDataSource` and attach it to a view in your chart. The following example uses data from a file named *plot1.dat*:

```
chart.getDataView(0).setDataSource(new
    JCURLDataSource(getDocumentBase(), "plot1.dat"));
```

**Parameter options for `JCURLDataSource`:**

The following are valid parameter combinations for `JCURLDataSource`:

- URL
- base, file
- host, file

**host:** The WWW hostname.

**file:** The fully qualified name of the file on the server.

**URL:** The URL address of a data file, eg, *http://www.quest.com/datafile.dat*.

**base:** A URL object representing the directory where the file is located.

In the example above, the first parameter passed is `getDocumentBase()`, a method that returns the path where the current applet is located.

## 4.5 Loading Data from an Applet

You can chart data from an applet using `JCAppletDataSource`.

To prepare the data, put it into the standard format, (see [Data Formats](#)), and insert it into the HTML file that calls your applet. The HTML syntax is as follows:

```
<Applet>
...
<PARAM NAME=Your_Data_Name VALUE=" ....formatted data... ">
...
</Applet>
```

“Your\_Data\_Name” is used by your applet to select the right set of information. Use the same name in the applet and the HTML source. If a name is not provided “data” is assumed.

With your data in the HTML file, instantiate an `JCAppletDataSource` and attach it to a view in your chart as follows:

```
chart.getDataView(0).setDataSource(new JCAppletDataSource(applet,
    "Your_Data_Name"));
```

You can also chart data from an HTML file. For a listing of the syntax of `JClass Chart` properties when specified in an HTML file, please see Appendix B, [HTML Syntax](#).

### Example of Data in an HTML file

```
<APPLET CODEBASE="../../../.."
CODE="jclass/chart/demos/labels/labels.class"

<PARAM NAME=data VALUE="

  ARRAY 'Oblivion Inc. 1996 Results' 2 4
      'Q1'      'Q2'      'Q3'      'Q4'
  'Quarter'    1          2          3          4
  'Expenses'   150.2     182.1     152.1     170.6
  'Revenue'    125.5     102.7     225.0     300.9
">
</APPLET>
```

## 4.6 Loading Data from a Swing TableModel

The `JCChartSwingDataSource` class enables you to use any type of `Swing TableModel` data object for the chart. `TableModel` is typically used for `Swing JTable` components, so your application may already have created this type of data object.

`JCChartSwingDataSource` “wraps” around a `TableModel` object, so that the data appears to the chart in the format it understands.

This data source is available through the `SwingDataModel` property in the `SimpleChart` and `MultiChart` Beans. To use it, prepare your data in a `Swing TableModel` object and set the `SwingDataModel` property to that object.

## 4.7 Loading Data from an XML Source

For more general XML information, refer to Chapter 12, [Loading and Saving Charts Using XML](#).

`JClass Chart` can accept XML data formatted to the specifications outlined in `com.klg.jclass.chart.data.JCXMLDataInterpreter`. This public class extends `JCDataInterpreter` and implements an interpreter for the `JClass Chart` XML data format. `JCXMLDataInterpreter` relies on an input stream reader to populate the specified `BaseDataSource` class.

Data can be specified either by series or by point. This is fully explained below.

### Examples of XML in JClass

For XML data source examples, see the *XMLArray*, *XMLArrayTrans*, and *XMLGeneral* examples in `JCLASS_HOME/examples/chart/datasource`. These use the *array.xml*, *arraytrans.xml*, and *general.xml* data files, respectively.

## Interpreter

The interpreter, which converts incoming data to the internal format used by JClass Chart, must be explicitly set by the user when loading XML-formatted data. The interpreter to use for this purpose is `com.klg.jclass.chart.data.JCXMLDataInterpreter`.

Many constructors in the various data sources in JClass Chart take the abstract class `JCDataInterpreter`, which is extended by `JCXMLDataInterpreter`. It is possible for the user to create a custom data format and a custom data interpreter by extending `JCDataInterpreter`.

Here are a few code examples that load XML data using JClass Chart's XML interpreter, `JCXMLDataInterpreter`:

```
ChartDataModel cdm = new JCFileDataSource(fileName, new
JCXMLDataInterpreter());
ChartDataModel cdm = new JCURLDataSource(codeBase, fileName, new
JCXMLDataInterpreter());
ChartDataModel cdm = new JCStringDataSource(string, new
JCXMLDataInterpreter());
```

### 4.7.1 Specifying Data by Series

When “specifying by series”, there can be any number of `<data-series>` tags. Within each `<data-series>` tag, there can be an optional `<data-series-label>` tag and any number of `<x-data>` tags (these tags represent the x-values for that series). If there are no `<x-data>` tags in any `<data-series>` tag, a single x-array is generated, starting at 1 and proceeding in increments of 1.

If only one series has `<x-data>` tags, then that list of x-data is used for all series. If more than one series has `<x-data>` tags, those tags are used only for the series in which they are located.

Within each `<data-series>` tag, there must be at least one `<y-data>` tag (generally there will be many). `<y-data>` tags represent the y-values for that series.

If the number of x-values and y-values do not match within one series, the one with the fewer number of values is padded out with `Hole` values.

Here is an example of an XML data file specifying data by series.

```
<?xml version="1.0"?>
<!DOCTYPE chart-data SYSTEM "JCChartData.dtd">
<chart-data Name="My Chart" Hole="MAX">
  <data-point-label>Point Label 1</data-point-label>
  <data-point-label>Point Label 2</data-point-label>
  <data-point-label>Point Label 3</data-point-label>
  <data-point-label>Point Label 4</data-point-label>
  <data-series>
    <data-series-label>Y Axis #1 Data</data-series-label>
    <x-data>1</x-data>
    <x-data>2</x-data>
    <x-data>3</x-data>
```

```

        <x-data>4</x-data>
        <y-data>1</y-data>
        <y-data>2</y-data>
        <y-data>3</y-data>
        <y-data>4</y-data>
    </data-series>
</data-series>
    <data-series-label>Y Axis #2 Data</data-series-label>
    <y-data>1</y-data>
    <y-data>4</y-data>
    <y-data>9</y-data>
    <y-data>16</y-data>
</data-series>
</chart-data>

```

This format is similar to both the array and the general formats of the default chart data source.

### 4.7.2 Specifying Data by Point

In the “specifying by point” format, there can be any number of `<data-point>` tags. Within each `<data-point>` tag, there can be one optional `<data-point-label>` tag and one optional `<x-data>` tag (these tags represent the x-value of that point). If there are no `<x-data>` tags in any of the `<data-point>` tags, x-values are generated, starting at 1 and then increasing in increments of 1.

If some `<data-point>` tags have `<x-data>` tags but others do not, the missing ones will be replaced with `None` values.

Within each `<data-point>` tag, there must be at least one `<y-data>` tag (in general, there will be many). `<y-data>` tags represent the y-values of each series at this point.

There should always be the same number of `<y-data>` tags within each `<data-point>` tag. If there are not, then the largest number of `<y-data>` tags in any one `<data-point>` tag is used as the number of series, and the other lists of y-values will be padded with `None` values.

Here is an example of an XML data file specifying data by point.

```

<?xml version="1.0"?>
<!DOCTYPE chart-data SYSTEM "JCChartData.dtd">
<chart-data Name="MyChart">
  <series-label>Y Data</series-label>
  <series-label>Y 2 Data</series-label>
  <data-point>
    <data-point-label>Point Label 1</data-point-label>
    <x-data>1</x-data>
    <y-data>1</y-data>
    <y-data>1</y-data>
  </data-point>
  <data-point>
    <data-point-label>Point Label 2</data-point-label>
    <x-data>2</x-data>

```



```

        <y-data>2</y-data>
        <y-data>4</y-data>
    </data-point>
    <data-point>
        <data-point-label>Point Label 3</data-point-label>
        <x-data>3</x-data>
        <y-data>3</y-data>
        <y-data>9</y-data>
    </data-point>
    <data-point>
        <data-point-label>Point Label 4</data-point-label>
        <x-data>4</x-data>
        <y-data>4</y-data>
        <y-data>16</y-data>
    </data-point>
</chart-data>

```

This format is similar to the transposed array format of the default chart data source.

### 4.7.3 Labels and Other Parameters

#### <data-point-label> and <data-series-label> Tags

<data-point-label> and <data-series-label> tags are optional with both the specifying by series or specifying by point methods. If there are more point labels than data points, or more series labels than data series, the extra labels are ignored. If there are more data points than point labels, or more data series than series labels, then the list is padded with blank labels. If there are no point labels or no series labels at all, the chart default is used – no point labels and series labels containing “Series 1”, “Series 2”, and so on.

#### Name and Hole Parameters

The *Name* and *Hole* parameters of the `chart-data` tag are also optional. *Name* can be any String. *Hole* can be a value, the String `MIN` (meaning `Double.MIN_VALUE`), or the String `MAX` (meaning `Double.MAX_VALUE`). To represent virtual hole values in an `x-data` or `y-data` tag, use the word `Hole`. Any `x-data` or `y-data` tag can contain a value, the String `MIN`, the String `MAX`, or the String `Hole`.

See the Section 4.7.1, [Specifying Data by Series](#), and Section 4.7.2, [Specifying Data by Point](#), to view these elements in code samples.

## 4.8 Text Data Formats

When specifying data using ASCII text, `JCApletDataSource`, `JCFileDataSource`, `JCURLDataSource`, `JCInputStreamDataSource`, and `JCStringDataSource`, all require that data be pre-formatted. The following table illustrates the formatting requirements of data for pre-built data sources. There are two main ways to format data: Array and General.

Array-formatted data shares a single series of x-data among one or more series of y-data. General-formatted data specifies a series of x-data for every series of y-data.

Array format is the recommended standard, because it works well with all of the chart types. **General Format may not display data properly in stacking bar, stacking area, pie, and bar charts.**

Note that for data arrays in polar charts,  $(x, y)$  coordinates in each data set will be interpreted as  $(\theta, r)$ . For array data, the x-array will represent a fixed theta value for each point.

In radar and area radar charts, only array data can be used.  $(x, y)$  points will be interpreted in the same way as for polar charts (above), except that the theta (that is, X) values will be ignored. The circle will be split into `nPoints` segments with `nSeries` points drawn on each radar line.

General format is intended for use in cases where you want to display multiple x-axis values on the same chart.

The following table shows four formatted data examples. An explanation of each element follows.

### 4.8.1 Formatted Data Examples

Array Data Format (Recommended)				
ARRAY 2 3				
HOLE 10000				
'Point 0' 'Point 1' 'Point 2'				
# x-values common to all points				
1.0 2.0 3.0				
# y-values				
'Series 0' 50.0 75.0 60.0				
'Series 1' 25.0 10.0 50.0				
# Series-label is optional				
Transposed Array Data Format (same data as previous)				
ARRAY 2 3 T				
HOLE 10000				
# 2 series of 3 points, Transposed				
'' 'Series 0' 'Series 1'				
# Optional Series-labels				
# x-values Y0-values Y1-values				
'Point 0' 1.0 50.0 25.0				
# Point-labels are optional				
'Point 1' 2.0 75.0 10.0				
'Point 2' 3.0 60.0 50.0				
General Data Format (Use if x-data is different for each series)				
GENERAL 2 4				
HOLE -10000				
# 2 series, max 4 points in each				
'Series 0' 2				
# Use only if custom hole value needed				
1.0 3.0				
# 2 points, optional series label				
50.0 60.0				
# x-values				
# y-values				
'Series 1' 4				
# 4 points				
2.0 2.5 3.5 5.0				
# x-values				
45.0 60.0 HOLE 70.0				
# y-values, including data hole				
Transposed General Data Format (same data as previous)				
GENERAL 2 4 T				
HOLE -10000				
# 2 series, max 4 points in each, Transposed				
'Series 0' 2				
# 2 points, optional series label				
# X Y				
1.0 50.0				
3.0 60.0				
'Series 1' 4				
# 4 points				
# X Y				
2.0 45.0				
2.5 60.0				
3.5 HOLE				
5.0 70.0				

### 4.8.2 Explanation of Format Elements

The first (non-comment) line must begin with either “ARRAY” or “GENERAL” followed by two integers specifying the number of series and the number of points in each series. For example:

```
# This is an Array data file containing 2 series of 4 points
ARRAY 2 4
```

The only difference with General data is that the second integer specifies the *maximum* number of points possible for each series:

```
# A General data file, 5 series, maximum 10 points
GENERAL 5 10
```

### Hole Value

The second line can *optionally* specify a data hole value. A hole value is a number that is interpreted by the chart as missing data. There should be only one hole value per `ChartDataView` class. Use a hole value if you know that a particular value in the data should be ignored in the chart:

```
HOLE 10000
```

You can also indicate that any particular point is a hole by specifying the word “HOLE” for that x- or y-value. For example:

```
50.0 75.0 HOLE 70.0
```

**Note:** If the hole value is later changed in the data view, values in the x- and y-data previously set with hole values will not change their values and will not draw.

By default, hole values are not drawn on charts. For a selection of chart types, you can change this default behavior by specifying a style to use for hole values. For more information, see [Hole Styles](#), in Chapter 7.

### Comments

You can use comments throughout the data file to make it easier for people to understand. Any text on a line following a “#” symbol is treated as a comment and is ignored.

### Point Labels

The third line can *optionally* specify text labels for each data point, which can be used to annotate the x-axis. Point-labels are generally only useful with Array data; if specified for General data they apply to the first series. The following shows how to specify Point-labels:

```
'Point 1' 'Point 2' 'Point 3' # Optional Point-labels
```

### The Data – Array layout

The rest of the file contains the data to be charted. Array layout uses the first line of data as x-values that are common to all points. Subsequent lines specify the y-values for each data series:

```
1.0 2.0 3.0 4.0           # x-values
150.0 175.0 160.0 170.0    # y-values, series 0
125.0 100.0 225.0 300.0    # y-values, series 1
# y-values continue, until end of data
```

## The Data – General layout

General layout provides more flexibility. For each series, the first line of data specifies the number of points in the series (this cannot be greater than the maximum number of points defined earlier). The second line specifies the x-values for that series; the third line specifies the y-values:

```
4                      # Series 0, 4 points
50.0 75.0 60.0 70.0   # x-values
25.0 10.0 25.0 30.0   # y-values
# Next series follows, until end of data
```

## Series Labels

You can *optionally* specify text labels for each series, which can be displayed in the legend. Series labels are enclosed in single quotes. In Array data, the label appears at the start of each line of y-values, for example:

```
'Series label' 150.0 175.0 160.0 170.0    # y-values, series 0
```

In General data, the label appears at the start of the line defining the number of points in that series, for example:

```
'Series label' 4      # Series 0, 4 points
50.0 75.0 60.0 70.0   # x-values
25.0 10.0 25.0 30.0   # y-values
```

## Transposed Data

JClass Chart can also interpret transposed data, where the meaning of the data series and points is switched. This may be a more convenient way to supply data to the chart for some applications. Note that transposing data also transposes series and point labels. To indicate that the data is transposed, add a “T” to the first line specifying the data layout and size. The following illustrates how data is interpreted when transposed:

```
ARRAY 2 3 T
# x-values  Y0-values  Y1-values
1.0         150.0      125.0
2.0         175.0      100.0
3.0         160.0      225.0
```

## 4.9 Making Your Own Chart Data Source

### 4.9.1 The Simplest Chart Data Source Possible

In order for a data source object to work with JClass Chart, it must implement the `ChartDataModel` interface. The `EditableChartDataModel` interface is an extension of `ChartDataModel` and can be used when you want to allow the data source to be editable. The `LabelledChartDataModel` and the `HoleValueChartDataModel` interfaces can be used in conjunction with `ChartDataModel` to extend the functionality of `ChartDataModel` to allow for label values (via the `LabelledChartDataModel` interface) and hole values (via the `HoleValueChartDataModel` interface).

The `ChartDataModel` interface is intended for use with existing data objects. It allows the chart to ask the data source for the number of data series, and the x-values and y-values for each data series. The interface looks like this:

```
public double[] getXSeries(int index);
public double[] getYSeries(int index);
public int getNumSeries();
```

Basically, `JClass Chart` organizes data based on data series. Each series has x-values and y-values, returned by `getXSeries()` and `getYSeries()`, respectively. It is expected that, for a given series index, the X series and Y series will be the same length.

If the x-data is the same for all y-data, then the same X series can be returned for each value. `JClass Chart` will automatically re-use the array.

As an example, consider `SimplestDataSource` in `examples.chart.datasource` example:

```
/**
 * This example shows the simplest possible chart data source.
 * The data source contains two data series, held in "xvalues"
 * and "yvalues" below.
 */
public class SimplestDataSource extends JPanel implements ChartDataModel {

    // x values for chart.
    protected double xvalues[] = { 1, 2, 3, 4 };
    // y values.
    protected double yvalues[][] = { {20, 10, 30, 25}, {30, 22, 10, 40}};

    /**
     * Retrieves the specified x-value series
     * In this example, the same x values are used regardless of
     * the index.
     * @param index data series index
     * @return array of double values representing x-value data
     */
    public double[] getXSeries(int index) {
        return xvalues;
    }

    /**
     * Retrieves the specified y-value series
     * In this example, yvalues contains the y data.
     * @param index data series index
     * @return array of double values representing x-value data
     */
    public double[] getYSeries(int index) {
        return yvalues[index];
    }

    /**
     * Retrieves the number of data series.
     * In this example, there are only two data
     * series.
     */
    public int getNumSeries() {
```

```
        return yvalues.length;
    }
}
```

There are two series in this example. The x-data is repeated for both series, and is stored in an array of doubles (xvalues). The y-data is stored in an array of arrays of doubles (yvalues). Each sub-array is the same length as x-values.

**Note:** You can run this example from the *JCLASS\_HOME/examples/chart/datasource/* directory.

## 4.9.2 LabelledChartDataModel – Labelling Your Chart

Sometimes, it is important to label each data series and each point in a graph. This information can be added to a data source using the `LabelledChartDataModel` interface.

The `LabelledChartDataModel` interface allows specification of series and point labels for your data. It is an optional part of the chart data model, but is very commonly used:

```
public int getNumSeries();
public String[] getPointLabels();
public String[] getSeriesLabels();
public String getDataSourceName();
```

The `getPointLabels()` call returns the point labels for all points in the chart. The size of the String array should correspond with the number of items in the `XSeries` and `YSeries` arrays.

The `getSeriesLabels()` call returns the series labels for the chart. The size of the String array should correspond to the value returned by `getNumSeries()`. Series labels appear in the legend.

The `getDataSourceName()` call returns the name of the data source. This appears in the chart as the title of the legend.

As an example, consider the following code (taken from `LabelledDataSource` in *JCLASS\_HOME/examples/chart/datasource/*):

```
/**
 * This example shows how to add point and series labelling
 * to a data source. It extends SimplestDataSource and
 * implements the LabelledChartDataModel interface to add
 * this information. The result can be seen on the x axis
 * (point labels representing quarters) and in the legend
 * (title, series names).
 */
public class LabelledDataSource extends SimplestDataSource implements
LabelledChartDataModel {

    // Point labels
    protected String pointLabels[] = { "Q1", "Q2", "Q3", "Q4" };

    // Series labels
```

```

protected String seriesLabels[] = { "West", "East" };

/*
 * Retrieves the labels to be used for each point in a
 * particular data series.
 * @return array of point labels
 */
public String[] getPointLabels() {
    return pointLabels;
}

/**
 * Retrieves the labels to be used for each data series
 */
public String[] getSeriesLabels() {
    return seriesLabels;
}

/**
 * Retrieves the name for the data source
 */
public String getDataSourceName() {
    return "Sales By Region";
}

```

As noted, this data source extends `SimplestDataSource`, adding in the required methods for returning point labels – `getPointLabels()` – and series labels – `getSeriesLabels()`.

Note that the number of items in the array returned by `getSeriesLabels()` should equal the number returned by `getNumSeries()`.

Also note that the number of items in the array returned by `getPointLabels()` should equal the number of items in the array returned by `getXSeries()` and `getYSeries()`. (In cases where the x-data is unique for each series and each series has a possibly different number of points, the point labels are applied to the first series.

**Note:** You can run this example from the `JCLASS_HOME/examples/chart/datasource/` directory.

### 4.9.3 EditableChartDataModel – Modifying Your Data

If you want to allow users to modify data using the edit trigger in JClass Chart, your data source must implement `EditableChartDataModel`. The `EditableChartDataModel` interface extends `ChartDataModel`, adding a single method that allows the chart to modify data in the data source:

```

public boolean setDataItem(int seriesIndex, int pointIndex,
    double newValue);

```

The `seriesIndex` and `pointIndex` values are used to save the data sent in `newValue`. Note that `EditableChartDataModel` only allows for Y-values to be changed. In other words, `newValue` is a Y-value!



As an example, consider `EditableDataSource` in *JCLASS\_HOME/examples/chart/datasource/*.

```
/**
 * This example shows how to make a data source editable
 * by adding the EditableChartDataModel interface to
 * the object.
 */
public class EditableDataSource extends LabelledDataSource implements
EditableChartDataModel {

/**
 * Change the specified y data value.
 * In this example, the series and point indices index
 * into the yvalues array originally defined in SimplestDataSource.
 *
 * @param seriesIndex series index for the point to be changed.
 * @param pointIndex point index for the point to be changed.
 * @param newValue new y value for the specified point
 * @return boolean value indicating whether the new value was
 * accepted. "true" means value was accepted.
 */
public boolean setDataItem(int seriesIndex, int pointIndex, double
                           newValue) {
    if (newValue < 0) return false;
    yvalues[seriesIndex][pointIndex] = newValue;
    return true;
}
```

In this example, the value is saved back into the `yvalues` array from `SimplestDataSource`, using the `seriesIndex` and `pointIndex` values to index to the appropriate array member.

This example extends `LabelledDataSource`, adding the `setDataItem()` method to allow chart to modify the data in the data source.

**Note:** You can run this example from *JCLASS\_HOME/examples/chart/datasource/*.

#### 4.9.4 HoleValueChartDataModel – Specifying Hole Values

If you want to supply a specific hole value along with your data, your data source must implement the [HoleValueChartDataModel interface](#).

As noted in Section 4.8.2, [Explanation of Format Elements](#), a hole value is a particular value in the data that should be ignored by the chart. There should be only one hole value per data source.

The `HoleValueChartDataModel` interface has one method, `getHoleValue()`. This method retrieves the hole value for the data source.

**Note:** The default hole value is `Double.MAX_VALUE`.

## 4.10 Making an Updating Chart Data Source

Quite often, the data shown in JClass Chart is dynamic. This kind of data requires creation of an updating data source. An updating data source is capable of informing chart that a portion of the data has been changed. Chart can then act on the change.

JClass Chart uses the standard AWT/Swing event/listener mechanism for passing changes between the chart data source and JClass Chart. At a very high level, JClass Chart is a listener to data source events that are fired by the data source.

### 4.10.1 Chart Data Source Support Classes

There are a number of data source related support classes included with JClass Chart. These classes make it easier to build updating data sources.

#### ChartDataEvent and ChartDataListener

The `ChartDataListener` interface is implemented by objects interested in receiving `ChartDataEvents`. Most often, the only `ChartDataListener` is JClass Chart itself. `ChartDataEvent` and `ChartDataListener` give data sources away to send update messages to Chart.

The `ChartDataListener` interface has only one method:

```
public void chartDataChange(ChartDataEvent e);
```

This method is used by the data source to inform the listener of a change. *In most systems, only JClass Chart need implement this interface.*

The `ChartDataEvent` object has three immutable properties: `Type`, `SeriesIndex`, and `PointIndex`. `SeriesIndex` and `PointIndex` are used to specify the data affected by the posted change. If all data is affected, the enum values `ALL_SERIES` and `ALL_POINTS` can be used.

`Type` is used to specify the kind of update:

Message	Meaning
<code>ADD_SERIES</code>	A new data series has been added to the end of the existing series in the data source.
<code>APPEND_DATA</code>	Used in conjunction with the <code>FastUpdate</code> feature, this tells the listener that data has been added to the existing series. Please see <a href="#">FastUpdate</a> , in Chapter 9, for full details.
<code>CHANGE_CHART_TYPE</code>	A request from the data source to change the chart type. The chart type is held inside <code>seriesIndex</code> .
<code>INSERT_SERIES</code>	A new data series has been added; <code>seriesIndex</code> indicates where the series should be added.

Message	Meaning
RELOAD	The data has completely changed; the difference here is that the dimensions of the data source (that is, number of data series and number of points) has not changed.
RELOAD_ALL_POINT_LABELS	Tells the listener to reload all point labels.
RELOAD_ALL_SERIES_LABELS	Tells the listener to reload all series labels.
RELOAD_DATA_SOURCE_NAME	Tells the listener the data source name has changed.
RELOAD_POINT	Single data value has changed, as specified by <code>seriesIndex</code> and <code>pointIndex</code> .
RELOAD_POINT_LABEL	Tells the listener to reload the point label specified by <code>pointIndex</code> .
RELOAD_SERIES	An entire data series has changed, as specified by <code>seriesIndex</code> ( <code>pointIndex</code> ignored).
RELOAD_SERIES_LABEL	Tells the listener to reload the series label specified by <code>seriesIndex</code> .
REMOVE_SERIES	Removes the series at <code>seriesIndex</code> .
RESET	The data source has completely changed.

### ChartDataManageable and ChartDataManager

This interface is used by a data source to tell Chart that it will be sending `ChartDataEvents` to Chart. Without this interface, there is no way for Chart to know that it has to attach itself as a `ChartDataListener` to the data source.

The only method in `ChartDataManageable` returns a `ChartDataManager`:

```
public abstract ChartDataManager getChartDataManager();
```

A `ChartDataManager` is an object that knows how to register and deregister `ChartDataListeners`. Chart uses this object to register itself as a listener to events from the data source.

The quickest way to get a data source set up is to extend or use `ChartDataSupport`.

### ChartDataSupport

`ChartDataSupport` provides a default implementation of `ChartDataManager`. It will manage a list of `ChartDataListeners`. It also provides two convenience methods for firing events to the listeners:

```
public void fireChartDataEvent(int type, int seriesIndex, int
    pointIndex)
public void fireChartDataEvent(ChartDataEvent evt)
```

The first method listed above is the most convenient. Given a `ChartDataEvent` Type, `SeriesIndex`, and `PointIndex`, it constructs and fires a `ChartDataEvent` to all listeners. The second method requires that you construct the `ChartDataEvent` yourself.

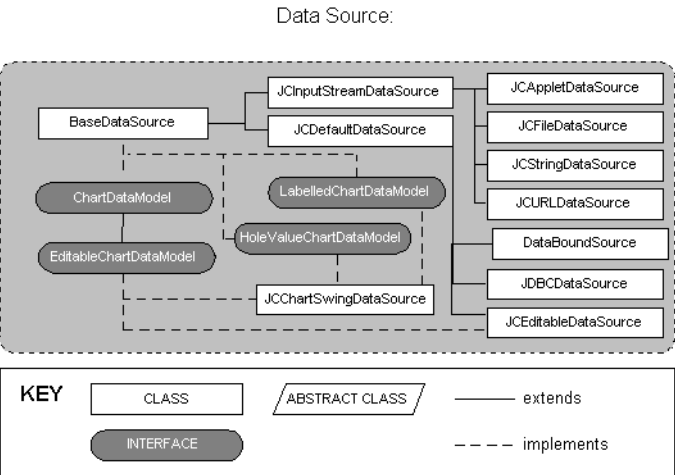
### Creating an Updating Data Source

If your datasource either extends or contains `ChartDataSupport`, sending updates from the data source to the chart is easy. Simply call `fireChartDataEvent()` with the event you wish to send.

```
fireChartDataEvent(ChartDataEvent.RESET, 0, 0);
```

To have JClass Chart automatically added as a listener, your data source needs to implement the `ChartDataManageable` interface and to return the `ChartDataSupport` instance in the `getChartDataManager()` method.

### JClass Chart Data Source Hierarchy



# Adding Data with the Targeted Data Model

*Overview of the Targeted Data Model* ■ *Review of JDBC Result Sets*  
*Adding Data from a Result Set to a Chart* ■ *Result Set Data Set Implementations by Chart Type*  
*Advanced Topics* ■ *Creating a Custom Data Set Implementation*

After you choose the chart type that you want to create, the next step is to add your data to the chart. The implementation of the targeted data model that ships with JClass Chart is designed for JDBC result sets.

**Important:** If you are developing with JavaBeans or XML, you need to use the underlying data model. And, even if your application meets the requirements of the targeted data model, you can still choose to use the underlying data model instead. For more information, see [Adding Data](#), in Chapter 1, and Chapter 4, [Adding Data with the Underlying Data Model](#).

**Note:** There are some properties that can only be set using the `ChartDataView` object in the underlying data model. These properties are noted throughout this guide.

The first two sections of this chapter provide an overview of the targeted data model and a review of JDBC result sets. The next section describes how to add data from a JDBC result set, starting with creating a data set implementation, followed by instantiating the `DefaultDataModel`, and finally setting the `DefaultDataModel` on the chart. The examples in this section make use of a basic pre-built data set implementation and use minimal options. Chart-type specific data set implementations are discussed in the next section.

If your data is not in a result set format, you can use the underlying data model, or you can create a custom data set implementation. The last section in this chapter outlines when and how to create a custom data set and describes the interfaces and classes that are available to you.

## 5.1 Overview of the Targeted Data Model

The targeted data model requires a *data model* object and at least one *data set* object. The *data model* object applies to the entire chart. It contains a list of the data sets that will be graphed on the chart as well as top-level image map information. The *data set* object contains the information to bind your data to elements in the chart.

Adding data to a chart can be broken down into three steps:

1. **Data Set:** Create an instance of a data set implementation and bind your data. See Section 5.3.1, [Creating a Data Set Implementation for a Result Set](#).
2. **Data Model:** Create an instance of `DefaultDataModel` and add your data set. See Section 5.3.2, [Creating an Instance of DefaultDataModel](#).
3. **Chart:** Set the data model on the chart. See Section 5.3.3, [Setting the Data Model on Your Chart](#).

The implementation of the targeted data model that ships with JClass Chart is designed for JDBC result sets. The next sections describe JDBC result sets and how to add data from a result set to your chart.

## 5.2 Review of JDBC Result Sets

A JDBC result set is in a table format of rows and columns. Each row represents a record in a relational database. Series of data can be organized by column or by row.

### Column-oriented Result Set

In a column-oriented result set, the values for each data series are represented by a single column. Columns may also represent other information that is part of the data set such as x-values, timestamps, or point labels. A row, therefore, represents the values of all series and related information at a single point.

In the following example, there are three data series: Product A, Product B, and Product C. Each row describes monthly data points for all the products.

Q1 Sales Report				
Month	Month Name	Product A	Product B	Product C
1	January	500	1000	2500
2	February	750	1500	2000
3	March	300	800	2200

Figure 26 Example of a column-oriented result set.

## Row-oriented Result Set

In a row-oriented result set, the values of all data series are combined into a single column. Other data set information (such as x-values, timestamps, or point labels) are similarly combined into their own columns. One column is designated to contain the series id or, in the case of a pie chart, the slice id. The data in any given row belongs to the series identified in the series id/slice id column. A row, therefore, represents the values of one series and its related information at a single point.

In the following example, there are two series: Product A and Product B. The Product column contains the series id values, that is, the names of the products. Each row describes a monthly data point for the stated product.

Q1 Sales Report			
Product	Month	Month Name	Sales
Product A	1	January	500
Product A	2	February	750
Product A	3	March	300
Product B	1	January	1000
Product B	2	February	1500
Product B	3	March	800

Figure 27 Example of a row-oriented result set.

## 5.3 Adding Data from a Result Set to a Chart

This section is organized into the following steps:

- [Creating a Data Set Implementation for a Result Set](#)
- [Creating an Instance of DefaultDataModel](#)
- [Setting the Data Model on Your Chart](#)

### 5.3.1 Creating a Data Set Implementation for a Result Set

The data set implementations for JDBC result sets are located in the package `com.klg.jclass.chart.model.impl`. If your data is in a result set format, you can instantiate one of these classes to create your data bindings. If your data is in another format, you may still want to work through this section and the associated class files to get an idea of how interfaces are implemented.

This section provides sample code to create an instance of `BasicResultSetDataSet` and to add data to a chart using the sample data from the preceding section, [Review of JDBC](#)

**Result Sets.** The examples use a `Vector` object to create lists, but you can use whichever `List` implementation you prefer. You can also add image maps from your result set or call other data-related methods that are available in the implementation classes. For more information, see Section 5.5, [Advanced Topics](#).

While these examples use `BasicResultSetDataSet`, the code for the other chart-type implementations is similar. You need to change two things. First, when creating an instance, substitute in the name of the data set implementation for the required chart type. Second, you need to review the parameters required by its `addResultSetBinding()` method and fill in the appropriate values. For many of the chart types, the sample parameter values will be no different than those found in these examples. For more information, see Section 5.4, [Result Set Data Set Implementations by Chart Type](#).

### 5.3.1.1 Creating an Instance of a Column-oriented Result Set

The following code snippet creates and uses `BasicResultSetDataSet` with a column-oriented result set. The three 'Product' columns are listed and then bound as the data series. The x-values are bound to the 'Month' column, and the labels for the x-values are bound to the 'Month Name' column. To see an image of a chart with the sample data, see Figure 28.

```
// Create an instance of the data set implementation.
BasicResultSetDataSet myResultSetDataSet = new BasicResultSetDataSet();

// Create a list containing the columns to bind to the data series.
Vector seriesColumnNames = new Vector();
seriesColumnNames.add("Product A");
seriesColumnNames.add("Product B");
seriesColumnNames.add("Product C");

// Bind column-oriented data to the chart in the following order:
// name of the resultset = myResultSet
// name of the column containing x-values = "Month"
// name of the column containing labels for the x-values = "Month Name"
// the List containing the data series bindings = seriesColumnNames
myResultSetDataSet.addResultSetBinding(myResultSet,
    "Month", "Month Name", seriesColumnNames, null);
```

The null value in the above code snippet means that the values for `seriesColumnNames` will be used for `seriesLabels` as well. You could specify your own labels.

### 5.3.1.2 Creating an Instance of a Row-oriented Result Set

The following code snippet creates and uses `BasicResultSetDataSet` with a row-oriented result set. The series id and the series label are bound to the 'Product' column. The series label format string is in the format specified by the `java.text.MessageFormat` class. The x-values are bound to the 'Month' column, and the labels for the x-values are bound to the 'Month Name' column. The y-values are bound to the 'Sales' column. To see an image of a chart with the sample data, see Figure 29.



```

// Create an instance of the data set implementation.
BasicResultSetDataSet myResultSetDataSet = new BasicResultSetDataSet();

// Create a list containing the name of the column with the series ids.
Vector seriesColumnNames = new Vector();
seriesColumnNames.add("Product");

// Create a String to label all series (uses java.text.MessageFormat style).
String seriesLabelFormatString = "Product Name: {0}";

// Create a list containing the name of the column to use for series labels.
Vector seriesLabelColumnNames = new Vector();
seriesLabelColumnNames.add("Product");

// Bind row-oriented data to the chart in the following order:
// name of the result set = myResultSet
// the List containing the name of the series id column = seriesColumnNames
// the String containing a label for all series = seriesLabelFormatString
// the List containing series labels column = seriesLabelColumnNames
// name of the column containing x-values = "Month"
// name of the column containing labels for the x values = "Month Name"
// name of the column containing y-values = "Sales"
myResultSetDataSet.addResultSetBinding(myResultSet, seriesColumnNames,
    seriesLabelFormatString, seriesLabelColumnNames, "Month",
    "Month Name", "Sales");

```

### 5.3.2 Creating an Instance of DefaultDataModel

After you create your data set implementation, you then add the data set to an instance of `DefaultDataModel`. The `DefaultDataModel` class is located in the package `com.klg.jclass.chart.model.impl`. It contains methods to add and return data sets, and to set and return top-level image map information. It is versatile enough so that it can be used as-is with most applications.

To use the `DefaultDataModel` class, you create an instance of it in your application. You then add data sets using one of the `addDataSet*()` methods.

**Note:** Data sets have to be instantiated and bound to your data before you can use them with a data model. The following code snippet assumes this has been done. For more information, see Section 5.3.1, [Creating a Data Set Implementation for a Result Set](#).

```

DefaultDataModel myDataModel = new DefaultDataModel();
// To add a single data set, use the addDataSet() method.
myDataModel.addDataSet(myResultSetDataSet);

// For multiple data sets, use the addDataSets() method.
Vector myDataSets = new Vector();
myDataSets.add(myResultSetDataSet);
myDataSets.add(anotherResultSetDataSet);

myDataModel.addDataSets(myDataSets);

```

### 5.3.3 Setting the Data Model on Your Chart

The final step in adding data to your chart is to set your instance of `DefaultDataModel` on the chart. You need to call one of the following methods:

- To replace any existing data on the chart, use the `setDataModel()` method.  
`JCChart.setDataModel(myDataModel);`
- To add the data to pre-existing chart data, use the `addDataModel()` method.  
`JCChart.addDataModel(myDataModel);`

The method queries the data model implementation for its list of data sets, style settings, and other data-related properties and sets this information on the chart. The data points returned by each data set are organized by the pertinent id (such as series, bar, slice) and converted into the data views and data series required for internal storage in chart. Data that is required to be in Array format (as per the chart type specified in the data set) is converted at this time. For more information on the internal workings of chart data storage, see Chapter 4, [Adding Data with the Underlying Data Model](#).

## 5.4 Result Set Data Set Implementations by Chart Type

This section describes all the pre-built `ResultSetDataSet` implementations provided with JClass Chart. These implementations use method parameter names and offer attributes that are meaningful in the context of a particular type of chart. To determine which implementation you should use, locate the type of chart that you want to create in the table below and go to the section listed beside it. If you are creating a custom chart, review the example for the chart type closest to the format of your custom chart.

Chart Type	Result Set Data Set Implementation
Area or Stacking Area	<i>See</i> <a href="#">Section 5.4.1, BasicResultSetDataSet</a>
Bar or Stacking Bar	<i>See</i> <a href="#">Section 5.4.2, BarResultSetDataSet</a>
Candle	<i>See</i> <a href="#">Section 5.4.3, FinancialResultSetDataSet</a>
HiLo or HiLoOpenClose	<i>See</i> <a href="#">Section 5.4.3, FinancialResultSetDataSet</a>
Pie	<i>See</i> <a href="#">Section 5.4.4, PieResultSetDataSet</a>
Plot or Scatter Plot	<i>See</i> <a href="#">Section 5.4.1, BasicResultSetDataSet</a>
Polar	<i>See</i> <a href="#">Section 5.4.5, PolarResultSetDataSet</a>
Radar or Area Radar	<i>See</i> <a href="#">Section 5.4.6, RadarResultSetDataSet</a>

### 5.4.1 BasicResultSetDataSet

The `BasicResultSetDataSet` data set implementation can be used to create most charts (financial-based charts excluded). However, the terminology used to name parameters and data point attributes is designed to be meaningful for area, stacking area, plot, and scatter plot charts. To bind data to these types of charts, you need to create an instance of `BasicResultSetDataSet.java`. For examples of how to do this, see Section 5.3.1, [Creating a Data Set Implementation for a Result Set](#).

#### 5.4.1.1 Column-oriented Result Set: BasicResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, xColumnName,
                                       xLabelColumnName, seriesColumnNames, seriesLabels);
```

The following table describes the method parameters.

Parameter	Description	Type
rs	The JDBC result set containing the source data.	ResultSet
xColumnName	The name of the column in the result set that contains the x-values for the data points. If null, the x-value is inferred from the row number.	String
xLabelColumnName	The name of the column that contains the label for the corresponding x-value. If null, no label is used.	String
seriesColumnNames	The names of the columns that contain the y-values that will be graphed at the corresponding x-value.	List of Strings
seriesLabels	Labels that identify each series in the legend. If null, series labels are taken from <code>seriesColumnNames</code> .	List of Strings

The following image shows how to bind the parameters to a sample result set.

Month	Month Name	Product A	Product B	Product C
1	January	500	1000	2500
2	February	750	1500	2000
3	March	300	800	2200

The following image shows the sample data graphed on a chart.

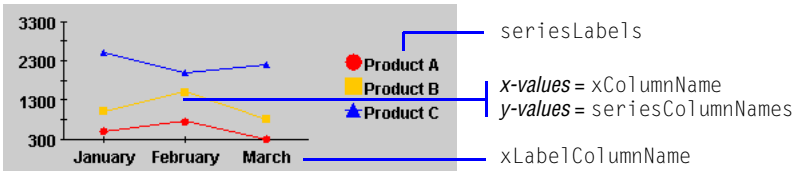


Figure 28 A plot chart displaying column-oriented sample data.

5.4.1.2 Row-oriented Result Set: BasicResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, seriesColumnNames,
    seriesLabelFormatString, seriesLabelColumnNames, xColumnName,
    xLabelColumnName, yColumnName);
```

The following table describes the method parameters.

Parameter	Description	Type
rs	The JDBC result set containing the source data.	ResultSet
seriesColumnNames	The names of the columns in the result set that, taken together, differentiate one data series from another.	List of Strings
seriesLabelFormatString	Text (in the format specified by the <code>java.text.MessageFormat</code> class) used to build the labels that identify the series in the legend. Values from the <code>seriesLabelColumnNames</code> parameter complete the String. For example, using sample data with <code>seriesLabelColumnNames</code> set to the 'Product' column, String <code>seriesLabelFormatString = "Product Name: {0}"</code> ; would display "Product Name: Product A" and "Product Name: Product B" on the chart. If null, default series labels are used.	String
seriesLabelColumnNames	The name of the columns whose values will be substituted in the <code>seriesLabelFormatString</code> parameter. If null, series labels are created from the <code>seriesLabelFormatString</code> parameter.	List of Strings
xColumnName	The name of the column containing the x-values for the data points. If null, the x-value is inferred from the row number within series id.	String
xLabelColumnName	The name of the column containing the label for the corresponding x-value. If null, no label is used.	String
yColumnName	The name of the column containing the y-value to be graphed at the corresponding x-value.	String

The following image shows how to bind the parameters to a sample result set.

seriesLabelColumnNames		xLabelColumnName		yColumnName
seriesColumnNames	xColumnName			
Product	Month	Month Name	Sales	
Product A	1	January	500	
Product A	2	February	750	
Product A	3	March	300	
Product B	1	January	1000	
Product B	2	February	1500	
Product B	3	March	800	

The following image shows the sample data graphed on a chart.

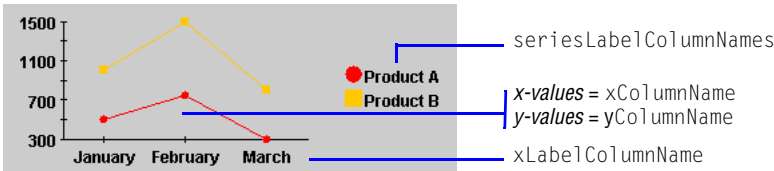


Figure 29 A plot chart displaying row-oriented sample data.

### 5.4.2 BarResultSetDataSet

To bind data in a result set to a bar or stacking bar chart, you need to create an instance of `BarResultSetDataSet`. For basic examples of how to do this, see Section 5.3.1, [Creating a Data Set Implementation for a Result Set](#).

### 5.4.2.1 Column-oriented Result Set: BarResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, clusterIdColumnName,  
                                     clusterLabelColumnName, seriesColumnNames, seriesLabels);
```

The following table describes the method parameters.

Parameter	Description	Type
<code>rs</code>	The JDBC result set containing the source data.	ResultSet
<code>clusterIdColumnName</code>	The name of the column in the result set that contains the bar cluster id values. The cluster id assigns the data point to a particular bar cluster. If null, the cluster id is inferred from the row number.	String
<code>clusterLabelColumnName</code>	The name of the column that contains the label for the cluster defined by the corresponding cluster id. If null, no label is used.	String
<code>seriesColumnNames</code>	The names of the columns that contain the y-values that will be graphed at the corresponding cluster id value.	List of Strings
<code>seriesLabels</code>	Labels that identify each series in the legend. If null, series labels are taken from <code>seriesColumnNames</code> .	List of Strings

The following image shows how to bind the parameters to a sample result set.

clusterIdColumnName		seriesColumnNames		
clusterLabelColumnName		seriesLabels		
Month	Month Name	Product A	Product B	Product C
1	January	500	1000	2500
2	February	750	1500	2000
3	March	300	800	2200

The following image shows the sample data graphed on a chart.

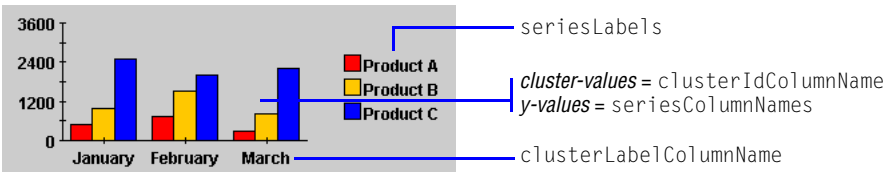


Figure 30 A bar chart displaying column-oriented sample data.

### 5.4.2.2 Row-oriented Result Set: BarResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, seriesColumnNames,  
    seriesLabelFormatString, seriesLabelColumnNames,  
    clusterIdColumnName, clusterLabelColumnName, yColumnName);
```

The following table describes the method parameters.

Parameter	Description	Type
<code>rs</code>	The JDBC result set containing the source data.	ResultSet
<code>seriesColumnNames</code>	The names of the columns in the result set that, taken together, differentiate one data series from another.	List of Strings
<code>seriesLabelFormatString</code>	Text (in the format specified by the <code>java.text.MessageFormat</code> class) used to build the labels that identify the series in the legend. Values from the <code>seriesLabelColumnNames</code> parameter complete the String. For example, using sample data with <code>seriesLabelColumnNames</code> set to the 'Product' column, String <code>seriesLabelFormatString</code> = "Product Name: {0}"; would display "Product Name: Product A" and "Product Name: Product B" on the chart. If null, default series labels are used.	String
<code>seriesLabelColumnNames</code>	The name of the columns whose values will be substituted in the <code>seriesLabelFormatString</code> parameter. If null, series labels are created from the <code>seriesLabelFormatString</code> parameter.	List of Strings
<code>clusterIdColumnName</code>	The name of the column containing the cluster id values. The cluster id assigns the data point to a particular bar cluster. If null, the cluster id is inferred from the row number within series id.	String
<code>clusterLabelColumnName</code>	The name of the column containing the label for the cluster defined by the corresponding cluster id. If null, no label is used.	String
<code>yColumnName</code>	The name of the column containing the y-value that is part of the bar cluster defined by the corresponding cluster id.	String

The following image shows how to bind the parameters to a sample result set.

seriesLabelColumnNames		clusterLabelColumnName	
seriesColumnNames	clusterIdColumnName	yColumnName	
Product	Month	Month Name	Sales
Product A	1	January	500
Product A	2	February	750
Product A	3	March	300
Product B	1	January	1000
Product B	2	February	1500
Product B	3	March	800

The following image shows the sample data graphed on a chart.

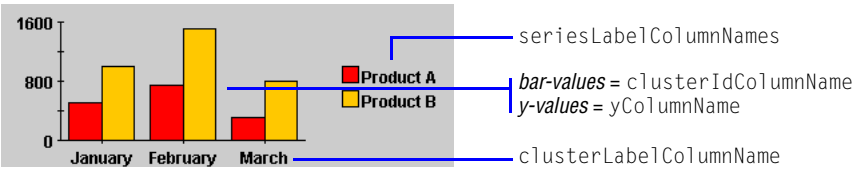


Figure 31 A bar chart displaying row-oriented sample data.

### 5.4.3 FinancialResultSetDataSet

To bind data in a result set to a candle, Hi-Lo, or Hi-Lo-Open-Close chart, you need to create an instance of `FinancialResultSetDataSet`. For basic examples of how to do this, see Section 5.3.1, [Creating a Data Set Implementation for a Result Set](#).

#### 5.4.3.1 Column-oriented Result Set: FinancialResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, xColumnName, xLabelColumnName,  
    highColumnNames, lowColumnNames, openColumnNames, closeColumnNames,  
    seriesLabels);
```

The following table describes the method parameters.

Parameter	Description	Type
rs	The JDBC result set containing the source data.	ResultSet
xColumnName	The name of the column in the result set that contains the x-values for the data points. If null, the x-value is inferred from the row number.	String



Parameter	Description	Type
xLabelColumnName	The name of the column that contains the label for the corresponding x-value. If null, no label is used.	String
highColumnNames	The names of the columns that contain the high financial values that will be graphed at the corresponding x-value.	List of Strings
lowColumnNames	The names of the columns that contain the low financial values that will be graphed at the corresponding x-value.	List of Strings
openColumnNames	The names of the columns that contain the open financial values that will be graphed at the corresponding x-value.	List of Strings
closeColumnNames	The names of the columns that contain the close financial values that will be graphed at the corresponding x-value.	List of Strings
seriesLabels	Labels that identify each series in the legend. If null, series labels are taken from the high, low, open, and close ColumnNames parameters.	List of Strings

The following image shows how to bind the parameters to a sample result set.

xColumnName	highColumnName	openColumnName	highColumnName	openColumnName				
xLabelColumnName	lowColumnName	closeColumnName	lowColumnName	closeColumnName				
Date	Stock A High	Stock A Low	Stock A Open	Stock A Close	Stock B High	Stock B Low	Stock B Open	Stock B Close
Jan 1, 2004	12.5	11.4	11.6	12.1	20.3	15.6	19.5	16.7
Jan 2, 2004	13.8	11.9	12.1	13.2	20.8	16.2	16.7	19.5
Jan 3, 2004	13.4	12.3	13.2	12.5	24.0	19.2	19.5	23.6

The following image shows the sample data graphed on a chart.

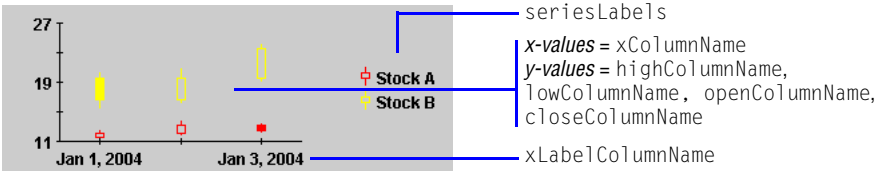


Figure 32 A financial chart displaying column-oriented sample data.

### 5.4.3.2 Row-oriented Result Set: FinancialResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, seriesColumnNames,  
    seriesLabelFormatString, seriesLabelColumnNames,  
    xColumnName, xLabelColumnName,  
    highColumnName, lowColumnName, openColumnName, closeColumnName);
```

The following table describes the method parameters.

Parameter	Description	Type
<code>rs</code>	The JDBC result set containing the source data.	ResultSet
<code>seriesColumnNames</code>	The names of the columns in the result set that, taken together, differentiate one data series from another.	List of Strings
<code>seriesLabelFormatString</code>	Text (in the format specified by the <code>java.text.MessageFormat</code> class) used to build the labels that identify the series in the legend. Values from the <code>seriesLabelColumnNames</code> parameter complete the String. For example, using sample data with <code>seriesLabelColumnNames</code> set to 'Stock Name' column, <code>String seriesLabelFormatString = "Stock: {0}";</code> would display "Stock: Stock A" and "Stock: Stock B" on the chart. If null, default series labels are used.	String
<code>seriesLabelColumnNames</code>	The name of the columns whose values will be substituted in the <code>seriesLabelFormatString</code> parameter. If null, series labels are created from the <code>seriesLabelFormatString</code> parameter.	List of Strings
<code>xColumnName</code>	The name of the column containing the x-values for the data points. If null, the x-value is inferred from the row number within series id.	String
<code>xLabelColumnName</code>	The name of the column containing the label for the corresponding x-value. If null, no label is used.	String
<code>highColumnName</code>	The name of column containing the high financial value to be graphed at the corresponding x-value.	String
<code>lowColumnName</code>	The name of the column containing the low financial value to be graphed at the corresponding x-value.	String
<code>openColumnName</code>	The name of the column containing the open financial value to be graphed at the corresponding x-value.	String
<code>closeColumnName</code>	The name of the column containing the close financial value to be graphed at the corresponding x-value.	String

The following image shows how to bind the parameters to a sample result set.

**Note:** High, low, open, and close values must be in separate columns in the result set.

xLabelColumnName	seriesLabelColumnNames	highColumnName	openColumnName
xColumnName	seriesColumnNameNames	lowColumnName	closeColumnName

Date	Stock Name	Stock High	Stock Low	Stock Open	Stock Close
Jan 1, 2004	Stock A	12.5	11.4	11.6	12.1
Jan 1, 2004	Stock B	20.3	15.6	19.5	16.7
Jan 2, 2004	Stock A	13.8	11.9	12.1	13.2
Jan 2, 2004	Stock B	20.8	16.2	16.7	19.5
Jan 3, 2004	Stock A	13.4	12.3	13.2	12.5
Jan 3, 2004	Stock B	24.0	19.2	19.5	23.6

The following image shows the sample data graphed on a chart.

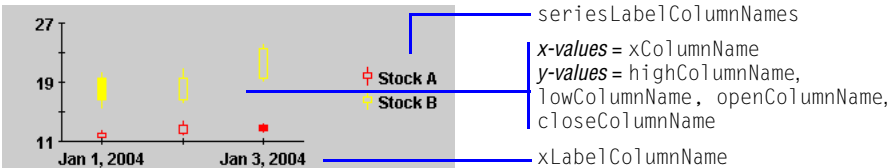


Figure 33 A financial chart displaying row-oriented sample data.

### 5.4.4 PieResultSetDataSet

To bind data in a result set to a pie chart, you need to create an instance of `PieResultSetDataSet`. For basic examples of how to do this, see Section 5.3.1, [Creating a Data Set Implementation for a Result Set](#).

### 5.4.4.1 Column-oriented Result Set: PieResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, pieIdColumnName,
                                       pieLabelColumnName, sliceColumnNames, sliceLabels);
```

The following table describes the method parameters.

Parameter	Description	Type
rs	The JDBC result set containing the source data.	ResultSet
pieIdColumnName	The name of the column in the result set that contains the pie id values. The pie id assigns the data point to a particular pie. If null, the pie id is inferred from the row number.	String
pieLabelColumnName	The name of the column that contains the label for the pie defined by the corresponding pie id. If null, no label is used.	String
sliceColumnNames	The names of the columns that contain the y-values that are turned into slices of the pie, where the pie is defined by the corresponding pie id.	List of Strings
sliceLabels	Labels that identify each pie slice in the legend. If null, the slice labels are taken from <code>sliceColumnNames</code> .	List of Strings

The following image shows how to bind the parameters to a sample result set.

Month	Month Name	Product A	Product B	Product C
1	January	500	1000	2500
2	February	750	1500	2000
3	March	300	800	2200

The following image shows the sample data graphed on a chart.

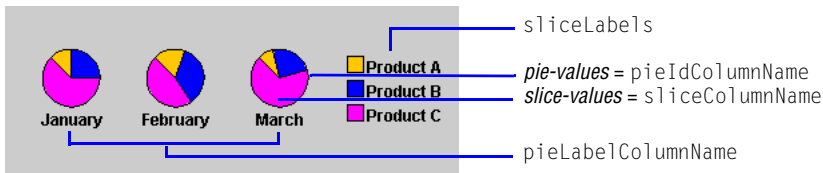


Figure 34 Pie charts displaying column-oriented sample data.

#### 5.4.4.2 Row-oriented Result Set: PieResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, sliceColumnNames,
    sliceLabelFormatString, sliceLabelColumnNames,
    pieIdColumnName, pieLabelColumnName, yColumnName);
```

The following table describes the method parameters.

Parameter	Description	Type
rs	The JDBC result set containing the source data.	ResultSet
sliceColumnNames	The names of the columns in the result set that, taken together, differentiate one data series (that is, a pie slice that exists in one or more pies) from another.	List of Strings
sliceLabelFormatString	Text (in the format specified by the <code>java.text.MessageFormat</code> class) used to build the labels that identify the slice in the legend. Values from the <code>sliceLabelColumnNames</code> parameter complete the String. For example, using sample data with <code>sliceLabelColumnNames</code> set to the 'Product' column, <code>String sliceLabelFormatString = "Product Name: {0}";</code> would display "Product Name: Product A" and "Product Name: Product B" on the chart. If null, default slice labels are used.	String
sliceLabelColumnNames	The name of the columns whose values will be substituted in the <code>sliceLabelFormatString</code> parameter. If null, series labels are created from the <code>sliceLabelFormatString</code> parameter.	List of Strings
pieIdColumnName	The name of the column containing the pie id values. The pie id assigns the data point to a particular pie. If null, the pie id is inferred from the row number in slice.	String
pieLabelColumnName	The name of the column containing the label for the pie defined by the corresponding pie id. If null, no label is used.	String
yColumnName	The name of the column containing the y-value of the slice defined by the corresponding slice column in the pie, where the pie is defined by the corresponding pie id.	String

The following image shows how to bind the parameters to a sample result set.

sliceLabelColumnNames		pieLabelColumnName	
sliceColumnNames	pieIdColumnName		yColumnName
Product	Month	Month Name	Sales
Product A	1	January	500
Product A	2	February	750
Product A	3	March	300
Product B	1	January	1000
Product B	2	February	1500
Product B	3	March	800

The following image shows the sample data graphed on a chart.

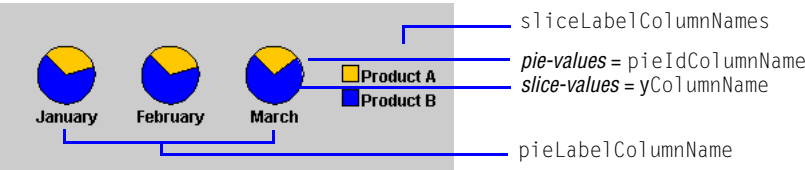


Figure 35 Pie charts displaying row-oriented sample data.

### 5.4.5 PolarResultSetDataSet

To bind data in a result set to a polar chart, you need to create an instance of `PolarResultSetDataSet`. For basic examples of how to do this, see Section 5.3.1, [Creating a Data Set Implementation for a Result Set](#).

#### 5.4.5.1 Column-oriented Result Set: PolarResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, angleColumnName,  
                                       angleLabelColumnName, seriesColumnNames, seriesLabels);
```

The following table describes the method parameters.

Parameter	Description	Type
rs	The JDBC result set containing the source data.	ResultSet
angleColumnName	The name of the column in the result set that contains the angle values for the data points. If null, the angle value is inferred from the row number.	String
angleLabelColumnName	The name of the column that contains the label for the corresponding angle value. If null, no label is used.	String
seriesColumnNames	The names of the columns that contain the y values that will be graphed at the corresponding angle value.	List of Strings
seriesLabels	Labels that identify each series in the legend. If null, series labels are taken from <code>seriesColumnNames</code> .	List of Strings

The following image shows how to bind the parameters to a sample result set.

angleColumnName		seriesColumnNames		
	angleLabelColumnName		seriesLabels	
Angle	Angle Name	Series A	Series B	Series C
0	East	4.0	5.0	6.0
90	North	4.0	5.0	6.0
180	West	4.0	5.0	6.0
270	South	4.0	5.0	6.0
360	East	4.0	5.0	6.0

The following image shows the sample data graphed on a chart.

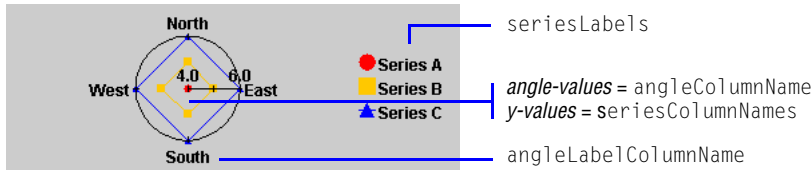


Figure 36 A polar chart displaying column-oriented sample data.

### 5.4.5.2 Row-oriented Result Set: **PolarResultSetDataSet**

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, seriesColumnNames,  
    seriesLabelFormatString, seriesLabelColumnNames,  
    angleColumnName, angleLabelColumnName, yColumnName);
```

The following table describes the method parameters.

Parameter	Description	Type
<code>rs</code>	The JDBC result set containing the source data.	ResultSet
<code>seriesColumnNames</code>	The names of the columns in the result set that, taken together, differentiate one data series from another.	List of Strings
<code>seriesLabelFormatString</code>	Text (in the format specified by the <code>java.text.MessageFormat</code> class) used to build the labels that identify the series in the legend. Values from the <code>seriesLabelColumnNames</code> parameter complete the String. For example, using sample data with <code>seriesLabelColumnNames</code> set to the <code>Product</code> column, String <code>seriesLabelFormatString = "Product Name: {0}"</code> ; would display "Product Name: Product A" and "Product Name: Product B" on the chart. If null, default series labels are used.	String
<code>seriesLabelColumnNames</code>	The name of the columns whose values will be substituted in the <code>seriesLabelFormatString</code> parameter. If null, series labels are created from the <code>seriesLabelFormatString</code> parameter.	List of Strings
<code>angleColumnName</code>	The name of the column containing the angle values for the data points. If null, the angle value is inferred from the row number within series id.	String
<code>angleLabelColumnName</code>	The name of the column containing the label for the corresponding angle value. If null, no label is used.	String
<code>yColumnName</code>	The name of the column containing the y-value to be graphed at the corresponding angle.	String



The following image shows how to bind the parameters to a sample result set.

seriesLabelColumnNames		angleLabelColumnName	
seriesColumnNames	angleColumnName		yColumnName
Series	Angle	Angle Name	Values
Series A	0	East	4.0
Series A	90	North	4.0
Series A	180	West	4.0
Series A	270	South	4.0
Series A	360	East	4.0
Series B	0	East	5.0
Series B	90	North	5.0
Series B	180	West	5.0
Series B	270	South	5.0
Series B	360	East	5.0

The following image shows the sample data graphed on a chart.

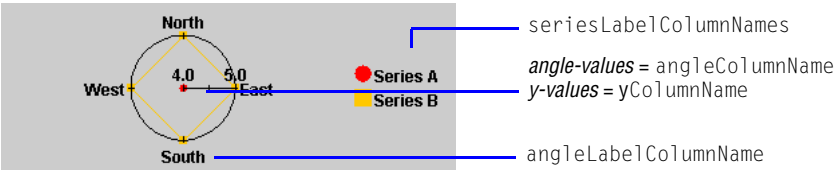


Figure 37 A polar chart displaying row-oriented sample data.

### 5.4.6 RadarResultSetDataSet

To bind data in a result set to a radar or area radar chart, you need to create an instance of `RadarResultSetDataSet`. For basic examples of how to do this, see Section 5.3.1, [Creating a Data Set Implementation for a Result Set](#).

#### 5.4.6.1 Column-oriented Result Set: `RadarResultSetDataSet`

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, spokeIdColumnName,  
    spokeLabelColumnName, seriesColumnNames, seriesLabels);
```

The following table describes the method parameters.

Parameter	Description	Type
rs	The JDBC result set containing the source data.	ResultSet
spokeIdColumnName	The name of the column in the result set that contains the spoke id values. The spoke id assigns the data point to a radar spoke. If null, the spoke id is inferred from the row number.	String
spokeLabelColumnName	The name of the column that contains the label for the radar spoke defined by the corresponding spoke id. If null, no label is used.	String
seriesColumnNames	The names of the columns that contain the y values that will be graphed at the corresponding spoke id.	List of Strings
seriesLabels	Labels that identify each series in the legend. If null, series labels are taken from seriesColumnNames.	List of Strings

The following image shows how to bind the parameters to a sample result set.

spokeIdColumnName		seriesColumnNames		
spokeLabelColumnName		seriesLabels		
Month	Month Name	Product A	Product B	Product C
1	January	500	1000	2500
2	February	750	1500	2000
3	March	300	800	2200

The following image shows the sample data graphed on a chart.

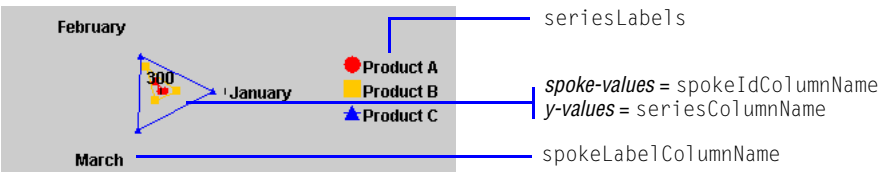


Figure 38 A radar chart displaying column-oriented sample data.

### 5.4.6.2 Row-oriented Result Set: RadarResultSetDataSet

You need to call the `addResultSetBinding()` method with the following parameters:

```
myResultSetDataSet.addResultSetBinding(rs, seriesColumnNames,  
    seriesLabelFormatString, seriesLabelColumnNames,  
    spokeIdColumnName, spokeLabelColumnName, yColumnName);
```

The following table describes the method parameters.

Parameter	Description	Type
rs	The JDBC result set containing the source data.	ResultSet
seriesColumnNames	The names of the columns in the result set that, taken together, differentiate one data series from another.	List of Strings
seriesLabelFormatString	Text (in the format specified by the <code>java.text.MessageFormat</code> class) used to build the labels that identify the series in the legend. Values from the <code>seriesLabelColumnNames</code> parameter complete the String. For example, using sample data with <code>seriesLabelColumnNames</code> set to the Product column, String <code>seriesLabelFormatString = "Product Name: {0}"</code> ; would display "Product Name: Product A" and "Product Name: Product B" on the chart. If null, default series labels are used.	String
seriesLabelColumnNames	The name of the columns whose values will be substituted in the <code>seriesLabelFormatString</code> parameter. If null, series labels are created from the <code>seriesLabelFormatString</code> parameter.	List of Strings
spokeIdColumnName	The name of the column containing the spoke id values. The spoke id assigns the data point to a radar spoke. If null, the spoke id is inferred from the row number within series id.	String
spokeLabelColumnName	The name of column containing the label for the radar spoke defined by the corresponding spoke id. If null, no label is used.	String
yColumnName	The name of column containing the y-value that will be graphed at the radar spoke defined by the corresponding spoke id.	String

The following image shows how to bind the parameters to a sample result set.

seriesLabelColumnNames		spokeLabelColumnName	
seriesColumnName	spokeIdColumnName		yColumnName
Product	Month	Month Name	Sales
Product A	1	January	500
Product A	2	February	750
Product A	3	March	300
Product B	1	January	1000
Product B	2	February	1500
Product B	3	March	800

The following image shows the sample data graphed on a chart.

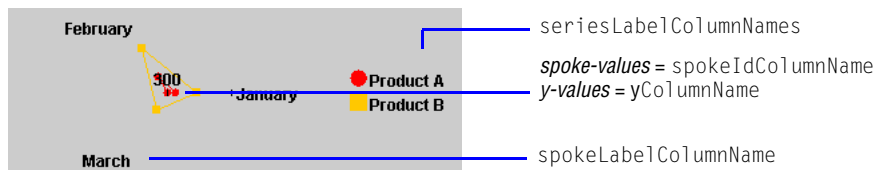


Figure 39 A radar chart displaying row-oriented sample data.

## 5.5 Advanced Topics

The result set data set implementations offer additional attributes that are related to the data, but not part of it. For example, you can set the name of an axis. The following sections summarize the available properties. Some properties are available in more than one data set implementation; these are listed under [Shared Properties](#). Other properties are only offered by a single data set implementation as outlined in Section 5.5.0.2, [Chart-specific Properties](#). The methods to set these properties are described in the *API Documentation*.

### 5.5.0.1 Shared Properties

The following table summarizes the properties that are shared among some or all of the result set data set implementations (as noted). You can use any of the properties indicated for the type of chart that you are creating. In addition, there are some chart-specific properties. For details, see Section 5.5.0.2, [Chart-specific Properties](#).

Property	Description	Basic	Bar	Financial	Pie	Polar	Radar
ChartType	Chart type (e.g. PLOT, BAR, etc.). Values are the same as those taken by the <code>ChartDataView.setChartType()</code> method.	X	X	X	X	X	X
FillStylePalette	List of <code>JCFillStyle</code> objects to be used as the palette from which fill styles are selected when drawing the chart. Fill styles are used to draw bar, pie, area, and area radar charts.	X	X	--	X	--	X
HoleValue	When this value is encountered, it is assumed to represent missing data.	X	X	X	X	X	X
LineStylePalette	List of <code>JCLineStyle</code> objects to be used as the palette from which line styles are selected when drawing the chart. Line styles are used to draw Hi-Lo, Hi-Lo-Open-Close, plot, polar, and radar charts.	X	--	X	--	X	X
OutlineLineStyle	<code>JCLineStyle</code> object representing line style used to outline fill areas in the chart. The outline Line style is used to draw bar, pie, area, and area radar charts.	X	X	--	X	--	X
SeriesFillStyle	Associates a <code>JCFillStyle</code> object with a series id. When the series is drawn, this fill style will be used. Fill styles are used to draw bar, pie, area, and area radar charts.	X	X	--	X	--	X
SeriesLabel	Associates a series label with a series id.	X	X	X	--	X	X
SeriesLineStyle	Associates a <code>JCLineStyle</code> object with a series id. When the series is drawn, this line style will be used. Line styles are used to draw Hi-Lo, Hi-Lo-Open-Close, plot, polar, and radar charts.	X	--	X	--	X	X
SeriesSymbolStyle	Associates a <code>JCSymbolStyle</code> object with a series id. When the series is drawn, this symbol style will be used. Symbol styles are used to draw plot, polar, and radar charts.	X	--	--	--	X	X
SymbolStylePalette	List of <code>JCLineStyle</code> objects to be used as the palette from which line styles are selected when drawing the chart. Symbol styles are used to draw plot, polar, and radar charts.	X	--	--	--	X	X
XAxisName	Identifies the x-axis in the chart against which data will be plotted.	X	X	X	--	--	--
XLabels	List of x-labels applied across all points in a every series.	X	--	X	--	--	--
XMarkers	List of <code>DataMarker</code> objects representing markers along the x-axis.	X	--	X	--	--	--
XNumericalTimeBase	Date object serving as time base when x numerical time data is turned on.	X	--	X	--	--	--
XNumericalTimeData	Turns interpretation of x-values as numerical time data on/off.	X	--	X	--	--	--

Property	Description	Basic	Bar	Financial	Pie	Polar	Radar
XNumericalTimeUnit	Units of numerical time data when x numerical time data is turned on. Values are the same as those set in <code>JCAxis.setTimeUnit()</code> .	X	--	X	--	--	--
XThresholds	List of <code>DataThreshold</code> objects representing thresholds along the x-axis.	X	--	X	--	--	--
XValues	List of x-values applied across all points in a every series.	X	--	X	--	--	--
YAxisName	Identifies the y-axis in the chart against which data will be plotted.	X	X	X	--	X	X
YMarkers	List of <code>DataMarker</code> objects representing markers along the y-axis.	X	X	X	--	--	--
YNumericalTimeBase	Date object serving as time base when y numerical time data is turned on.	X	X	X	--	--	--
YNumericalTimeData	Turns interpretation of y-values as numerical time data on/off.	X	X	X	--	--	--
YNumericalTimeUnit	Units of numerical time data when y numerical time data is turned on. Values are the same as those set in <code>JCAxis.setTimeUnit()</code> .	X	X	X	--	--	--
YThresholds	List of <code>DataThreshold</code> objects representing thresholds along the y-axis.	X	X	X	--	--	--

### 5.5.0.2 Chart-specific Properties

The following data set implementations also contain properties unique to the chart types:

- [BarResultSetDataSet](#)
- [FinancialResultSetDataSet](#)
- [PieResultSetDataSet](#)
- [PolarResultSetDataSet](#)
- [RadarResultSetDataSet](#)

#### BarResultSetDataSet

Bar Property	Description
ClusterIds	List of bar cluster ids applied across all points in a every series.
ClusterLabels	List of bar cluster labels applied across all points in a every series.
ClusterMarkers	List of <code>DataMarker</code> objects representing markers at cluster ids along the x-axis.

## FinancialResultSetDataSet

Financial Property	Description
CandleOutlineStylePalette	List of <code>JCChartStyle</code> objects to be used as the palette from which financial candle outline styles are selected when drawing the chart.
FallingCandleStylePalette	List of <code>JCChartStyle</code> objects to be used as the palette from which financial falling candle styles are selected when drawing the chart.
HiloStylePalette	List of <code>JCChartStyle</code> objects to be used as the palette from which financial Hi-Lo styles are selected when drawing the chart.
RisingCandleStylePalette	List of <code>JCChartStyle</code> objects to be used as the palette from which financial rising candle styles are selected when drawing the chart.
SeriesCandleOutlineStyle	Associates a <code>JCChartStyle</code> object containing a financial candle outline style with a series id. When the series is drawn, this style will be used.
SeriesFallingCandleStyle	Associates a <code>JCChartStyle</code> object containing a financial falling candle style with a series id. When the series is drawn, this style will be used.
SeriesHiloStyle	Associates a <code>JCChartStyle</code> object containing a financial Hi-Lo style with a series id. When the series is drawn, this style will be used.
SeriesRisingCandleStyle	Associates a <code>JCChartStyle</code> object containing a financial rising candle style with a series id. When the series is drawn, this style will be used.
TickSize	Associates a tick size to be used in a financial Hi-Lo-Open-Close chart with one or all series.

## PieResultSetDataSet

Pie Property	Description
OtherSliceExploded	Marks the other slice of a particular pie or of all pies to be drawn exploded.
OtherSliceFillStyle	<code>JCFillStyle</code> objects to be used as the fill style for the other slice on a pie chart.
OtherSliceLabel	The label that identifies the other slice in the legend.
PieIds	List of pie ids applied across all points in a every series.
PieLabels	List of pie labels applied across all points in a every series.
SliceExploded	Marks a particular slice in a particular pie or a particular slice across all pies to be drawn exploded.
SliceLabel	Associates a slice label with a slice id.

## PolarResultSetDataSet

Polar Property	Description
AngleLabels	List of angle labels applied across all points in a every series.
AngleUnit	Unit of angle in x-values.
AngleValues	List of angle values applied across all points in a every series.
ThetaAxisName	Identifies the theta axis in the chart against which data will be plotted.

## RadarResultSetDataSet

Radar Property	Description
SpokeIds	List of radar spoke ids applied across all points in a every series.
SpokeLabels	List of radar spoke labels applied across all points in a every series.

## 5.6 Creating a Custom Data Set Implementation

The JDBC result set data set implementations are based on a set of interfaces and classes. If your data is not in a JDBC result set, you can use these interfaces and classes to create a data set implementation that works with the format of your existing data source. The data source can be a flat file, a database-related format, or some other format.

This section provides an overview of the data model, followed by a `BasicDataSet` implementation example. After the example is a guide to help you select interfaces and classes for your own data set implementation. The sample code at the end of the section extends the `BasicDataSet` implementation example and demonstrates using the `BarDataSet` interface with its related style interface and data point class.

### 5.6.1 Understanding the Targeted Data Model

The targeted data model is based on a set of interfaces and classes. This section provides a high-level look at the interfaces and classes and how they work together. For details, see the [JClass API Documentation](#).

#### 5.6.1.1 Data Model Interface

The highest level interface is `DataModel`. It represents all data being added to the chart. Data is returned from the `getDataSets()` method as a `List` of data set implementations. An implementation of the `DataModel` interface, called `DefaultDataModel.java`, is provided and can be used in most applications. For more information, see Section 5.3.2, [Creating an Instance of DefaultDataModel](#).



### 5.6.1.2 Data Set Interfaces and Classes

The data set interfaces and classes are divided into five categories: chart-type data sets, data-type data sets, style data sets, iterators, and data points. Many of the interfaces and classes are optimized for specific chart types. This enables you to bind data to the chart using terminology and properties suitable for that chart type. For example, the interface for a pie chart has pie slice properties while the interface for a radar chart has spoke properties.

The following table lists the interfaces and classes by category, describes the purpose for each category, and notes which interfaces and classes are required.

Categories	Interfaces	Description
Chart-type Data Sets <sup>a</sup>	BasicDataSet BarDataSet PieDataSet PolarDataSet RadarDataSet	Interfaces for a chart type or set of chart types for one collection of data. For example, <code>PieDataSet</code> is, obviously, for pie charts, while <code>BasicDataSet</code> can be used for most charts, but is best for plot, scatter plot, area, stacking area, and financial charts.
Data-type Data Sets	ClusterDataSet NumericalTimeDataSet SeriesDataSet	Interfaces that either interpret the data provided by the <code>DataIterator</code> or allow parts of the data to be specified separately.
Style Data Sets	AreaStyleDataSet AreaRadarStyleDataSet BarStyleDataSet CandleStyleDataSet HiloStyleDataSet HLOCStyleDataSet PieStyleDataSet PlotStyleDataSet PolarStyleDataSet RadarStyleDataSet	Interfaces for chart style elements that are tightly coupled with the data for one collection of data.
Iterators	DataIterator <sup>b</sup> MarkerIterator ThresholdIterator	Interfaces that iterate over your data for the purpose of returning data to JClass Chart. <code>DataIterator</code> returns all data points in the data set. <code>MarkerIterator</code> and <code>ThresholdIterator</code> do likewise with markers and thresholds in the data set.
Data Points <sup>a</sup>	BasicDataPoint BarDataPoint FinancialDataPoint PieDataPoint PolarDataPoint RadarDataPoint	Classes that describe the properties of a point of data for a particular chart type or set of chart types. Each instance of a class represents a single point of data in the data set. These classes are returned from the <code>getNextDataPoint()</code> method in the <code>DataIterator</code> class.

a. Required. You need to implement the interface/class that matches your chart type.

b. Required.

The data set for a chart is made by implementing the required interfaces and classes, plus any other interfaces that suit the chart type or data source. For example, if you want to create a data set implementation for an area chart where the data contains time data, you can implement the following interfaces and classes:

- `BasicDataSet`
- `NumericalTimeDataSet`
- `AreaStyleDataSet` (optional)
- `DataIterator`
- `BasicDataPoint`

For more information, see Section 5.6.3, [Summary of the Interfaces and Classes](#).

### 5.6.2 Creating a `BasicDataSet` Implementation

The primary purpose of the data set implementation is to retrieve your data from wherever it is stored, assign the data to fields in a suitable `DataPoint` class, and return the `DataPoint` class from your implementation of `DataIterator`. The following example creates a data set implementation for a plot chart with three small series of three points each graphed against the default x and y axes. It is a simple data set implementation that takes values stored in local arrays and maps them to the fields in the `BasicDataPoint` class. There are no image maps, no markers, and no thresholds used in this example.

In your own implementation, you can expand on the methods used here or implement other interfaces, such as relevant data-type `DataSet` interfaces or a `StyleDataSet` interface.

```
import com.klg.jclass.chart.model.*;
import com.klg.jclass.chart.JCChart;
import com.klg.jclass.util.ImageMapInfo;

public class MyDataSet implements BasicDataSet, DataIterator {

    protected double[][] yvalues = {{5.0, 6.0, 7.0},
                                     {10.0, 4.5, 2.7},
                                     {3.8, 8.6, 4.3}};

    protected String[] seriesIds = {"Series 1", "Series 2", "Series 3"};
    protected String[] xLabels = {"Jan", "Feb", "Mar"};

    protected String dataSetName = null;
    protected int dataCounter = 0;
    protected int seriesCounter = 0;
    private BasicDataPoint basicDataPoint;

    public MyDataSet() {
        dataSetName = "My Data Set";
        basicDataPoint = new BasicDataPoint();
    }

    public void incrementCounters() {
        dataCounter++;
    }
```

```

        if (dataCounter >= yvalues[seriesCounter].length) {
            seriesCounter++;
            dataCounter = 0;
        }
    }

    /**
     * DataIterator implementation
     */
    public boolean hasMoreDataPoints() {
        if (seriesCounter >= yvalues.length) {
            return false;
        }
        return true;
    }

    public DataPoint getNextDataPoint() {
        basicDataPoint.clear();
        basicDataPoint.xValue = new Integer(dataCounter);
        basicDataPoint.yValue = new Double(yvalues[seriesCounter]
                                           [dataCounter]);
        basicDataPoint.seriesId = seriesIds[seriesCounter];
        basicDataPoint.seriesLabel = seriesIds[seriesCounter];
        basicDataPoint.xLabel = xLabels[dataCounter];
        incrementCounters();
        return basicDataPoint;
    }

    /**
     * BasicDataSet implementation
     */
    public String getName() {
        return dataSetName;
    }

    public int getChartType() {
        return JCChart.PLOT;
    }

    public DataIterator getDataIterator() {
        return this;
    }

    public Number getHoleValue() {
        return new Double(Double.MAX_VALUE);
    }

    public DataOrder getDataOrder() {
        return DataOrder.ASCENDING;
    }

    public ImageMapInfo getLegendImageMap() {
        return null;
    }

    public Class getXDataType() {

```

```

        try {
            return Class.forName("java.lang.Number");
        }
        catch (ClassNotFoundException cnfe) {}
        return null;
    }

    public Class getYDataType() {
        try {
            return Class.forName("java.lang.Number");
        }
        catch (ClassNotFoundException cnfe) {
            return null;
        }
    }

    public String getXAxisName() {
        return null;
    }

    public String getYAxisName() {
        return null;
    }

    public MarkerIterator getXMarkerIterator() {
        return null;
    }

    public MarkerIterator getYMarkerIterator() {
        return null;
    }

    public ThresholdIterator getXThresholdIterator() {
        return null;
    }

    public ThresholdIterator getYThresholdIterator() {
        return null;
    }
}

```

### 5.6.3 Summary of the Interfaces and Classes

The following tables briefly describe the purpose of the interfaces and classes. After you select the interfaces and classes that you think you might use, consult the [JClass API Documentation](#) for details.

### 5.6.3.1 Chart-type DataSet Interfaces

When creating a custom data set implementation, you should start by selecting a DataSet interface that suits the type of chart that you want to provide.

Chart-type DataSet	Description
BasicDataSet	Provides properties and methods relevant to most chart types, however the terminology used is best suited for area, stacking area, plot, scatter plot, and financial (Hi-Lo, Hi-Lo-Open-Close, candle) charts.
BarDataSet	Provides properties and methods for bar and stacking bar charts.
PieDataSet	Provides properties and methods for pie charts.
PolarDataSet	Provides properties and methods for polar charts.
RadarDataSet	Provides properties and methods for radar and area radar charts.

### 5.6.3.2 Data-type DataSet Interfaces

The data-type DataSet interfaces are required to support some types of data, in particular time data and data associated with a cluster or series of data. If your data source does not include these types of data, you do not need to implement a data-type DataSet interface.

Data-type DataSet	Description
ClusterDataSet	Provides information associated with a cluster of data (rather than a single point). For example, you can iterate over a list of x values applied to points at the same position in all series; these x values override the x values in the DataPoint classes.
NumericalTimeDataSet	Required when numerical data returned from the DataIterator should be interpreted as time data relative to a user-provided time base and unit.
SeriesDataSet	Provides information associated with a data series (rather than a point).

### 5.6.3.3 StyleDataSet Interfaces

StyleDataSet interfaces provide control over chart style elements that are tightly coupled with the data for one collection of data. If you are satisfied with the default styles, you do not need to implement a StyleDataSet interface. If you implement a StyleDataSet interface, select one that works with the chart-type DataSet that you are implementing.

StyleDataSet	Description	Chart-type DataSet
AreaStyleDataSet	Provides fill style and outline style information for data in area and stacking area charts.	BasicDataSet
AreaRadarStyleDataSet	Provides fill style and outline style information for data in area radar charts.	RadarDataSet <i>or</i> BasicDataSet

StyleDataSet	Description	Chart-type DataSet
BarStyleDataSet	Provides fill style and outline style information for data in bar and stacking bar charts.	BarDataSet <i>or</i> BasicDataSet
CandleStyleDataSet	Provides data-associated style information for data in candle charts, such as styles to use for the box portion of a candle chart.	BasicDataSet
HiloStyleDataSet	Provides line style information for data in Hi-Lo charts.	BasicDataSet
HLOCStyleDataSet	Provides line style and tick size information for data in HLOC charts.	BasicDataSet
PieStyleDataSet	Provides fill style and outline style information for data in pie charts.	PieDataSet <i>or</i> BasicDataSet
PlotStyleDataSet	Provides line style and symbol style information for data in plot and scatter plot charts.	BasicDataSet
PolarStyleDataSet	Provides line style and symbol style information for data in polar charts.	PolarDataSet
RadarStyleDataSet	Provides line style and symbol style information for data in radar charts.	RadarDataSet <i>or</i> BasicDataSet

### 5.6.3.4 Iterator Interfaces

The `DataIterator` interface is required so that JClass Chart can iterate over the data points in the data source. If you intend to use markers or thresholds in your chart (not all chart types support them), you also need to implement the corresponding interface.

Iterator	Description
<code>DataIterator</code>	Iterates over the data points until the end of the data is reached. Implement this interface in the class that iterates over your data points.
<code>MarkerIterator</code>	If you are adding markers to your chart, you need to implement this interface in the class that iterates over the markers. <b>ote:</b> Not applicable to pie data set implementations because this chart type does not support markers.
<code>ThresholdIterator</code>	If you are adding thresholds to your chart, you need to implement this interface in the class that iterates over the thresholds. <b>Note:</b> Not applicable to pie data set implementations because this chart type does not support thresholds.

### 5.6.3.5 DataPoint Classes

`DataPoint` classes define the information necessary to describe a single data point of a specific chart type. You use the class to map the elements in your data source to a neutral

format that can be understood by JClass Chart. Each instance of a class represents a single point of data in the data set.

DataPoint	Description	Chart-type DataSet
BasicDataPoint	Represents a chart data point for use with all chart types, however, it is best suited for area, stacking area, plot, and scatter plot chart types.	BasicDataSet
BarDataPoint	Represents a chart data point for bar and stacking bar charts.	BarDataSet <i>or</i> BasicDataSet
FinancialDataPoint	Represents a chart data point for Hi-Lo, Hi-Lo-Open-Close, and candle charts.	BasicDataSet
PieDataPoint	Represents a chart data point for pie charts	PieDataSet <i>or</i> BasicDataSet
PolarDataPoint	Represents a chart data point for polar charts.	PolarDataSet
RadarDataPoint	Represents a chart data point for radar and area radar charts.	RadarDataSet <i>or</i> BasicDataSet

#### 5.6.4 A BarDataSet Implementation Example

The following example demonstrates the use of one of the specialized chart-type DataSet interfaces: BarDataSet. The sample code creates a MyBarDataSet implementation that extends the MyDataSet implementation covered in Section 5.6.2, [Creating a BasicDataSet Implementation](#). It implements the BarDataSet and BarStyleDataSet interfaces and uses the BarDataPoint class. BarStyleDataSet is used to define the fill style and outline style to be used in the chart. BarDataPoint has properties that are especially suited to binding data to a bar chart.

```
import com.klg.jclass.chart.*;
import com.klg.jclass.chart.model.*;

import java.awt.Color;

public class MyBarDataSet extends MyDataSet
    implements BarDataSet, BarStyleDataSet {

    private Color[] colors = {Color.red, Color.white, Color.blue};
    private BarDataPoint barDataPoint;

    public MyBarDataSet() {
        dataSetName = "My Bar Data Set";
        barDataPoint = new BarDataPoint();
    }

    public DataPoint getNextDataPoint() {
        barDataPoint.clear();
        barDataPoint.clusterId = xLabels[dataCounter];
        barDataPoint.clusterLabel = xLabels[dataCounter];
    }
}
```

```

        barDataPoint.yValue = new Double(yvalues[seriesCounter][dataCounter]);
        barDataPoint.seriesId = seriesIds[seriesCounter];
        barDataPoint.seriesLabel = seriesIds[seriesCounter];
        incrementCounters();
        return barDataPoint;
    }

    public int getChartType() {
        return JCChart.STACKING_BAR;
    }

    /**
     * BarDataSet implementation. Rest of methods inherited from super class.
     */
    public MarkerIterator getClusterMarkerIterator() {
        return(null);
    }

    /**
     * BarStyleDataSet implementation
     */
    public JCFillStyle getFillStyle(Object seriesId) {
        JCFillStyle fillStyle = null;
        for (int i = 0; i < seriesIds.length; i++) {
            if (seriesIds[i].equals(seriesId)) {
                fillStyle = new JCFillStyle(colors[i], JCFillStyle.SOLID);
                break;
            }
        }
        return fillStyle;
    }

    public JCLineStyle getOutlineStyle() {
        return new JCLineStyle(1, Color.MAGENTA, JCLineStyle.SOLID);
    }
}

```



# Defining Axis Controls

*Axis Labelling and Annotation Methods* ■ *Positioning Axes* ■ *Chart Orientation and Axis Direction*  
*Setting Axis Bounds* ■ *Customizing Origins* ■ *Logarithmic Axes*  
*Titling Axes and Rotating Axis Elements* ■ *Gridlines* ■ *Adding a Second Y-Axis*

JClass Chart can automatically set properties based on the data, so axis numbering and data display usually do not need much customizing. However, you can control any aspect of the chart axes, depending on your requirements. This chapter covers the different axis controls available.

**Note:** If you are developing your chart application using one of the JClass Chart Beans, go to Chapter 13, [JClass Chart Beans](#) instead.

## 6.1 Axis Labelling and Annotation Methods

There are several ways to annotate the chart's axes, each suited to specific situations. The chart can automatically generate numeric annotation appropriate to the data it is displaying; you can provide a label for each point in the chart (x-axis only); you can provide a label for specific values along the axis; or the chart can automatically generate time-based annotations.

Please note that none of the axis properties discussed in this section apply to pie charts, because pie charts do not have axes. To annotate a pie chart, use chart labels; for more information, please see [Chart Labels](#), in Chapter 7.

Whichever annotation method you choose, the chart makes considerable effort to produce the most natural annotation possible, even as the data changes. You can fine-tune this process using axis annotation properties.

User-set annotations support the use of HTML tags. The use of HTML tags overrides the default `Font` and `Color` properties of the label.

Please note that HTML labels may not work with PDF, PS, or PCL encoding.

### 6.1.1 Choosing an Annotation Method

A variety of properties combine to determine the annotation that appears on the axes. The `JCAxis.AnnotationMethod` property specifies the method used to annotate the axis. The valid annotation methods are:

<code>JCAxis.VALUE</code> (default)	The chart chooses appropriate axis annotation automatically (with possible callbacks to a label generator), based on the chart type and the data itself.
<code>JCAxis.POINT_LABELS</code> (x-axis only)	The chart spaces the points based on the x-values and annotates them with text you specify (in the data source) for each point.
<code>JCAxis.VALUE_LABELS</code>	The chart annotates the axis with text you define for specific x-axis or y-axis coordinates.
<code>JCAxis.TIME_LABELS</code>	The chart interprets the x- or y-values as units of time, automatically choosing time/date annotation based on the starting point and format you specify. Not for polar, radar, or area radar charts.

#### Notes:

- Point labels annotation (`JCAxis.POINT_LABELS`) is only valid for an x-axis when it has been added to the x-axis collection in `JCChartArea`. This means that a new `JCAxis` instance that has not yet been added to `JCChartArea` will not be considered an x-axis.
- The spokes of area radar and radar charts are automatically labelled “0”, “1”, “2”, and so forth, unless the x-annotation method is `JCAxis.POINT_LABELS`.
- For polar charts, the default annotation for `JCAxis.VALUE` depends on the angle units specified. If it is radians, the symbol for pi will not be used (it will be represented by 3.14 instead). Also, the x-axis will always be linear; that is, setting the logarithmic properties to `true` will be ignored.

The following topics discuss setting up and fine-tuning each type of annotation.

### 6.1.2 Values Annotation

Values annotation produces numeric labelling along an axis, based on the data itself. The chart can produce very natural-looking axis numbering automatically, but you can fine-tune the properties that control this process.

#### Axis Annotation Increments, Numbering, and Precision

When a `JCAxis` is instantiated, a pair of `JCAnno` objects representing default labels and ticks are automatically created and set on the axis. Those default `JCAnno` objects may be modified, deleted, or augmented with other `JCAnno` objects.

The following table describes the different properties that can be set on a `JCAnno` object in order to customize the labels and tick marks:

Property	Function
<code>startValue</code>	Sets the value at which the annotation begins.
<code>stopValue</code>	Sets the value where the annotation ends.
<code>incrementValue</code>	Sets the increment between annotation along an axis.
<code>innerExtent</code>	Defines the space, in pixels, that tick marks extend into the plot area.
<code>outerExtent</code>	Defines the space, in pixels, that tick marks extend out of the plot area.
<code>tickColor</code>	Determines the color of the tick marks.
<code>drawTicks</code>	Determines whether or not the tick marks defined by <code>JCAnno</code> are drawn.
<code>labelExtent</code>	Defines the distance, in pixels, of the labels from the axis.
<code>labelColor</code>	Determines the color of the labels.
<code>precision</code>	Sets the number of decimal places to use when displaying a chart label number. The effect depends on whether it is positive or negative: * <i>Positive</i> values add that number of places after the decimal place. For example, a value of 2 displays an annotation of 10 as “10.00”. * <i>Negative</i> values indicate the minimum number of zeros to use before the decimal place. For example, a value of -2 displays annotation in multiples of 100.
<code>drawLabels</code>	Determines whether or not the labels defined by <code>JCAnno</code> are drawn.

When the annotation method for an axis is `VALUE_LABELS`, `POINT_LABELS`, or `TIME_LABELS`, the labels are either user-supplied or internally generated without the use of `JCAnno` objects. The boolean `UseAnnoTicks` property of a `JCAxis` determines how tick marks are drawn in those cases. If `UseAnnoTicks` is `true`, tick marks are drawn at the labels. If the value is `false`, ticks defined by `JCAnno` objects are drawn instead.

Using multiple `JCAnno` objects, an axis can be drawn with major and minor ticks. Labels can be turned on or off for the individual tick series, as can the actual tick marks, enabling further flexibility.

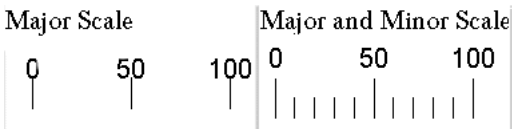


Figure 40 Different tick styles that can be applied to a chart axis.

Please refer to the `AnnoGrid.java` example included in the `examples/chart/intro` package to view different tick marks in a `JClass Chart` example.

### 6.1.3 PointLabels Annotation

`PointLabels` annotation displays defined labels along an x-axis. This is useful for annotating the x-axis of any chart for which all series share common x-values. `PointLabels` are most useful with bar, stacking bar, and pie charts. It is possible to add, remove, and edit `PointLabels`. In JClass Chart, `PointLabels` are typically defined with the data.

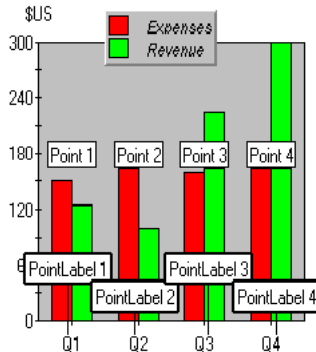


Figure 41 `PointLabels` x-axis annotation.

`PointLabels` are a collection of labels. The first label applies to the first point, the second label applies to the second point, and so on. The labels can also be supplied by setting the `PointLabels` property of the `ChartDataView` object for this chart. For example, the following code specifies labels for each of the three points on the x-axis:

```
String[] labels = {"Q1", "Q2", "Q3", "Q4"};
c.getChartArea().getXAxis(0).setAnnotationMethod(JCAxis.POINT_LABELS);
ChartDataView cd = c.getDataView(0);
ArrayList pLabels = new ArrayList();
for (int i = 0; i < labels.length; i++) {
    pLabels.add(labels[i]);
}
cd.setPointLabels(pLabels);
```

For polar, radar, and area radar charts, if the x-axis annotation is `POINT_LABELS` and the data is of type array, then a point label is drawn at the outside of the x-axis for each point. (Series labels are used in the legend as usual.)

**Note:** If you are using the targeted data model, you can set x-axis labels in your data set implementation.

### 6.1.4 ValueLabels Annotation

`ValueLabels` annotation displays labels at the axis coordinate specified. This is useful for displaying special text at a specific axis coordinate, or when a type of annotation that the chart does not support is needed, such as scientific notation. You can set the axis

coordinate and the text to display for each `ValueLabel`, and also add and remove individual `ValueLabels`.

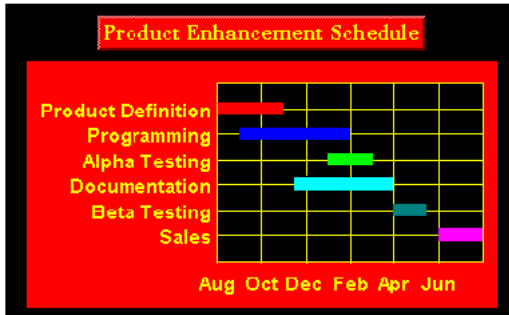


Figure 42 Using `ValueLabels` to annotate axes.

Every label displayed on the axis is one `ValueLabel`. Each `ValueLabel` has a `Value` property and a `Label` property.

If the `AnnotationMethod` property is set to `JCAxis.VALUE_LABELS`, the chart places labels at explicit locations along an axis. The `ValueLabels` property of `JCAxis`, which is a `ValueLabels` collection, supplies this list of Strings and their locations. For example, the following code sets value labels at the locations 10, 20, and 30:

```
String[] labels = {"Sales", "Beta Testing", "Documentation",
                  "Alpha Testing", "Programming",
                  "Production Definition"};
JCAxis y = c.getChartArea().getYAxis(0);
y.setAnnotationMethod(JCAxis.VALUE_LABELS);
JCValueLabel[] valueLabels = new JCValueLabel[labels.length];
for (int i = 0; i < labels.length; i++) {
    valueLabels[i] = new JCValueLabel(10.0 * (i + 1), labels[i], y);
}
y.setValueLabels(valueLabels);
```

The `ValueLabels` collection can be indexed either by subscript or by value:

```
JCAxis x = c.getChartArea().getXAxis(0);
// The following retrieves the second value label
JCValueLabel v1 = x.getValueLabels(1);
// The following retrieves the closest label to chart coordinate 2.0
JCValueLabel v2 = x.getValueLabel(2.0);
```

### 6.1.5 TimeLabels Annotation

`TimeLabels` annotation interprets the value data as units of time. The chart calculates and displays a time-axis based on the starting point and format specified. A time-axis is useful for charts that measure something in seconds, minutes, hours, days, weeks, months, or years.

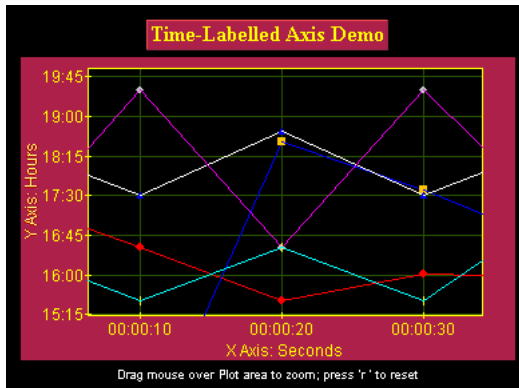


Figure 43 TimeLabels annotating x-axes and y-axes.

Four properties are used to control the display and behavior of TimeLabels:

- AnnotationMethod (set to JCAxis.TIME\_LABELS to use this annotation method)
- TimeUnit
- TimeBase
- TimeFormat

### Time Unit

Use the TimeUnit property to specify how to interpret the values in the data. Select either JCAxis.SECONDS, JCAxis.MINUTES, JCAxis.HOURS, JCAxis.WEEKS, JCAxis.MONTHS, or JCAxis.YEARS. For example, when set to JCAxis.YEARS, values that range from 5 to 15 become a time-axis spanning 10 years. By default, TimeUnit is set to JCAxis.SECONDS.

### Time Base

Use the TimeBase property to set the date and time that the time-axis starts from. Use the Java Date class (java.util.Date) to specify the TimeBase. The default for TimeBase is the current time.

For example, the following statement sets the starting point to January 15, 1985:

```
c.getChartArea().getXAxis(0).setTimeBase(new Date(85,0,15));
```

### Time Format

Use the TimeFormat property to specify the text to display at each annotation point. The TimeFormatIsDefault property allows the chart to automatically determine an appropriate format based on the TimeUnit property and the data, so it is often unnecessary to customize the format.

TimeFormat specifies a *time format*. You build a time format using the Java time format codes from the java.text.SimpleDateFormat class. The chart displays only the parts of

the date/time specified by `TimeFormat`. The format codes are based on the default Java formatting provided by `java.text`.

Symbol	Meaning	Presentation	Example
G	era designator		AD
y	year	Number	1997
M	month in year	Text & Number	July 07
d	day in month	Number	10
h	hour in am/pm (1 ~ 12)	Number	12
H	hour in day (0~23)	Number	0
m	minute in hour	Number	30
s	second in minute	Number	55
S	millisecond	Number	978
E	day in week	Text	Tuesday
D	day in year	Number	189
F	day of week in month	Number	2nd Wed in July
w	week in year	Number	27
W	week in month	Number	2
a	am/pm marker	Text	PM
k	hour in day (1~24)	Number	24
K	hour in am/pm (0~11)	Number	0
z	time zone	Text	Pacific Standard Time
'	escape for text	delimiter	
”	single quote	Literal	

The default for `TimeFormat` is the same as the default used by Java’s `SimpleDateFormat` class (located in the `java.text` package).

### Using Date Methods

The `dateToValue()` method converts a Java date value into its corresponding axis value (a floating-point value). The `valueToDate()` method converts a value along an axis to the date that it represents. Note that the axis must already be set as a time label axis.

Here is a code example showing the `dateToValue()` method converting a date (in this case, February 2, 1999) to a y-axis value, and showing the `valueToDate()` method converting a y-axis value (in this case, 3.0) to the date that it represents.

```
JCAxis y = chart.getChartArea().getYAxis(0);
Date d = y.valueToDate(3.0);
double val = y.dateToValue(new Date(99,1,2));
```

### 6.1.6 Custom Axes Labels

JClass Chart will label axes by default. However, you can also generate custom labels for the axes by implementing the `JCLabelGenerator` interface. This interface has one method – `makeLabel()` – that is called when a label is required at a particular value.

Note that the spokes of radar and area radar charts will be automatically labelled “0”, “1”, “2”, and so forth, unless the x-annotation method is `JCAxis.POINT_LABELS`.

To generate custom axes labels, the axis’ `AnnotationMethod` property, which determines how the axis is labelled, must be set to `VALUE`. Also, the `setLabelGenerator()` method must be called with the class that implements the `JCLabelGenerator` interface.

The number of labels, that is, the number of times `makeLabel()` is called, depends on the `NumSpacing` parameter of the axis. Not all labels will be displayed if there is not enough room.

The `makeLabel()` method takes two parameters: `value` (the axis value to be labelled) and `precision` (the numeric precision to be used).

- In the usual case, the `makeLabel()` method returns a `String`, and that `String` will be used as the axis label at `value`.
- If the `makeLabel()` method returns a `ChartText` object, then that `ChartText` object will be used as the axis label at `value`.
- If an object other than `String` or `ChartText` is returned, the `String` derived from calling that object’s `toString()` method will be used as the axis label at `value`.

Here is a code example showing how to customize the labels for a linear axis by implementing the `JCLabelGenerator` interface. In this case, Roman numeral labels are going to be generated (instead of the usual Arabic labels) for the numbers 1 through 13.

```
class MyLabelGenerator implements JCLabelGenerator
{
    public Object makeLabel(double value, int precision) {
        int intvalue = (int) value;
        String s = null;
        switch (intvalue) {
            case 1 :
                s = "I";
                break;
            case 2 :
                s = "II";
                break;
```



```

        case 3 :
            s = "III";
            break;
        case 4 :
            s = "IV";
            break;
        case 5 :
            s = "V";
            break;
        case 6 :
            s = "VI";
            break;
        case 7 :
            s = "VII";
            break;
        case 8 :
            s = "VIII";
            break;
        case 9 :
            s = "IX";
            break;
        case 10 :
            s = "X";
            break;
        default :
            s = "";
            break;
    }
    return s;
}
}

```

Note that the user will need to specify the label generator as follows:

```
axis.setLabelGenerator(new MyLabelGenerator());
```

Also note that JClass Chart calls the `makeLabel()` method for each *needed* label (recall that each axis requests needed labels based on its `NumSpacing`, `Min`, and `Max` properties). Thus, if JClass Chart needs  $n$  labels, the `makeLabel()` method is called  $n$  times.

## 6.2 Positioning Axes

Use the `Placement` property to make a specific axis placement or use the `PlacementIsDefault` property to specify whether the chart is meant to determine axis placement. When making a specific axis placement, the axes may be placed against its partner axis at that axis' minimum value, maximum value, origin value, or a user-specified value.

For example,

```
axis.setPlacement(JCAxis.MIN);
```

will place the axis against its partner axis' minimum value, while

```
axis.setPlacement(otherAxis, 5.0)
```

will place the axis against otherAxis at the value 5.0

**Note:** When Placement is set to Origin, changing the axis origin will move the placed axis to the new origin value.

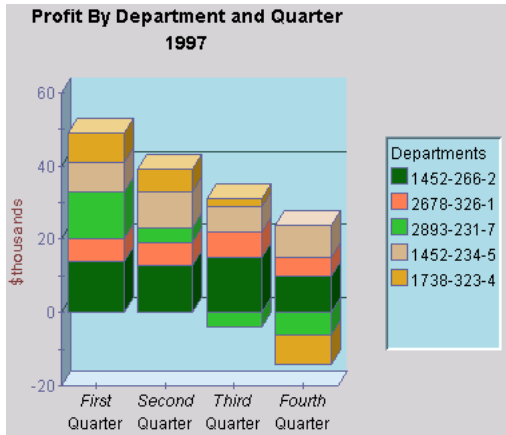


Figure 44 An example of axes positioning; the x-axis is placed against the y-axis' minimum value.

### Polar Charts – Special Minimum and Maximum Values

Note that for polar charts, the x-axis maximum and minimum values are fixed, and these fixed values change depending on the angle unit type. The y-axis maximum and minimum values are adjustable, but are constrained to avoid data clipping. The y-axis minimum will never be less than zero (unless the y-axis is reversed).  $(\theta, -r)$  will be interpreted as  $(\theta+180, r)$ . The y-axis minimum will always be at the center unless the axis is reversed, in which case the y-axis maximum will be at the center.

### Radar and Area Radar Charts – Minimum Values

The minimum value for a y-axis in radar and area radar charts can be negative.

## 6.3 Chart Orientation and Axis Direction

A typical rectangular chart draws the x-axis horizontally from left-to-right and the y-axes vertically from bottom-to-top. You can reverse the orientation of the entire chart, and/or the direction of each axis.

### 6.3.1 Inverting Chart Orientation

Use the `ChartDataView` object's `Inverted` property to change the chart orientation for rectangular charts. When set to `true`, the x-axis is drawn vertically and the y-axis horizontally for the data view. Any properties set on the x-axis then apply to the vertical axis, and y-axis properties apply to the horizontal axis.

**Note:** To switch the orientation of charts with *multiple* data views, you must set the `Inverted` property of each `ChartDataView` object.

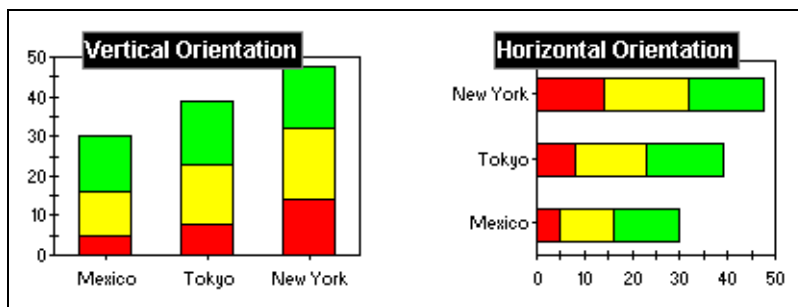


Figure 45 Normal and inverted orientation.

This property is ignored for circular charts.

### 6.3.2 Changing Axis Direction

Use the `Reversed` property of `JCAxis` to reverse the direction of an axis. By default, `Reversed` is set to `false`.

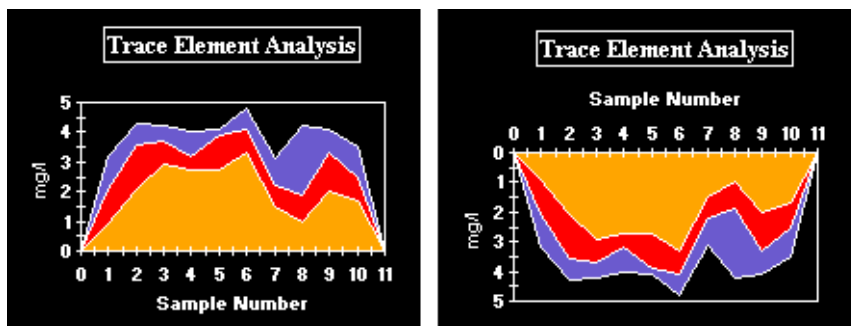


Figure 46 Two charts depicting a normal and reversed y-axis.

For polar charts, data points with positive x-values will be displayed in a counterclockwise direction starting at the origin base. When the `XAxis.reversed` flag is `true`, positive x-values will be displayed clockwise.

## 6.4 Setting Axis Bounds

Normally a graph displays all of the data it contains. There are situations where only part of the data is to be displayed. This can be accomplished by fixing axis bounds.

### Min and Max

Use the `Min` and `Max` properties of `JCAxis` to frame a chart at specific axis values. The `MinIsDefault` and `MaxIsDefault` properties allow the chart to automatically determine axis bounds based on the data bounds.

## 6.5 Customizing Origins

The chart can choose appropriate origins for the axes automatically, based on the data. It is also possible to customize how the chart determines the origin, or to directly specify the coordinates of the origin.

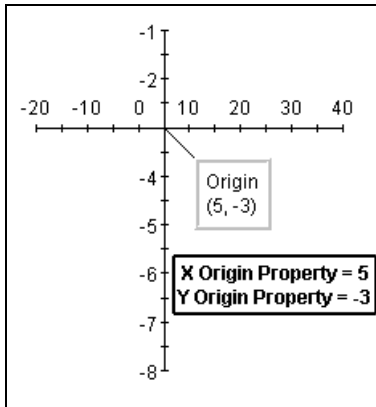


Figure 47 Defining origins for x-axes and y-axes.

### Origin Placement

The easiest way to customize an origin is by controlling its placement, using the Axes' `OriginPlacement` property. It has four possible values: `AUTOMATIC`, `ZERO`, `MIN`, and `MAX`. When set to `AUTOMATIC`, the origin is placed at the axis minimum or at zero, if the data contains positive and negative values or is a bar chart. `ZERO` places the origin at zero, `MIN` places the origin at the minimum value on the axis, and `MAX` places the origin at the maximum value on axis.

### Origin Coordinates

When the origin of a coordinate must be set to a value different from the default (0,0), use the Axes' `Origin` property. The `OriginIsDefault` property allows the chart to automatically determine the origin coordinate based on the data.

**Note:** When an origin coordinate is explicitly set or fixed, the chart ignores the `OriginPlacement` property.

## 6.6 Logarithmic Axes

Axis annotation is normally interpreted and drawn in a *linear* fashion. It is also possible to set any axis to be interpreted *logarithmically* (log base 10), as shown in the following image. Logarithmic axes are useful for charting certain types of scientific data.

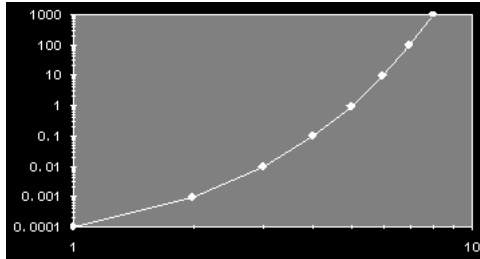


Figure 48 Logarithmic x-axes and y-axes.

Because of the nature of logarithmic axes, they impose the following restrictions on the chart:

- any data that is less than or equal to zero is not graphed (it is treated as a *data hole*), since a logarithmic axis only handles data values that are greater than zero. For the same reason, axis and data minimum/maximum bounds and origin properties cannot be set to zero or less.
- axis numbering increment, ticking increment, and precision properties have no effect when the axis is logarithmic.
- the x-axis of bar and stacking bar charts cannot be logarithmic.
- the annotation method for the x-axis cannot be `PointLabels` or `TimeLabels`.

### Specifying a Logarithmic Axis

Use the `Logarithmic` property of `JCAxis` to make an axis logarithmic.

**Note:** Pie charts are not affected by logarithmic axes.

## 6.7 Titling Axes and Rotating Axis Elements

Adding a title to an axis clarifies what is charted along that axis. You can add a title to any axis, and also rotate the title or the annotation along the axis, as shown below.

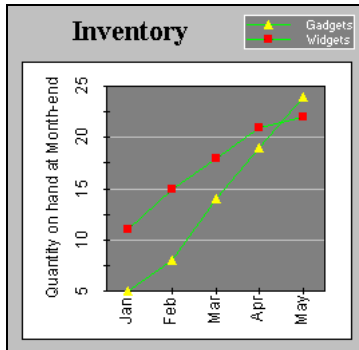


Figure 49 Rotated axis title and annotation.

### Adding an Axis Title

Use the `Title` property to add a title to an axis. It sets the `JCAxisTitle` object associated with the `JCAxis`. `JCAxisTitle` controls the appearance of the axis title. `JCAxisTitle`'s `Text` property specifies the title text.

### Axis Title Rotation

Use the `Rotation` property of `JCAxisTitle` to set the rotation of the title. Valid values are defined in `ChartText`: `DEG_0` (no rotation), `DEG_90` (90 degrees counterclockwise), `DEG_180` (180 degrees), and `DEG_270` (270 degrees).

### Rotating Axis Annotation

Use the `AnnotationRotation` property of `JCAxis` to rotate the axis annotation to either 90, 180, or 270 degrees clockwise from the horizontal position. 90-degree rotation usually looks best on a right-hand side axis.

This property can also be used to rotate the annotation at any other specified angle, if it is set to `AnnotationRotation.ROTATION_OTHER`. The new angle will be determined by the `AnnotationRotationAngle` property's value. By default, the angle is 0.0 degrees.

It is important to know that some fonts may not draw properly at an angle; therefore, they might not be visually appealing. If you are using rotated labels, your font choice should be made with care.

**Note:** In some cases, rotated labels will overlap. When labels overlap, the `visible` property for the higher indexed label is cleared, and only the lower indexed label is visible.

## 6.8 Gridlines

Displaying a grid on a chart can make it easier to see the exact value of data points. Gridlines are hidden by default. To show gridlines, set the `GridVisible` property to `true`. You can customize the spacing between gridlines as well as the appearance of the lines.

### Gridlines in Rectangular Charts

In a rectangular charts, such as plot and bar charts, gridlines are laid out in standard grid format. Horizontal gridlines are a property of the y-axis. Vertical gridlines are a property of the x-axis. Each can be given unique spacing and style properties, as described later in this section.

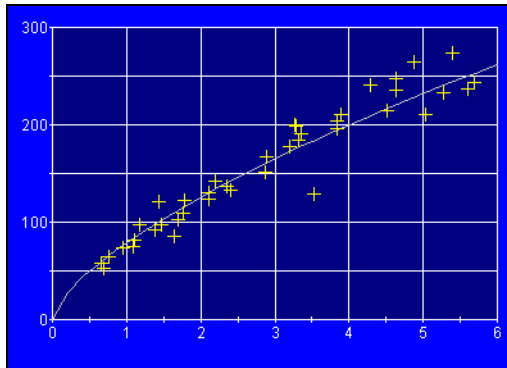


Figure 50 Gridlines in a rectangular chart.

### Gridlines in Polar Charts

For polar charts, y-axis gridlines are circular while x-axis gridlines are radial lines from the center to the outside of the plot area. Each can be given unique spacing and style properties, as described later in this section.

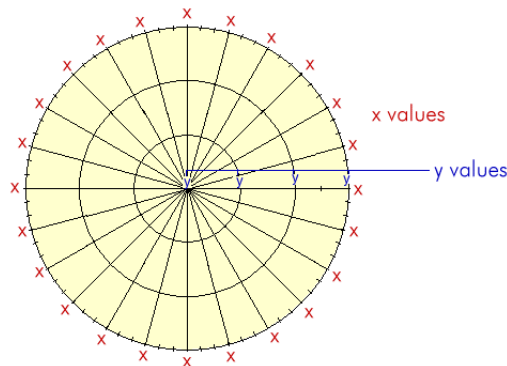


Figure 51 Circular Gridlines.

## Gridlines in Radar Charts

For radar and area radar charts, y-axis gridlines are represented as concentric circles around the center of the chart. If you would prefer webbed gridlines, where the lines between radial gridlines are drawn straight rather than as arcs, you need to set the `RadarCircularGrid` property to `false`.

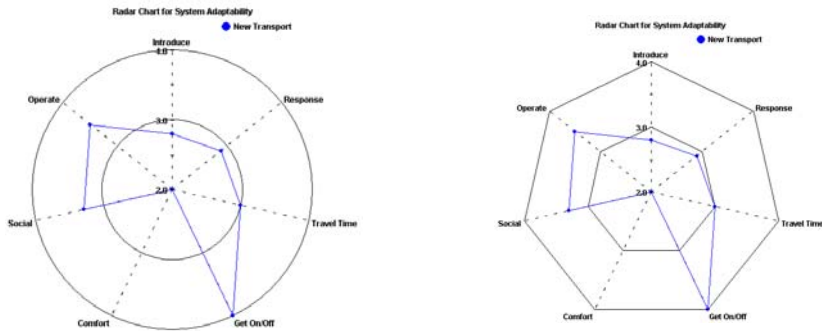


Figure 52 Circular gridlines vs webbed gridlines.

## Grid Spacing

By default, the spacing between gridlines automatically corresponds with the axis annotations. You can customize the interval between gridlines for each axis. To specify the gridline spacing for an axis, you set the `GridSpacing` property for the axis and specify the interval between gridlines as a positive `double` (setting a value of zero means gridlines are not shown).

For example:

```
// Set grid spacing for the x-axis  
axis.setGridSpacing(10);
```

## Gridline Attributes

You can customize the line pattern, thickness, and color of the gridlines by axis. To set the line attributes, you set the `GridStyle` property for each axis.

For example, the following code fragment changes the line color to green:

```
otherXAxis.setGridVisible(true);  
otherXAxis.getGridStyle().getLineStyle().setColor(Color.green);  
otherYAxis.setGridVisible(true);  
otherYAxis.getGridStyle().getLineStyle().setColor(Color.green);
```



## 6.9 Adding a Second Y-Axis

There are two ways to create a second y-axis on a chart. The simplest way is to define a numeric relationship between the two y-axes, as shown in the following illustration. Use this to display a different scale or interpretation of the same graph data.

**Note:** For polar, radar, and area radar charts, there is no second y-axis.

### Defining Axis Multiplier

Use the `Multiplier` property to define the multiplication factor for the second axis. This property is used to generate axis values based on the first axis. The multiplication factor can be positive or negative.

### Using a Constant Value

Use the `Constant` axis property to define a value to be added to or subtracted from the axis values generated by `Multiplier`.

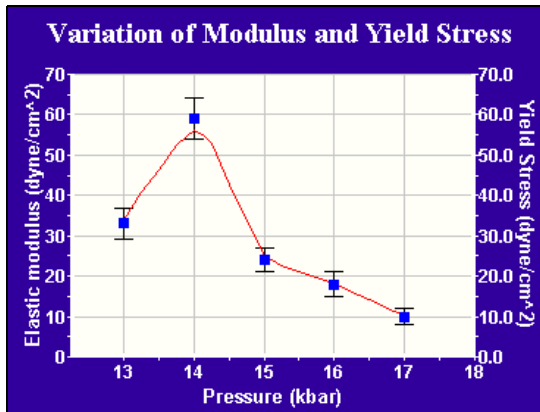


Figure 53 Chart containing multiple y-axes.

In some cases, it may be desirable to show two sets of data in the same chart that are plotted against different axes. JClass Chart supports this by allowing each `DataView` to specify its own `XAxis` and `YAxis`. For example, consider a case in which a second data set `d2` is to be plotted against its own y-axis. A `JCAxis` instance must be created and added to the `JCChartArea`, as shown:

```
// Create a new axis and set it vertical
otherYAxis = new JCAxis();
otherYAxis.setVertical(true);

// Add it to the list of new axes in the chart area
c.getChartArea().setYAxis(1, otherYAxis);
// Add it to the data view
d2.setYAxis(otherYAxis);
```

**Hiding the Second Axis**

Set the `Visible` property to `false` to remove it from display. By default, it is set to `true`.

**Other Second-Axis Properties**

All axes have the same features. Any property can be set on any axis.

# Defining Text and Style Elements

*Header and Footer Titles* ■ *Legends* ■ *Chart Labels*  
*Chart Styles* ■ *Outline Style* ■ *Hole Styles* ■ *Borders* ■ *Fonts* ■ *Colors*  
*Positioning Elements on the Chart Object* ■ *3D Effect* ■ *Anti-Aliasing*

This chapter describes the different formatting elements available within JClass Chart, and how they can be used.

**Note:** If you are developing your chart application using one of the JClass Chart beans, go to Chapter 13, [JClass Chart Beans](#) instead.

## 7.1 Header and Footer Titles

A chart can have two titles, called the header and footer. A title consists of one or more lines of text with an optional border. By default they are `JLabel` instances and behave accordingly. A `JLabel` object can display text, an image, or both. By default, labels are vertically centered in their display area. Text-only labels are left-aligned, while image-only labels are horizontally centered by default.

You can change the text alignment by setting the `HorizontalAlignment` and `VerticalAlignment` properties of `JLabel`. You can also customize the title border, font, colors, and the size and position of the title. For more information, see Section 7.7, [Borders](#), Section 7.8, [Fonts](#), Section 7.9, [Colors](#), and Section 7.10, [Positioning Elements on the Chart Object](#).

## 7.2 Legends

A legend shows the visual attributes (or `ChartStyle`) used for each series in the chart, with text that labels the series.

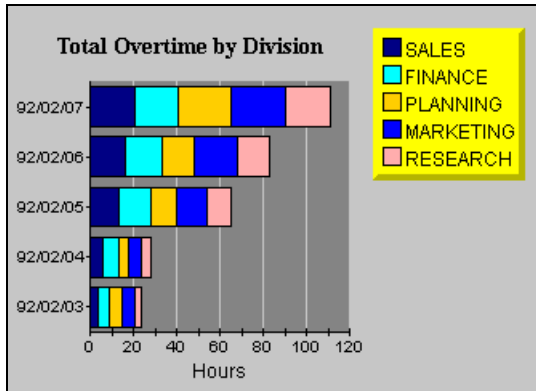


Figure 54 Vertically oriented legend anchored NorthEast.

### 7.2.1 Types of Legends

There are two types of legend objects: `JCGridLegend` (the default) for a single-column layout and `JCMultiColLegend` for a multiple-column layout. If these legends do not provide the desired functionality, you can customize the legend using the `JCLegend Toolkit`. For more information, see Section 7.2.3, [Creating Custom Legends with the JCLegend Toolkit](#).

#### Single-Column Legends

The classic single-column legend layout is provided by `JCGridLegend`. This is the default layout in `JClass Chart`.

#### Multi-Column Legends

Multi-column legend layout is available using `JCMultiColumnLegend`. To designate this layout, follow these steps:

1. Create an instance.
2. Set the number of rows and columns.
3. Set the `legend` property of the chart to this instance.

For example:

```
JCMultiColLegend mc1 = new JCMultiColumnLegend();
mc1.setNumColumns(2);
myChart.setLegend(mc1);
```

This example creates a legend for the current chart that has two columns. The number of rows depends on the number of items in the legend. To fix the number of rows, use `setNumRows()`. Both the number of rows and the number of columns are variable by default.

To reset the number of rows and columns to a variable state after they have been fixed, call the appropriate set method with a negative value. If both the `NumRows` and `NumColumns` properties are set to fixed values, the legend will be of that exact size and will ignore any extra items.

## 7.2.2 Customizing Legends

You can customize the series label and positioning. The legend is a `JComponent`, and all properties such as border, colors, font, and so on, apply. You can also specify the maximum width of a column in the legend.

### 7.2.2.1 Displaying Series Labels in the Legend

The legend displays the text contained in the `Label` property of each `Series` in a `DataView`. The `VisibleInLegend` property of the series determines whether the `Series` will appear in the legend. `SeriesLabels` support the use of HTML tags. The use of HTML tags overrides the default `Font` and `Color` properties of the label. Please note that HTML labels may not work with PDF, PS, or PCL encoding.

### 7.2.2.2 Displaying Marker or Threshold Labels in the Legend

If you use markers or thresholds in your chart, you can choose to display their labels in the legend using the `VisibleInLegend` properties from `JCMarker` and `JCThreshold` respectively. When `true`, the text contained in the `Label` property of the marker or threshold is displayed in the legend. `VisibleInLegend` is `false` by default.

### 7.2.2.3 Setting the Legend Orientation

Use the legend `Orientation` property to lay out the legend horizontally or vertically.

### 7.2.2.4 Positioning the Legend

You can use the legend `Anchor` property to specify where to position the legend relative to the `ChartArea`. You can select from eight compass points around the `ChartArea`. Valid values are: `JCLegend.NORTH`, `JCLegend.SOUTH`, `JCLegend.EAST`, `JCLegend.WEST`, `JCLegend.NORTHWEST`, `JCLegend.SOUTHWEST`, `JCLegend.NORTHEAST`, and `JCLegend.SOUTHEAST`. The default value is `JCLegend.EAST`.

To specify an absolute position for the legend, you set the `LayoutHints` property from `JCChart` and provide coordinates. For more information, see Section 7.10, [Positioning Elements on the Chart Object](#).

### 7.2.2.5 Setting the Width of the Legend and its Columns

If the legend text is very long, you may find that by default the legend becomes very wide, leaving proportionally less room for the chart itself. You can improve the balance between chart and legend by controlling the width of the legend. You have two choices

for setting the width. You can set the width of the legend explicitly and allow the columns within the legend to be sized automatically, or you can set the column widths and allow the legend width to be calculated.

### Specifying the Legend Width

To set the width of the entire legend, you set the `LayoutHints` property from `JCChart` and provide the width of the legend rectangle. For example, the following code snippet sets the width of the legend to 200 pixels:

```
chart.setLayoutHints(chart.getLegend(),
                    new Rectangle(Integer.MAX_VALUE, Integer.MAX_VALUE,
                                   200, Integer.MAX_VALUE));
```

`Integer.MAX_VALUE` means that the dimension is dynamic. In the above example, there are no restrictions on the positioning of the legend or on the height dimension. For more information, see Section 7.10, [Positioning Elements on the Chart Object](#).

### Specifying Column Widths

To set the width of columns within the legend, you set the `MaxItemTextWidth` property from `JCLegend` and specify the width in pixels as a non-negative `Integer`. By default, the value is `Integer.MAX_VALUE`, which means the width is dynamic.

For example, the following code sets the width for each of the columns in the legend to 100 pixels.

```
legend.setMaxItemTextWidth(100);
```

To specify different widths for columns in a multicolumn legend, you need to provide an additional parameter that specifies the column number. For example, the following code specifies column widths of 50, 100, and 75 pixels for consecutive columns in a three-column legend:

```
legend.setMaxItemTextWidth(50, 0);
legend.setMaxItemTextWidth(100, 1);
legend.setMaxItemTextWidth(75, 2);
```

#### 7.2.2.6 Handling Truncated Text

You can set properties to control what happens when the length of the text exceeds the width of a column. By default, column text is aligned with the leading edge of the column (for example, it is aligned left in a left-to-right orientation). When text is truncated, the trailing text (the rightmost text in a left-to-right orientation) is hidden and an ellipsis is displayed in its place. You can modify this behavior by setting the `JCLegend` properties described below.

To change the text alignment, you set the `ItemTextAlignment` property and specify the value using one of the following enumerations: `SwingConstants.LEFT`, `SwingConstants.RIGHT`, `SwingConstants.CENTER`, `SwingConstants.LEADING` (*default*), or

`SwingConstants.TRAILING`. For example, the following code causes text to be right aligned for all columns except the second column (column 1), where the text is centered:

```
legend.setItemTextAlignment(SwingConstants.RIGHT);  
legend.setItemTextAlignment(SwingConstants.CENTER, 1);
```

To change how the text is truncated, you set the `TruncateMode` property. The following table shows the possible values followed by how the text would appear:

<code>JCLegend.TRUNCATE_LEFT</code>	<code>...text</code>
<code>JCLegend.TRUNCATE_MIDDLE</code>	<code>text...text</code>
<code>JCLegend.TRUNCATE_RIGHT</code>	<code>text...</code>
<code>JCLegend.TRUNCATE_END</code>	<code>...text...</code>
<code>JCLegend.TRUNCATE_LEADING</code>	In a left-to-right orientation, same as <code>JCLegend.TRUNCATE_LEFT</code> . In a right-to-left orientation, same as <code>JCLegend.TRUNCATE_RIGHT</code> .
<code>JCLegend.TRUNCATE_TRAILING</code> ( <i>default</i> )	In a left-to-right orientation, same as <code>JCLegend.TRUNCATE_RIGHT</code> . In a right-to-left orientation, same as <code>JCLegend.TRUNCATE_LEFT</code> .

For example, the following code causes text to be truncated on the right for all columns, except for the third column (column 2), where the ends are truncated:

```
legend.setTruncateMode(JCLegend.TRUNCATE_RIGHT);  
legend.setTruncateMode(JCLegend.TRUNCATE_END, 2);
```

To stop the ellipsis from being displayed, you set the `UseEllipsisWhenTruncating` property to `false`. There will be no visual indication that text is hidden. This property always applies to all columns.

You can also choose to display the entire legend item text in a tooltip whenever the mouse hovers over a legend item. The tooltip appears whether or not the legend text is truncated. To activate the tooltips, set the `ItemTextToolTipEnabled` property to `true`. This property always applies to all columns.

### 7.2.3 Creating Custom Legends with the JCLegend Toolkit

The JCLegend Toolkit allows you the freedom to design your own legend implementations. The options range from simple changes, such as affecting the order of the items in the legend, to providing more complex layouts.

The JCLegend Toolkit consists of a `JCLegend` class that can be subclassed to provide legend layout rules and two interfaces: `JCLegendPopulator` and `JCLegendRenderer`. `JCLegendPopulator` is implemented by classes wishing to populate a legend with data,

and `JCLegendRenderer` is implemented by a class that wishes to help render the legend's elements according to the user's instructions. Examples of how to use the `JCLegend Toolkit` are provided in *JCLASS\_HOME/examples/chart/legend/*.

`JCChartLegendManager` is the class used by `JClass Chart` to implement both the `JCLegendPopulator` and `JCLegendRenderer` interfaces, and to provide a built-in mechanism for itemizing range objects in a legend.

### 7.2.3.1 Custom Legends – Layout

The `Legend Toolkit` allows you to create custom legend implementations. `JCLegend` is an abstract class that can be subclassed by users wishing to customize the legend layout or other legend behavior.

To provide a custom layout, override the method:

```
public abstract Dimension layoutLegend(List itemList, boolean vertical,
                                      Font useFont)
```

The `itemList` argument is a `List` containing a `Vector` for each data view contained in the chart. Each of these sub-vectors contains one `JCLegendItem` instance for each series in the data view and one instance for the data view title.

The `vertical` argument is `true` if the orientation of the legend is vertical and `false` if the orientation of the legend is horizontal.

The `useFont` argument contains the default font to use for the legend.

Each item in the legend consists of a text portion and a symbol portion. For example, in a `Plot Chart`, the text portion is the name of the series, and is preceded by the symbol used to mark a point on the chart. For the title of the data view, the text portion is the name of the data view and there is no symbol.

`JCLegendItem` is a class that encapsulates an item in the legend with the properties.

Property name	Description
<code>Point pos;</code>	position of this legend item within the legend
<code>Point symbolPos;</code>	position of the symbol within the legend item
<code>Point textPos;</code>	position of the text portion within the legend item
<code>Dimension dim;</code>	full size of the legend item
<code>Dimension symbolDim;</code>	size of the symbol; provided by <code>JCLegend</code>
<code>Dimension textDim;</code>	size of the text portion; provided by <code>JCLegend</code>
<code>Rectangle pickRectangle;</code>	the rectangle to use for pick operations; optional



Property name	Description
<code>int drawType;</code>	determines drawing type; one of <code>JCLegend.NONE</code> , <code>JCLegend.BOX</code> , <code>JCLegend.IMAGE</code> , <code>JCLegend.IMAGE_OUTLINED</code> , <code>JCLegend.CUSTOM_SYMBOL</code> , or <code>JCLegend.CUSTOM_ALL</code>
<code>Object itemInfo;</code>	data related to this legend item; is a <code>JCDataIndex</code> object containing the data view and series to which the legend item is related
<code>Object symbol;</code>	the symbol if other than the default type; usually null (means <code>drawLegendItem</code> decides)
<code>Object contents;</code>	the text portion; a <code>String</code> or <code>JCString</code>

When the `itemList` is passed to `layoutLegend`, it has been filled in with `JCLegendItem` instances representing each data series and data view title. These instances will have the `symbolDim`, `textDim`, `symbol`, `contents`, `itemInfo`, and `drawType` already filled in.

The value of `drawType` will determine whether a particular default symbol type will be drawn or whether user-provided drawing methods will be called.

The `layoutLegend()` method is expected to calculate and fill in the `pos`, `symbolPos`, `textPos`, and `dim` fields. Additionally, the method must return a `Dimension` object containing the overall size of the legend. Optionally, it may also calculate the `pickRectangle` member of the `JCLegendItem` class. The `pickRectangle` is used in pick operations to specify the region in the legend that is associated with the series that this legend item represents. If left null, a default `pickRectangle` will be calculated using the `dim` and `pos` members.

Any of the public methods in the `JCLegend` class may be overridden by a user requiring custom behavior. One such method is:

```
public int getSymbolSize()
```

`getSymbolSize()` returns the size of the legend-calculated symbols to be drawn in the legend. Default `JCLegend` behavior sets the symbol size to be equal to the ascent of the default font that is used to draw the legend text. If you want to use a different symbol, you can override it. One possible implementation is to use a symbol size identical to that which appears on the actual chart.

### 7.2.3.2 Custom Legends – Population

`JCLegendPopulator` is an interface that can be implemented by any user desiring to populate the legend with custom items. This interface comprises two methods that need to be implemented:

```
public List getLegendItems(FontMetrics fm)
public boolean isTitleItem(JCLegendItem item)
```

`getLegendItems()` should return a `List` object containing any number of `Vector` objects where each `Vector` object represents one column in the legend. Each `Vector` object contains the `JCLegendItem` objects for that column. In `JClass Chart`, each column generally represents one data view.

`isTitleItem()` should return `true` or `false`, depending on whether the passed `JCLegendItem` object represents a title for the column. This is used to determine whether a symbol is drawn for a particular legend item.

If implemented, the legend should be notified of the new populator with the `setLegendPopulator()` method of `JCLegend`.

### 7.2.3.3 Custom Legends – Rendering

`JCLegendRenderer` is an interface that can be implemented by any user desiring to custom render legend items. This interface consists of four methods that need to be implemented:

```
public void drawLegendItem(Graphics gc, Font useFont, JCLegendItem thisItem)
public void drawLegendItemSymbol(Graphics gc, Font useFont, JCLegendItem
thisItem)
public Color getOutlineColor(JCLegendItem thisItem)
public void setFillGraphics(Graphics gc, JCLegendItem thisItem)
```

`JCLegendRenderer` also has the capacity to implement custom text objects for drawing, and is called when the legend cannot interpret an object placed in the `contents` field of the `JCLegendItem`. This interface consists of one method that needs to be implemented:

```
void drawLegendItemText (Graphics gc, Font useFont, JCLegendItem thisItem);
```

`drawLegendItem()` provides a way for a user to define a custom drawing routine for an entire legend item. It is called when a legend item's draw type has been set to `JCLegend.CUSTOM_ALL`.

`drawLegendItemSymbol()` provides a way for a user to define a custom drawing routine for a legend item's symbol. It is called when a legend item's draw type has been set to `JCLegend.CUSTOM_SYMBOL`.

`getOutlineColor()` should return the outline color to be used to draw the legend item's symbol. If `null` is returned, the legend's foreground color will be used.

`getOutlineColor()` is called when a legend item's draw type has been set to either `JCLegend.BOX` or `JCLegend.IMAGE_OUTLINED`.

`setFillGraphics()` should set the appropriate fill properties on the provided `Graphics` object for drawing the provided legend item. `setFillGraphics()` is called when the legend item's draw type has been set to `JCLegend.BOX`.

If implemented, the legend should be notified of the new renderer with the `setLegendRenderer()` method of `JCLegend`.

### 7.2.3.4 Examples of Simple Custom Legends

The easiest way to perform simple legend customizations is to extend an existing legend. The following example (taken from the *Reversed Legend* example in `JCLASS_HOME/examples/chart/legend/`) overrides the `JCChartLegendManager` class (the class that implements the `JCLegendPopulator` and `JCLegendRenderer` interfaces in `JClass Chart`) to reverse the order of the legend items. This class overrides the `getLegendItems()` method, first calling the superclass' method to get the list of legend items and then rearranging the order before returning the newly reversed list of legend items.

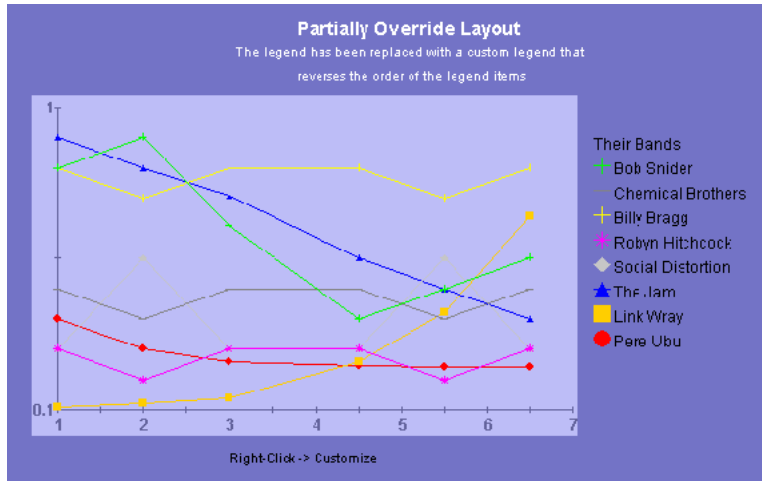


Figure 55 The *Reversed Legend* example reverses the order of the legend items.

Here is the *Reverse Legend* example code:

```
public ReverseLegend() {  
  
    setLayout(new GridLayout(1,1));  
  
    // replace standard legend with custom legend that reverses  
    // the order of the legend items  
    JCChart c = new JCChart(JCChart.PLOT);  
    ...  
    RevLegendManager legMan = new RevLegendManager(c);  
    c.getLegend().setLegendPopulator(legMan);  
    c.getLegend().setLegendRenderer(legMan);  
    c.getLegend().setVisible(true);  
    ...  
}  
  
/** RevLegendManager overrides the standard legend representation  
 * to reverse the drawing order of the legend items. It does this by  
 * overriding getLegendItems() method of the JCChartLabelManager
```

```

*   class to reverse the order of the items in the legend
*   vector.
*/

class RevLegendManager extends JCChartLegendManager
{
    RevLegendManager(JCChart chart)
    {
        super(chart);
    }

    /** Override getLegendItems(). Reverse order of items in legend
    *   vector.
    */
    public List getLegendItems(FontMetrics fm)
    {
        // get the list of legend items from the superclass
        List itemList = super.getLegendItems(fm);

        // reverse the list
        for (int i = 0; i < itemList.size(); i++) {
            List viewItems = (List) itemList.get(i);

            List reverseView = new Vector();
            for (int j = viewItems.size() - 1; j >= 0; j--) {
                JCLegendItem thisItem = (JCLegendItem) viewItems.get(j);

                // reverse items in list, but keep the title at the top.
                if (isTitleItem(thisItem)) {
                    reverseView.add(0, thisItem);
                } else {
                    reverseView.add(thisItem);
                }
            }
            itemList.set(i, reverseView);
        }
        // now that we've set up the list correctly, let the superclass
        // position it
        return itemList;
    }
}

```

The *Separator Legend* example in *JCLASS\_HOME/examples/chart/legend/* shows how to place a separator between the data view title and the series beneath it. Similar to the *Reversed Legend* example, the *Separator Legend* example overrides the `JCChartLegendManager` class.

In the *Separator Legend* example, a new `JCLegendItem` is inserted into the list after the data view title item as part of the `layoutLegend()` method. This new `JCLegendItem` has only its `textDimension` filled in with the size of the separator, but the actual contents field remains null – which is how one recognizes the separator when it is time to draw it.

The `drawType` field of the `JCLegendItem` is set to `JCLegend.CUSTOM_ALL` to ensure that the `drawLegendItem()` method will be called. Finally, the example returns the item list with the newly added item and lets the superclass do the positioning and sizing calculations.

The `drawLegendItem()` method is also overridden so that the separator can be drawn. Before drawing, however, it is first determined whether the provided legend item is, indeed, the separator created above.

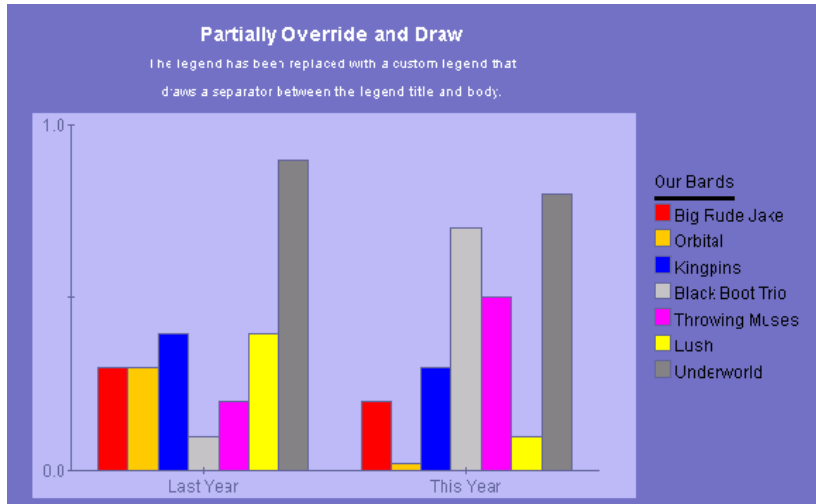


Figure 56 The *Separator Legend* example places a separator between the data view title and the series beneath it, and extends `JCLegendManager`.

Here is the *Separator Legend* example code:

```
public SeparatorLegend() {
    setLayout(new GridLayout(1,1));

    // replace standard legend with custom legend that draws a
    // separator between the title and the body
    JCChart c = new JCChart(JCChart.BAR);
    ...
    SepLegendManager sepMan = new SepLegendManager(c);
    c.getLegend().setLegendPopulator(sepMan);
    c.getLegend().setLegendRenderer(sepMan);
    c.getLegend().setVisible(true);
    ...
}

/** sepLegendManager overrides the standard legend populator and
 *  * renderer implementations to draw a separator between the legend
 *  * title and body. It does this by overriding the
```

```

* JCChartLegendManager's getLegendItem() method (to insert an item
* to take the place of a separator) and drawLegendItem() (to draw
* the separator) methods.
*/
public class SepLegendManager extends JCChartLegendManager
{

    public SepLegendManager(JCChart chart)
    {
        super(chart);
    }

    /** Override getLegendItems() to insert separator item into
    * legend vector.
    */
    public List getLegendItems(FontMetrics fm)
    {
        // get the list of legend items from the superclass
        List itemList = super.getLegendItems(fm);

        // go through the list to find the spot for the separator
        for (int i = 0; i < itemList.size(); i++) {
            List viewItems = (List) itemList.get(i);

            for (int j = 0; j < viewItems.size(); j++) {
                JCLegendItem thisItem = (JCLegendItem) viewItems.get(j);

                // Insert separator item after title item
                // our separator is identified by having null contents
                // but an existing text dimension. Make the separator as
                // wide as the text portion of the title.

                if (isTitleItem(thisItem)) {
                    JCLegendItem newItem = new JCLegendItem();
                    boolean vertical = chart.getLegend().getOrientation()
JCLegend.VERTICAL;
                    if (vertical) {newItem.textDim = new
Dimension(thisItem.textDim.width, 3);

                        } else {
                            newItem.textDim = new Dimension(3, thisItem.textDim.height);
                        }
                    // make sure to set draw type as CUSTOM_ALL so that
                    // drawLegendItem() will be called.
                    newItem.drawType = JCLegend.CUSTOM_ALL;
                    viewItems.add(j+1, newItem);
                    break;
                }
            }
        }

        // now that the list is set up, let the superclass worry about
        // positioning everything
        return itemList;
    }
}

```

```

/** Override drawLegendItem() to draw the separator item
 * when encountered.
 */
public void drawLegendItem(Graphics gc, Font useFont,
                           JCLegendItem thisItem)
{
    // if our separator, draw it
    if (thisItem.contents == null && thisItem.textDim != null) {
        if (gc.getColor() != getForeground())
            gc.setColor(getForeground());

        gc.fillRect(thisItem.pos.x + thisItem.textPos.x,
                    thisItem.pos.y + thisItem.textPos.y,
                    thisItem.textDim.width,
                    thisItem.textDim.height);
    }
}

```

Remember to use the `setLegendPopulator()` and `setLegendRenderer()` methods of the `JCLegend` class to notify the legend of the new class.

### 7.2.3.5 Examples of Complex Legends

More complex customizations are also possible. Legends that require full-scale changes to the rules of layout can override the `JCLegend` class and create their own implementation. Have a look at [JCLASS\\_HOME/examples/chart/legend/FlowLegend](http://JCLASS_HOME/examples/chart/legend/FlowLegend) for an example of a custom legend layout.

## 7.3 Chart Labels

Chart labels allow you to add more information to your chart. There are static labels that display continuously and interactive labels that pop-up when a cursor moves over a data item. Labels can be attached to different parts of a chart: absolute coordinates, coordinates in the plotting area, or a specific data item. To see a wide range of label uses, browse the demos in the [JCLASS\\_HOME/demos/chart/labels/](http://JCLASS_HOME/demos/chart/labels/) directory.

### 7.3.1 Label Implementation

The list of labels is managed by the `ChartLabelManager`. This property is initially null. By calling `getChartLabelManager()`, `JClass Chart` will create a manager class with an empty list of labels. When you create a label, you must add it to the manager with `addChartLabel()`. Labels are instances of the `JCChartLabel` class.

### 7.3.2 Adding Labels to a Chart

Labels are added to a chart in two ways: with the `AutoLabels` property of `ChartDataView`, or by attaching an instance of `JCChartLabel` to a chart element.

Individual labels are attached in three ways: to coordinates on the chart (`ATTACH_COORD`); coordinates on the plot area (`ATTACH_DATACOORD`); or to a data item (`ATTACH_DATAINDEX`). Interactive labels must use the `ATTACH_DATAINDEX` method.

Each label on the chart below uses a different attachment method. The “Point(100,50)” label, is attached to coordinates originating from the top left corner of the chart.

“Value(2,220)” is attached to axes coordinates, and “Data(Set0,Point2)” is attached to a specific data item.

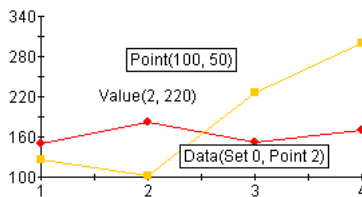


Figure 57 Adding chart labels.

#### Attaching a Label to a Data Item

To attach a label to a point, bar or slice, set the `AttachMethod` property to `ATTACH_DATAINDEX`. The labels move with the data element; the labels also move when the chart is resized. Note that the points and series are zero-based. The following example puts a label on a chart next to the fourth data point in the second data series.

```
c1 = new JCChartLabel("Fourth data point");
c1.setDataIndex(new JCDataIndex(view, series, 1, 3));
c1.setAttachMethod(JCChartLabel.ATTACH_DATAINDEX);
c1.setAnchor(JCChartLabel.AUTO);
chart.getChartLabelManager().addChartLabel(c1)
```

#### Attaching a Label to Chart Coordinates

To attach a label to a point on the chart, set the `AttachMethod` property to `ATTACH_COORD`. The coordinate origin for this method is the top left corner of the chart.

```
JCChartLabel c1 = new JCChartLabel("Point( 100, 50 )");
c1.setAttachMethod( JCChartLabel.ATTACH_COORD );
c1.setCoord( new Point( 100, 50 ) );
chart.getChartLabelManager().addChartLabel(c1)
```

#### Attaching a Label to Plot Area Coordinates

To attach a label to coordinates on the plot area, set the `AttachMethod` property to `ATTACH_DATACOORD`. The plot area is defined by the chart's x-axis and y-axis.



The following example places a label in the plot area at x-value 2.5, y-value 160.

```
JCChartLabel c1 = new JCChartLabel("Attached to the data coordinate",
                                   false);
c1.setDataCoord( new JCDataCoord( 2.5, 160 ) );
c1.setAnchor( JCChartLabel.NORTH );
c1.setAttachMethod( JCChartLabel.ATTACH_DATACoord );
c1.setBorderType( Border.ETCHED_OUT );
c1.setBorderWidth( 5 );
chart.getChartLabelManager().addChartLabel(c1)
```

### 7.3.3 Interactive Labels

You can have labels pop-up in your chart when the mouse cursor dwells over a particular point, bar, or slice contained in your chart. For example, in the following figure, the number “225” appears on top of the bar as the cursor passes over it, to indicate the value of the bar.

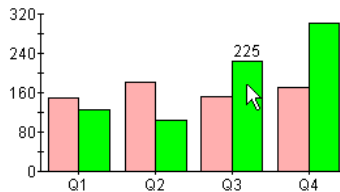


Figure 58 Bar chart displaying a dwell label.

In JClass Chart, these interactive labels are called *dwell labels*. You can use the `AutoLabel` property to set up a complete series of dwell labels.

#### Automatically Generated Labels

By default, the `AutoLabel` property of `ChartDataView` will generate a complete series of chart labels. It attaches chart labels to every data index.

The following code adds automatic dwell labels to the data:

```
chart.getDataView(0).setAutoLabel(true);
```

#### Adding Individual Dwell Labels

Attaching an individual dwell label follows the same procedure as attaching a static label to a data item, except that the `DwellLabel` property is set to `true`:

```
JCChartLabel c1 = new JCChartLabel();
c1.setDwellLabel( true );
```

A dwell label can only be used when the `AttachMethod` property is set to `ATTACH_DATAINDEX`.

### 7.3.4 Adding and Formatting Label Text

`JCChartLabel` is just a holder for any `JComponent`. By default it is a `JLabel` instance, and text can be set the same way you would set text on a `JLabel`. You can access the component portion of the chart label with the `getComponent()` method.

`JLabels` support the use of HTML tags. The use of HTML tags overrides the default `Font` and `Color` properties of the label. Please note that HTML labels may not work with PDF, PS, or PCL encoding.

#### Adding Label Text

You can add text to a label by passing it to the constructor, or by using the `Text` property. To add text to a label when it is constructed, include the text in the constructor's argument, as follows:

```
JCChartLabel c1 = new JCChartLabel("I'm a Label", false);
```

To add text using the `Text` property, use the `setText` method, as follows:

```
((JLabel)c1.getComponent()).setText("I'm a Label");
```

#### Formatting Label Text

```
Font f = new Font("timesroman", Font.BOLD, 24);  
c1.getComponent().setFont(f)
```

`JComponent` properties such as fonts, borders, and colors are set in the same manner.

### 7.3.5 Positioning Labels

The `Anchor` property determines the position of the label, relative to the point of attachment. The valid constants are:

- `JCChartLabel.NORTHWEST`
- `JCChartLabel.NORTH`
- `JCChartLabel.NORTHEAST`
- `JCChartLabel.EAST`
- `JCChartLabel.SOUTHEAST`
- `JCChartLabel.SOUTH`
- `JCChartLabel.SOUTHWEST`
- `JCChartLabel.WEST`

The following example shows the syntax:

```
c1.setAnchor(JCChartLabel.EAST);
```

### 7.3.6 Adding Connecting Lines

You can add lines that connect a label to its point of attachment. This can help the end-user pinpoint what a label refers to on a chart.



Figure 59 An example of a connecting line.

To add a connecting line to a label, set the `Connected` property to `true`, as follows:

```
cl.setConnected( true );
```

## 7.4 Chart Styles

Chart styles define all of the visual attributes of how data appears in the chart, including:

- Lines and points in plots and financial charts.
- Color of each bar in bar charts.
- Slice colors in pie charts.
- Color of each filled area in area charts.

Each series in a data view has its own `JCChartStyle` object; as new series are added, new `JCChartStyle` objects are created automatically by the chart. `JClass Chart` *automatically* defines a set of visually different styles for up to 13 series, so while you can customize any chart style, you may not need to.

**Note:** If you are using the targeted data model, you can change the default chart styles by implementing the appropriate `StyleDataSet` in your data set implementation. For more information, see [StyleDataSet Interfaces](#), in Chapter 5.

Every `ChartStyle` has a `FillStyle`, a `LineStyle`, and a `SymbolStyle`. `FillStyles` are used for area, bar, candle, Hi-Lo, Hi-Lo-Open-Close, pie, and stacking bar charts. `LineStyle`s and `SymbolStyle`s are used for plots.

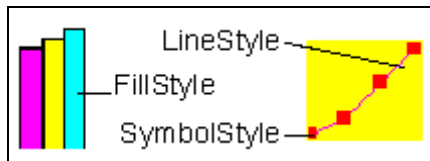


Figure 60 Types of ChartStyles available.

`ChartStyle` is an indexed property of `ChartDataView` that “owns” the `JCChartStyle` objects for that data view. It can be manipulated like any other indexed property, for example:

```
arr.setChartStyle(0, new JCChartStyle());
```

This adds the specified `ChartStyle` to the indexed property at the specified index. If the `ChartStyle` is null, the `JCChartStyle` at the specified point is removed. The following lists some of the other ways `ChartStyle` can be used:

- `getChartStyle(index)` – retrieves the chart style at the specified index
- `setChartStyle(List)` – replaces all existing chart styles
- `List getChartStyle()` – retrieves a copy of the array of chart styles

Normally, you will not need to add or remove `JCChartStyle` objects from the collection yourself. If a `JCChartStyle` object already exists when its corresponding series is created, the previously created `JCChartStyle` object is used to display the data in this series.

### Customizing Existing ChartStyles

Each `JCChartStyle` object contains three smaller objects that control different aspects of the style: `JCFillStyle`, `JCLineStyle`, and `JCSymbolStyle`.

The most common chart style sub-properties are repeated in `JCChartStyle`. For example, `FillColor` is a property of `JCChartStyle` that corresponds to the `Color` property of `JCFillStyle` object.

The following properties are repeated in the specified class:

- `LinePattern`, `LineWidth`, and `LineColor` repeat `JCLineStyle` properties.
- `SymbolShape`, `SymbolColor`, `SymbolSize`, and `SymbolCustomShape` repeat `JCSymbol` properties.
- `FillColor`, `FillPattern`, and `FillImage` repeat `JCFillStyle` properties.

### FillStyle

`JCFillStyle` controls the fills used in bar, pie, area, and candle charts. Its properties include `Color` and `Pattern`. Use `Pattern` to set the fill drawing pattern and `Color` to set the fill color. The default pattern is solid fill.

Available fill patterns include none, solid, 25%, 50%, 75%, horizontal stripes, vertical stripes, 45 degree angle stripes, 135 degree angle stripes, diagonal hatched pattern, cross hatched pattern, custom fill, custom paint, or, for bar charts only, custom stack fill. Custom fill and custom stack fill draw using the image set in the `Image` property. Custom paint draws using the `TexturePaint` object, which is set in the `CustomPaint` property.

**Note:** Filled areas are not supported for polar charts.

### LineStyle

`JCLineStyle` controls line drawing, used in line and Hi-Lo charts. Its properties are `Color`, `Pattern` and `Width`. Use `Pattern` to set the line drawing pattern, `Color` to set the line color, and `Width` to set the line width. Custom line patterns can be set with a `setPattern()` method that specifies the line pattern arrays to use.

## SymbolStyle

JCSymbolStyle controls the symbol used to represent points in a data series, used in plot or scatter plot charts. Its properties are Shape, Color and Size. Use Shape to set the symbol type, Size to set its size, and Color to set the symbol color.



Figure 61 Symbols available in JCSymbolStyle.

You can also provide a custom shape by implementing an abstract class JCSShape and assigning it to the CustomShape property.

## Customizing All ChartStyles

By looping through the JCChartStyle indexed property, you can quickly change the appearance of all of the bars, lines, or points in a chart. For example, the following code lightens all of the bars in a chart:

```
for (Iterator i = c.getDataView(1).getChartStyle().listIterator();
i.hasNext();)
{
    JCChartStyle cs = (JCChartStyle) i.next();
    JCFillStyle fs = cs.getFillStyle();
    fs.setColor(fs.getColor().brighten());
}
```

## 7.5 Outline Style

The ChartDataView's OutlineStyle property controls the outlines of area, stacking area, area radar, bar, stacking bar, and pie charts. It is of type JCLineStyle and thus the properties of the line can be controlled by getting the JCLineStyle object using getOutlineStyle() and setting its properties.

The default outline style is a solid line of width one in the chart area's foreground color.

## 7.6 Hole Styles

Hole values are data points that are invalid or missing in the data series, or that are defined as hole values in the data source. For more information, see [Hole Value](#) under [Text Data Formats](#), in Chapter 4.

By default, hole values are not drawn on the chart. If you want, you can choose to indicate that a hole value has occurred by specifying a *hole style*. A hole style is a `JCChartStyle` object that defines the line and fill styles to use when drawing hole values. Each data series can have a different style for holes. Hole styles are supported for plot, polar, and area charts.

**Note:** If hole styles are defined for the other chart types, the hole styles are ignored.

To specify the style to use for hole values in a data series, set the `HoleStyle` property in the `ChartDataViewSeries` object. The `HoleStyle` property takes a `JCChartStyle` object. The hole style objects for all the data series are stored in a `Vector` called `HoleStyles` in the `ChartDataView` object. You can access and manipulate the objects in the `HoleStyles` `Vector` in much the same way as described for `ChartStyle` in Section 7.4, [Chart Styles](#).

### 7.6.1 Example of Hole Styles

For example, in the following code sample (taken from the *holes.java* example located in *JCLASS\_HOME/examples/chart/datasource/*) when `true` is passed to the `createStyles()` method, it creates hole styles for each of the data series. Otherwise, it creates the basic chart styles. For hole styles, the line style is a long dash in the same color and width as the data series. The fill style is solid and uses a color that is defined in a `holeColor[]` array elsewhere in the code. Symbol styles are ignored; hole styles use the same symbol as the rest of the data series. The resulting `JCChartStyle` instances are added to a `styles[]` array.

```
...
// Create chart styles for hole display
holeStyles = createStyles(true);
dataView.setHoleStyle(holeStyles);
...
public List createStyles(boolean holeStyles)
{
    List styles = new ArrayList();
    for (int i = 0; i < yData.length; i++) {
        // Define line style
        int linePattern =
            holeStyles ? JCLineStyle.LONG_DASH : JCLineStyle.SOLID;
        JCLineStyle lineStyle = new JCLineStyle(1, colors[i],
            linePattern);

        // Define fill style
        Color fillColor = holeStyles ? holeColors[i] : colors[i];
        JCFillStyle fillStyle = new JCFillStyle(fillColor,
            JCFillStyle.SOLID);
    }
}
```

```

// Define symbol style
JCSymbolStyle symbolStyle = new JCSymbolStyle(symbolPatterns[i],
        colors[i], 6);

// Create the JCChartStyle instance
JCChartStyle chartStyle = new JCChartStyle(lineStyle, fillStyle,
        symbolStyle);
// Add the style to the styles array
styles.add(chartStyle);
}
return styles;
}

```

The following sections show examples of the various chart types before and after the hole styles defined above are applied. The charts all use the same data series. The hole values are specified using the constant `hole`.

```

// Y-axis values for each of the three data series
protected double yData[][] = {
    {7, 8, hole, 9, hole, 8, 7},
    {4, 6, hole, hole, hole, 6, 4},
    {1, hole, 2, 3, 2, hole, 1}
}

```

## 7.6.2 Hole Styles in Plot and Polar Charts

By default, when a hole value is encountered in a data series for a plot or polar chart, the hole value is not drawn and the lines that would have connected the missing value to valid data points on either side are omitted. The result is a broken line. When hole styles are specified, the chart uses the hole style's line attributes to connect valid data points.

**Note:** Two valid data points are required to draw a line. Therefore, if the first or last point in a series is a hole, no line is drawn even if a hole style is present.

The following figures show plot and polar charts before and after the hole styles defined in the preceding example are applied to the three data series.

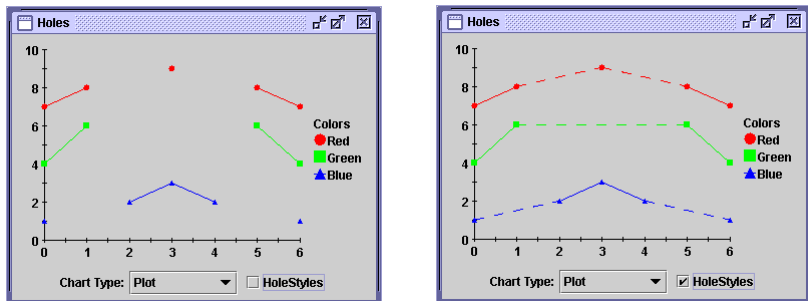
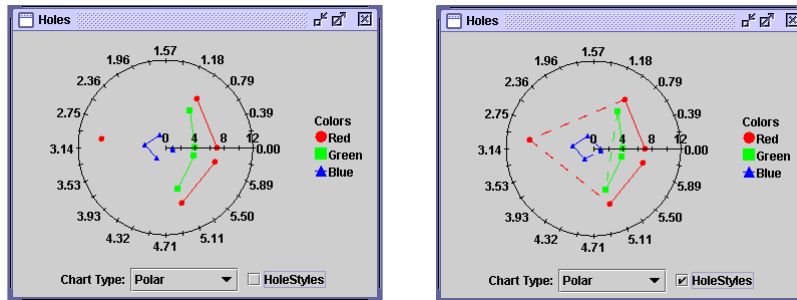


Figure 62 Plot chart before (left) and after (right) hole styles are applied.



### 7.6.3 Hole Styles in Area Charts

By default, when a hole value is encountered in a data series for an area chart, the hole value is not drawn and the region that spans the hole (that is, the region extending from the last valid point before the hole to the next valid point after the hole) is not filled. If a single valid data point is bounded by hole values on either side, the valid data point is drawn as a single line (see Figure 63, left image). When a hole style is specified, the fill attributes are used to fill the region between valid data points.

**Note:** Two valid data points are required to fill a region. Therefore, if the first or last point in a series is a hole, no fill style is used even if a hole style is present.

The following figure shows an area chart before and after the hole styles defined in the preceding example are applied to the three data series.

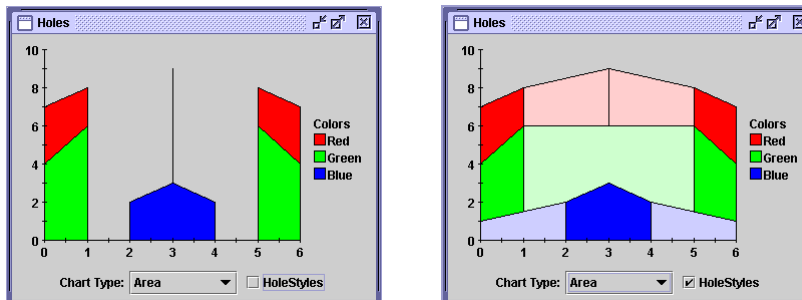


Figure 63 Area chart before (left) and after (right) hole styles are applied.



## 7.7 Borders

One way to highlight important information or improve the chart's appearance is to use a border. You can customize the border of the following chart objects:

- Header and Footer titles
- Legend
- ChartArea
- each ChartLabel added to the chart
- the entire chart

Border properties are set using the standard `JComponent` border facilities, `getBorder()` and `setBorder()`.

## 7.8 Fonts

A chart can have more impact when you customize the fonts used for different chart elements. You may also want to change the font size to make an element better fit the overall size of the chart. Any font available when the chart is running can be used. You can set the font for the following chart elements:

- Header and Footer titles
- Legend
- Axis annotation and title
- each ChartLabel added to the chart

### Changing a Font

Font properties are set using the standard `JComponent` font facilities, `getFont()` and `setFont()`. Use the font properties to set the font, style, and size attributes.

## 7.9 Colors

Color can powerfully enhance a chart's visual impact. You can customize chart colors using Java color names or RGB values. Using `JClass Chart Customizer` can make selecting custom colors quick and easy. Each of the following visual elements in the chart has a background and foreground color that you can customize:

- the entire chart
- header and footer titles
- legend
- chart area

- plot area (foreground colors JCCChartArea's AxisBoundingBox)
- each chart label added to the chart

Other chart objects have color properties too, including ChartDataView (bar/pie outline color) and ChartStyles.

### Color Defaults

All chart subcomponents are transparent by default with no background color. If made opaque, the legend, chart area and plot will inherit background color from the parent chart. The same objects will always inherit the foreground color from the chart. Headers and footers are independent objects and behave according to the rules of whatever object they are. However, once the application sets the colors of an element, they do not change when other elements' colors change.

### Specifying Foreground and Background Colors

Each chart element listed above has a Background and Foreground property that specifies the current color of the element. The easiest way to specify a color is to use the built-in colnames defined in java.awt.Color. The following table summarizes these colors:

Built-in Colors in java.awt.Color		
black	blue	cyan
darkGray	gray	green
lightGray	magenta	orange
pink	red	white
	yellow	

Alternately, you can specify a color by its RGB components, useful for matching another RGB color. RGB color specifications are composed of a value from 0 – 255 for each of the red, green and blue components of a color. For example, the RGB specification of Cyan is “0-255-255” (combining the maximum value for both green and blue with no red).

The following example sets the header background using a built-in color, and the footer background to an RGB color (a dark shade of Turquoise):

```
c.getHeader().setBackground(Color.cyan);

mycolor = new Color(95,158,160);
c.getFooter().setBackground(mycolor);
```

Take care not to choose a background color that is also used to display data in the chart. The default ChartStyles use all of the built-in colors in the following order: Red, Orange, Blue, Light Gray, Magenta, Yellow, Gray, Green, Dark Gray, Cyan, Black, Pink, and White. Note that JClass Chart will skip colors that match background colors. For

example, if the chart area background is Red, then the line, fill, and symbol colors will start at Orange.

For all charts, the foreground and background colors of the plot area are adjustable.

### **Transparency**

If the JClass Chart component is meant to have a transparent background, set the `Opaque` property to `False`; then generated GIFs and PNGs will also contain a transparent background.

## **7.10 Positioning Elements on the Chart Object**

Each of the main chart elements (Header, Footer, Legend, and ChartArea) has properties that control its position and size. While the chart can automatically control these properties, you can also customize the following:

- positioning of any element
- size of any element

When the chart controls positioning, it first allows space for the Header, Footer, and Legend, if they exist (size is determined by contents, border, and font). The ChartArea is sized and positioned to fit into the largest remaining rectangular area. Positioning adjusts when other chart properties change.

ChartLabels do not figure into the overall Chart layout. Instead, they are positioned above all other Chart elements.

### **Changing the Location and Size**

To specify the absolute location and size of a chart element, call `setLayoutHints()` in `JCChart` with the object you wish to move and a rectangle containing its desired X and Y location, width, and height. If you desire any of those values to be calculated rather than set, make them equal to `Integer.MAX_VALUE`.

For example, the following code sets the legend to be 200 pixels wide and 300 pixels high and places it at the x,y coordinate (0,150):

```
chart.setLayoutHints(legend, newRectangle(0,150,200,300))
```

Whereas this code allows the legend size to be dynamic, but places the legend at (0,150):

```
chart.setLayoutHints(legend, Rectangle(0,150,  
Integer.MAX_VALUE,Integer.MAX_VALUE, Integer.MAX_VALUE))
```

## 7.11 3D Effect

Data in bar, stacking bar, and pie charts can be displayed with a three-dimensional appearance using several `JCChartArea` properties:

- **Depth** – Specifies the apparent depth as a percentage of the chart’s width. No 3D effect appears unless this property is set greater than zero.
- **Elevation** – Specifies the eye’s position above the horizontal axis, in degrees.
- **Rotation** – Specifies the number of degrees the eye is positioned to the right of the vertical axis. This property has no effect on pie charts.

You can set the visual depth and the “elevation angle” of the 3D effect. You can also set the “rotation angle” on bar and stacking bar charts. `Depth`, `Rotation` and `Elevation` are all properties of the `ChartArea`.

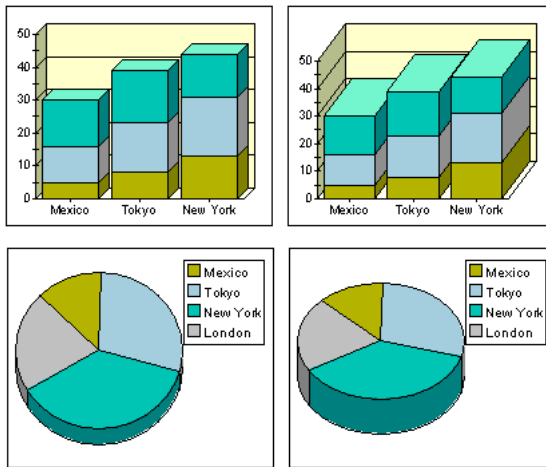


Figure 64 Four charts illustrating 3D effects.

## 7.12 Anti-Aliasing

Anti-aliasing is the process of smoothing out lines and curves to remove the pixelated appearance of text and graphics. The smoothing is done by padding pixels with intermediate colors. For example, a black and white image would be smoothed out using gray.

# Anti-aliasing off

# Anti-aliasing on

*Figure 65 The appearance of text with and without anti-aliasing.*

JClass Chart is equipped with the `AntiAliasing` property which can turn anti-aliasing on or off when the chart and its subcomponents are painted.

- `JCChart.ANTI_ALIASING_ON` turns on anti-aliasing for the chart;
- `JCChart.ANTI_ALIASING_OFF` turns off anti-aliasing for the chart;
- `JCChart.ANTI_ALIASING_DEFAULT`, which is the default value, ensures that the graphics object will be untouched with respect to anti-aliasing when the chart is painted.



# Defining Markers and Thresholds

Markers ■ Thresholds

You can enhance your clients' understanding of the data in your chart by implementing *markers* or *thresholds*. Markers are displayed as *lines* in the chart, and can be used to mark things like the average data value, data limits, or a particular coordinate in the chart. Thresholds cover a range of data values and are displayed as *areas* of color in the chart background. Thresholds can be used to identify and draw attention to groups of data values, such as values that fall below expectations, meet expectations, or exceed expectations. You can use both markers and thresholds in the same chart.

**Important:** This section assumes that you are using the underlying data model. If you are using the targeted data model, you need to implement the marker or threshold iterator. For more information, see Chapter 5, [Adding Data with the Targeted Data Model](#).

## 8.1 Markers

Markers enable you to draw lines in the plot area of the chart. For example, you could create a control chart by using parallel marker lines to represent an upper limit, an average, and a lower limit. Alternatively, you could create a crosshair at a particular coordinate to highlight a target value.

**Note:** Markers are not available for pie charts.

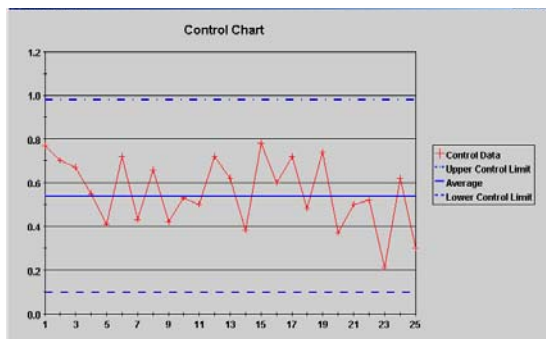


Figure 66 A control chart that uses markers to create the upper and lower limits and an average.

You can have a list of markers for each data view. The `ChartDataView` object maintains this list and has methods to add markers, remove markers, and get the current list of markers. For more information, see `ChartDataView` in the [JClass API Documentation](#).

After you create a `JCMarker` object, you associate the marker with a data view and an axis and then specify the value on that axis at which to draw the marker line. The following sections describe how to create a marker on the x-axis and the y-axis, as well as how to create a crosshair.

### 8.1.1 Creating X-axis Markers

To add an x-axis marker to the chart, you need to create a `JCMarker` object with its `AssociatedWithYAxis` property set to `false`, add it to the data view, and specify the value on the x-axis at which to draw the line. You can customize the line style and length, as well as control whether or not the markers are drawn on top of the data. For more information, see Section 8.1.4, [Customizing Markers](#).

The following code snippet demonstrates how to create an x-axis marker. The `JCMarker` constructor shown in this example takes the following parameters: a label (`String`) and a value at which to draw the marker line (`double`). The `AssociatedWithYAxis` property is set after the object is created. For more information, see `JCMarker` in [Appendix A.23](#) and in the [JClass API Documentation](#).

```
ChartDataView dataView = chart.getDataView(0);
JCMarker marker = new JCMarker("Marker at 2", 2.0);
marker.setAssociatedWithYAxis(false);
dataView.addMarker(marker);
```

#### X-axis Marker in a Rectangular Chart

In a rectangular chart (default orientation), associating a marker with the x-axis creates a vertical marker. By default, the vertical marker line spans the height of the plot area.

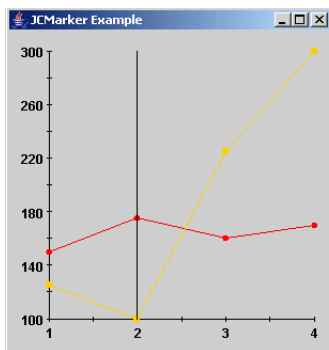


Figure 67 Plot chart (non-inverted) with an x-axis (vertical) marker.



If the chart orientation is inverted so that the x-axis is the vertical axis, the marker is drawn horizontally.

### X-axis Marker in a Circular Chart

In a circular chart, markers associated with the x-axis are drawn as radial lines.

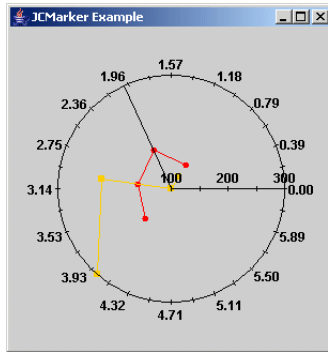


Figure 68 Polar chart with an x-axis (radial) marker.

## 8.1.2 Creating Y-axis Markers

To add an y-axis marker to the chart, you need to create a `JCMarker` object with its `AssociatedWithYAxis` property set to `true`, add it to the data view, and specify the value on the y-axis at which to draw the line. You can customize the line style and length, as well as control whether or not the markers are drawn on top of the data. For more information, see Section 8.1.4, [Customizing Markers](#).

The following code snippet demonstrates how to create a y-axis marker. The `JCMarker` constructor shown in this example takes the following parameters: a label (`String`) and a value at which to draw the marker line (`double`). The `AssociatedWithYAxis` property is set after the object is created. For more information, see `JCMarker` in [Appendix A.23](#) and in the [JClass API Documentation](#).

```
ChartDataView dataView = chart.getDataView(0);
JCMarker marker = new JCMarker("Marker at 200", 200.0);
marker.setAssociatedWithYAxis(true);
dataView.addMarker(marker);
```

### Y-axis Marker in a Rectangular Chart

In a rectangular chart (default orientation), associating a marker with the y-axis creates a horizontal marker. By default, the horizontal marker line spans the width of the plot area.

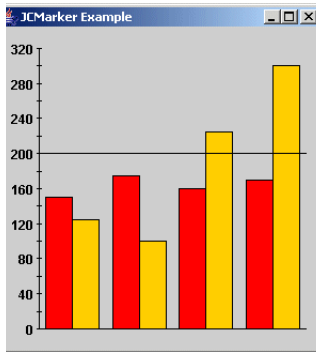


Figure 69 Bar chart (non-inverted) with a y-axis (horizontal) marker.

If the chart orientation is inverted so that the y-axis is the horizontal axis, the marker is drawn vertically.

### Y-axis Marker in a Circular Chart

In a circular chart, markers associated with the y-axis are drawn as circles (or arcs, if a line length is specified).

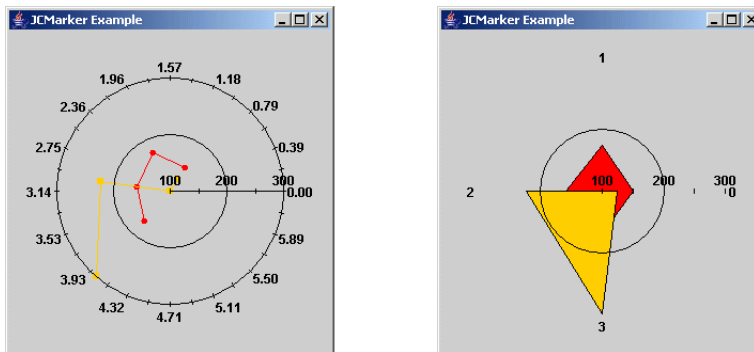


Figure 70 Polar chart and an area radar chart displaying y-axis markers as circles.

### Y-axis Marker in a Webbed Chart

In radar charts, if webbed gridlines are drawn, then a y-axis marker is displayed as a web shape rather than a circular shape. For more information, see [Gridlines](#), in Chapter 6.

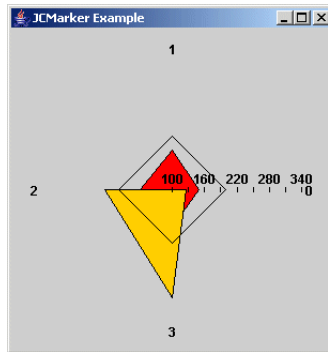


Figure 71 Area radar chart displaying a y-axis marker drawn in a web-like fashion.

### 8.1.3 Creating a Crosshair with Markers

To create a crosshair, you add two markers to your selected data view, one associated with the x-axis and one with the y-axis, and position them so that they intersect at the coordinate that you want to highlight.

The following code snippet demonstrates how to create a crosshair.

```
ChartDataView dataView = chart.getDataView(0);
JCMarker xMarker = new JCMarker("X Marker", 2.0);
xMarker.setAssociatedWithYAxis(false);
dataView.addMarker(xMarker);

JCMarker yMarker = new JCMarker("Y Marker", 175.0);
yMarker.setAssociatedWithYAxis(true);
dataView.addMarker(yMarker);
```

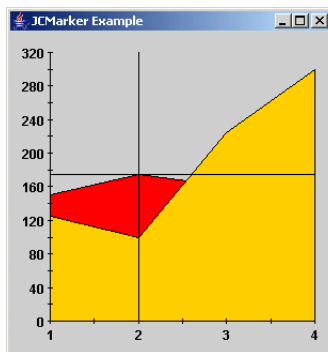


Figure 72 Area chart with markers that create a crosshair.

## 8.1.4 Customizing Markers

You can set the start and end points for the marker and customize the line style. You can also choose when the markers are drawn on the chart, that is, before or after the data is displayed. If you like, you can display marker labels in the legend or add chart labels to your markers.

### 8.1.4.1 Setting the Start and End Points

By default, a marker line spans the entire plot area. You can choose to specify the start point and end point of the line in terms of the non-associated axis. For example, the start and end point values for a x-axis marker refer to values on the y-axis. If the start point is undefined, the marker is drawn from the minimum value on the y-axis. Conversely, if the end point is not set, the marker line ends at the maximum value on the y-axis.

**Note:** For radar charts, markers associated with the y-axis can start and end only at spoke boundaries.

In the following code snippet, the `StartPoint` and `EndPoint` properties are used to modify the markers so that they do not span the entire plot area. Note that the `StartPoint` property for 'Marker 2' is not specified, so the line starts from the minimum value on the y-axis.

```
ChartDataView dataView = chart.getDataView(0);
JCMarker marker1 = new JCMarker("Marker 1", 2.0);
marker1.setAssociatedWithYAxis(false);
marker1.setStartPoint(175.0);
marker1.setEndPoint(275.0);
dataView.addMarker(marker1);

JCMarker marker2 = new JCMarker("Marker 2", 3.0);
marker2.setAssociatedWithYAxis(false);
marker2.setEndPoint(160.0);
dataView.addMarker(marker2);
```



Figure 73 Stacking bar chart and area radar chart displaying two markers.

In a circular chart where the y-axis marker is drawn as a circle, setting a start and end point results in an arc that spans the distance between the two points. The following code snippet demonstrates how you might specify a y-axis marker for a polar chart.

```
ChartDataView dataView = chart.getDataView(0);
JCMarker marker = new JCMarker("Marker", 200.0);
marker.setStartPoint(Math.PI / 4.0);
marker.setEndPoint(1.5 * Math.PI);
marker.setAssociatedWithYAxis(true);
dataView.addMarker(marker);
```

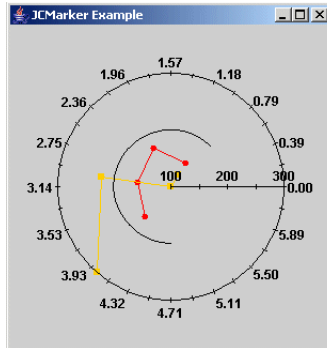


Figure 74 Polar chart displaying a y-axis marker with a start and end point (arc).

#### 8.1.4.2 Setting the Line Style

You can define the width of the line, the color, and the line attribute (solid, dashed, dotted, etcetera.).

The following code snippet creates two markers. Unique `LineStyle` properties are set for each of the markers. You may also notice that one marker is drawn before the data, and the other is drawn after the data. For more information, see Section 8.1.4.3, [Controlling When Markers are Drawn](#).

```
ChartDataView dataView = chart.getDataView(0);
JCMarker marker1 = new JCMarker("Marker 1", 100.0);
marker1.setAssociatedWithYAxis(true);
marker1.setLineStyle(new JLineStyle(3, Color.BLUE,
                                   JLineStyle.SHORT_DASH));
dataView.addMarker(marker1);

JCMarker marker2 = new JCMarker("Marker 2", 400.0);
marker2.setAssociatedWithYAxis(true);
marker2.setLineStyle(new JLineStyle(3, Color.DARK_GRAY,
                                   JLineStyle.DASH_DOT));
marker2.setDrawnBeforeData(true);
dataView.addMarker(marker2);
```

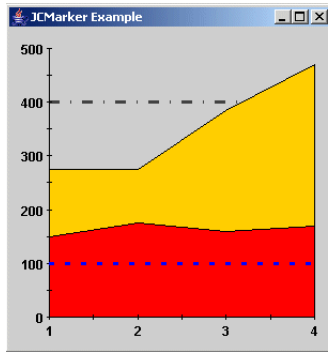


Figure 75 Stacking area chart showing two markers with different line styles.

### 8.1.4.3 Controlling When Markers are Drawn

You can choose to draw markers before or after the data is added to the chart. By default, the markers are drawn after the data and axes, but before chart labels are added. You can change the order so that the markers are drawn before the axes and data are displayed, but after the background, thresholds, and grid lines are added.

In the following code snippet, the `DrawnBeforeData` property is set to `true` so that the markers are drawn before the data.

```
ChartDataView dataView = chart.getDataView(0);
JCMarker xMarker = new JCMarker("X Marker", 2.0);
xMarker.setAssociatedWithYAxis(false);
xMarker.setDrawnBeforeData(true);
dataView.addMarker(xMarker);

JCMarker yMarker = new JCMarker("Y Marker", 175.0);
yMarker.setAssociatedWithYAxis(true);
yMarker.setDrawnBeforeData(true);
dataView.addMarker(yMarker);
```

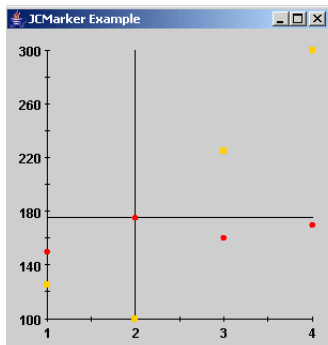


Figure 76 Plot chart with crosshair markers drawn before the data, so that the data point is displayed on top.

For another example of using the `DrawnBeforeData` property, see Section 8.1.4.2, [Setting the Line Style](#).

#### 8.1.4.4 Identifying Markers in the Legend

If you want, you can add labels for your markers to the legend. Marker labels are displayed below series labels but before threshold labels (if any).

In the following code snippet, the marker labels are added to the legend by setting the `VisibleInLegend` property to `true`. The programmer also needed to set different `LineStyle`s on the markers to be able to distinguish the markers from each other in the legend. For more information, see Section 8.1.4.2, [Setting the Line Style](#).

```
ChartDataView dataView = chart.getDataView(0);
chart.getLegend().setVisible(true);
JCMarker marker1 = new JCMarker("Marker 1", 2.0);
marker1.setAssociatedWithYAxis(false);
marker1.setLineStyle(new JLineStyle(1, Color.black,
                                   JLineStyle.SHORT_DASH));
marker1.setVisibleInLegend(true);
dataView.addMarker(marker1);

JCMarker marker2 = new JCMarker("Marker 2", 200.0);
marker2.setAssociatedWithYAxis(true);
marker2.setLineStyle(new JLineStyle(1, Color.black,
                                   JLineStyle.DASH_DOT));
marker2.setVisibleInLegend(true);
dataView.addMarker(marker2);
```

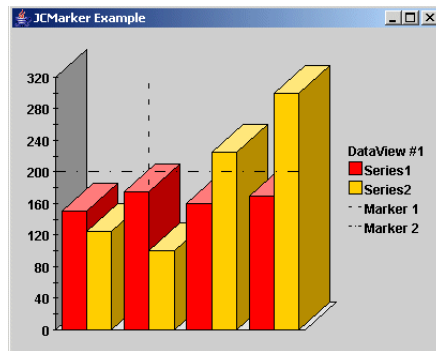


Figure 77 3D bar chart with crosshair markers identified in the legend.

#### 8.1.4.5 Attaching Chart Labels

To attach a chart label to a marker, you create the chart label and then set the `ChartLabel` property on the marker. You can set most chart label properties as usual, though there are some restrictions (see Restrictions below). By default, the chart label is attached to the marker midway between the start point and the end point. You can change where the

marker is attached by setting the `DataCoord` property of the chart label. For more information, see [Chart Labels](#), in Chapter 7, and `JCChartLabel` in [Appendix A.13](#) and in the [API Documentation](#).

*Restrictions:*

- The `JCChartLabel` component must be a `JLabel` (default), or an exception is thrown.
- The attach type is always `JCChartLabel.ATTACH_DATACoord`; if this is not the case, it is forced upon the chart label by the marker.
- When setting the attach point, the coordinate value that represents the associated axis for the marker must be on the marker. If this is not the case, it is forced upon the chart label. For example, for an x-axis marker, the x-value of the attach point will be on the marker line, even if you set the x-value to something else.
- The data view for the chart label must be the same as the marker; if this is not the case, it is forced upon the chart label.
- A chart label cannot be a dwell label, or an exception is thrown.
- A chart label must not have been added to the chart's label manager, or an exception is thrown.

The following code snippet sets a chart label on an x-axis marker and sets some properties. The chart label is attached to the marker at the default midway point.

```
ChartDataView dataView = chart.getDataView(0);
JCMarker marker = new JCMarker("My marker", 2.0);
marker.setAssociatedWithYAxis(false);
JCChartLabel label = new JCChartLabel("A Marker Label");
label.setOffset(new Point(60, -60));
label.setConnected(true);
label.getComponent().setBorder(new LineBorder(Color.black, 2));
marker.setChartLabel(label);
dataView.addMarker(marker);
```

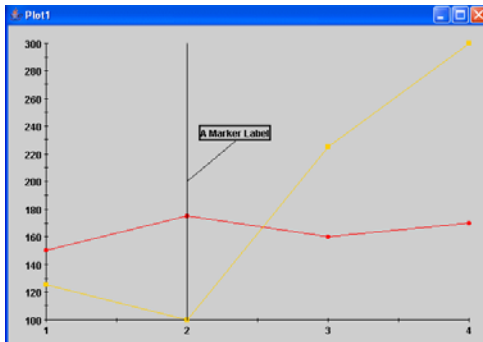


Figure 78 Plot chart showing a marker with a chart label that is attached at the midway point.



#### 8.1.4.6 Troubleshooting Missing Markers

If a marker is not drawn on your chart, it may be that its value falls outside the minimum or maximum data bounds for its associated axis. For example, if an x-axis marker value is 19, and the maximum value displayed on the chart for that axis is 10, the marker is not drawn.

By default, the data bounds for an axis are calculated based on the range of values for that axis contained in the data set; marker values are ignored. To include a marker value in the data bounds calculation, set the marker's `IncludedInDataBounds` property to `true`.

**Note:** This property is ignored for axes that have fixed bounds. Axes that have fixed bounds are the x-axis for polar and radar charts and the y-axis for 100% stacking charts. In addition, the y-axis in polar, radar, and area radar charts have a fixed minimum bound of zero (when not reversed).

## 8.2 Thresholds

Thresholds enable you to specify regions of different colors in the plot area of the chart. For example, you can create a red zone to indicate that the data values located within the zone are problematic in some specified way. You can use multiple thresholds in a single chart, and the thresholds can overlap.

**Note:** Thresholds are not available for pie charts.

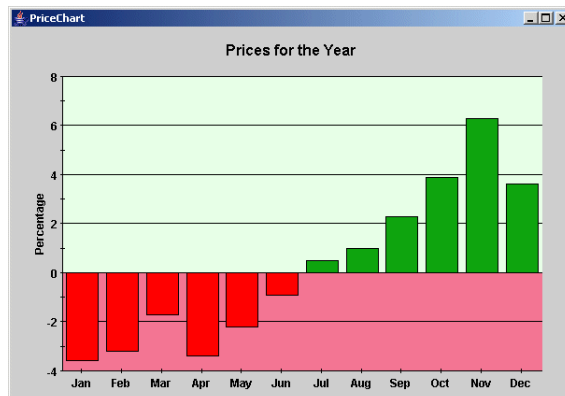


Figure 79 Bar chart that uses thresholds to highlight positive (green) and negative (red) price fluctuations.

You can have a list of thresholds for each data view. The `ChartDataView` object maintains this list and has methods to add thresholds, remove thresholds, and get the current list of thresholds. For more information, see `ChartDataView` in the [JClass API Documentation](#).

After you create a `JCThreshold` object, you associate the threshold with a data view and an axis. Thresholds are drawn immediately after the plot area is drawn, and they are drawn in the order in which they appear in the threshold list for the data view. The following sections describe how to create a threshold on the x-axis or the y-axis, and what happens when thresholds overlap or intersect.

## 8.2.1 Creating X-axis Thresholds

To add an x-axis threshold to the chart, you need to create a `JCThreshold` object with its `AssociatedWithYAxis` property set to `false` and then add it to a data view. You can define the width of the threshold by specifying a start value and an end value on the x-axis. You can also customize the fill style and add boundary lines. For more information, see Section 8.2.4, [Customizing Thresholds](#).

The following code snippet demonstrates how to create an x-axis threshold. The `JCThreshold` constructor shown in this example takes the following parameters: a label (`String`), the value at which to start the threshold (`double`), the value at which to end the threshold (`double`), the boolean `isAssociatedWithYAxis` (which is set to `false` for an x-axis threshold), and a color (in the form specified by `java.awt.Color`). For more information, see `JCThreshold` in [Appendix A.28](#) and in the [JClass API Documentation](#).

```
ChartDataView dataView = chart.getDataView(0);
boolean isAssociatedWithYAxis = false;
JCThreshold threshold = new JCThreshold("My Threshold", 2.0, 3.0,
                                     isAssociatedWithYAxis, Color.blue);
dataView.addThreshold(threshold);
```

### X-axis Threshold in a Rectangular Chart

In a rectangular chart (default orientation), associating a threshold with the x-axis creates a vertical threshold. By default, the vertical threshold spans the height of the plot area.

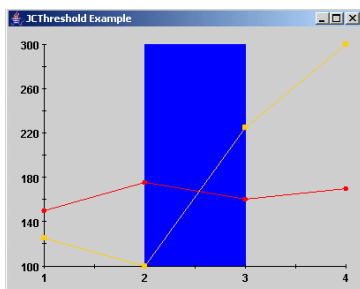


Figure 80 Plot chart (non-inverted) with an x-axis (vertical) threshold.

If the chart orientation is inverted so that the x-axis is the vertical axis, the threshold is drawn horizontally.

## X-axis Threshold in a Circular Chart

In a circular chart, thresholds associated with the x-axis are drawn as colored slices of a pie in the circular plot area.

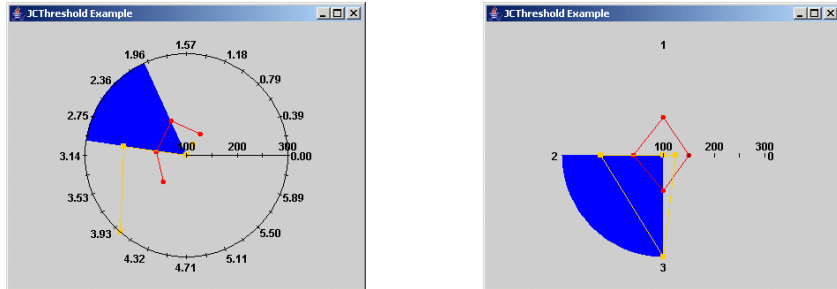


Figure 81 Polar chart and radar chart displaying pie-shaped x-axis thresholds.

## X-axis Threshold in a Webbed Chart

In radar charts, if webbed gridlines are drawn, then an x-axis threshold is displayed with straight outer edges rather than the circular arcs. For more information on webbed gridlines, see [Gridlines](#), in Chapter 6.

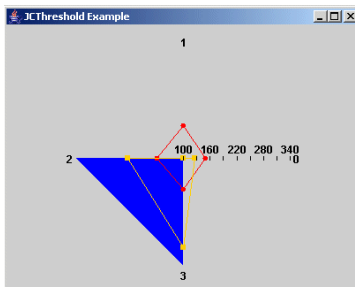


Figure 82 Radar chart displaying an x-axis threshold drawn in a web-like fashion.

### 8.2.2 Creating Y-axis Thresholds

To add a y-axis threshold to the chart, you need to create a `JCThreshold` object with its `AssociatedWithYAxis` property set to `true` and then add it to a data view. You can define the area of the threshold by specifying a start value and an end value on the y-axis, customize the fill style, and add boundary lines. For more information, see Section 8.2.4, [Customizing Thresholds](#).

The following code snippet demonstrates how to create a y-axis threshold. The `JCThreshold` constructor shown in this example takes the following parameters: a label (`String`), the value at which to start the threshold (`double`), the value at which to end the

threshold (double), the boolean `isAssociatedWithYAxis` (which is set to true for a y-axis threshold), and a color (in the form specified by `java.awt.Color`).

```
ChartDataView dataView = chart.getDataView(0);
boolean isAssociatedWithYAxis = true;
JCThreshold threshold = new JCThreshold("My Threshold", 150.0, 250.0,
                                     isAssociatedWithYAxis, Color.blue);
dataView.addThreshold(threshold);
```

### Y-axis Threshold in a Rectangular Chart

In a rectangular chart (default orientation), associating a marker with the y-axis creates a horizontal marker. By default, the horizontal marker line spans the width of the plot area.

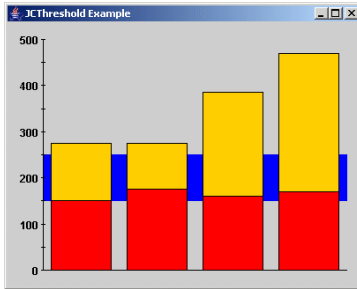


Figure 83 Stacking bar chart (non-inverted) with a y-axis (horizontal) threshold.

If the chart orientation is inverted so that the y-axis is the horizontal axis, the marker is drawn vertically.

### Y-axis Threshold in a Circular Chart

In a circular chart, thresholds associated with the y-axis are drawn as circular bands of color.

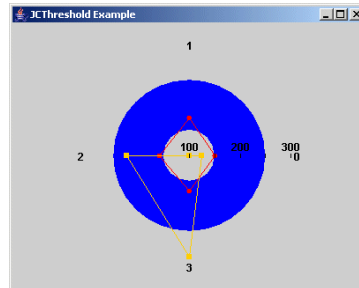
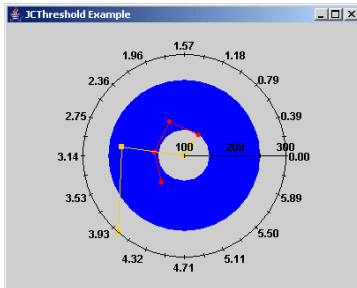


Figure 84 Polar chart and radar chart showing y-axis thresholds as circular bands of color.

## Y-axis Threshold in a Webbed Chart

In radar charts, if webbed gridlines are drawn, then a y-axis threshold is displayed as a band of color with a webbed shape rather than a circular shape. For more information on webbed gridlines, see [Gridlines](#), in Chapter 6.

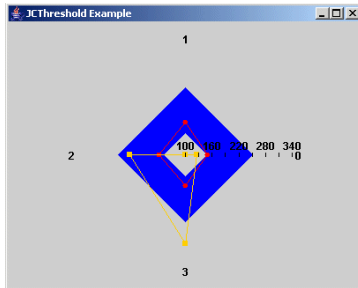


Figure 85 Radar chart displaying a y-axis threshold drawn as a web-shaped band of color.

### 8.2.3 Overlapping and Intersecting Thresholds

When you add multiple thresholds to a chart, the thresholds can be associated with either axis. If thresholds on the same axis overlap, or if you intersect thresholds by using both axes, the order in which the thresholds are drawn becomes important. Each threshold is painted on top of previously drawn thresholds. If the fill style is solid (which is the default), this means that parts or all of previously drawn thresholds may be hidden.

When the following code snippet executes, the y-axis threshold, `threshold2`, is drawn on top of the x-axis threshold because the y-axis threshold is added to the data view last.

```
ChartDataView dataView = chart.getDataView(0);
boolean isAssociatedWithYAxis = false;
JCThreshold threshold1 = new JCThreshold("My Threshold", 1.5, 2.5,
                                         isAssociatedWithYAxis, Color.green);
dataView.addThreshold(threshold1);

isAssociatedWithYAxis = true;
JCThreshold threshold2 = new JCThreshold("My Threshold", 150.0, 250.0,
                                         isAssociatedWithYAxis, Color.blue);
dataView.addThreshold(threshold2);
```

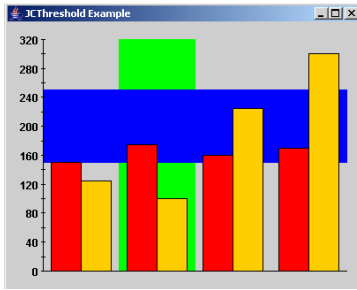


Figure 86 Bar chart with intersecting thresholds.

## 8.2.4 Customizing Thresholds

You can specify the fill style for a threshold and add boundary lines to the edges of the threshold. If you like, you can add threshold labels to the legend.

### 8.2.4.1 Setting the Fill Style for Thresholds

You can specify a fill color, fill pattern, or an image to fill the threshold region. If the `FillStyle` property of the threshold is set to null, the threshold is not filled; however, if the start and end lines are specified, the lines are still drawn. For more information, see `JCFillStyle` in [Appendix A.16](#) and in the [JClass API Documentation](#).

The following code snippet creates two x-axis thresholds with different fill styles. The fill styles are specified using the `FillStyle` property and the `JCFillStyle` object.

```
ChartDataView dataView = chart.getDataView(0);
boolean isAssociatedWithYAxis = false;
JCThreshold threshold1 = new JCThreshold("Threshold 1", 1.0, 2.5,
                                         isAssociatedWithYAxis);
threshold1.setFillStyle(new JCFillStyle(Color.blue,
                                         JCFillStyle.STRIPE_135));
dataView.addThreshold(threshold1);

JCThreshold threshold2 = new JCThreshold("Threshold 2", 2.5, 4.0,
                                         isAssociatedWithYAxis);
threshold2.setFillStyle(new JCFillStyle(Color.red,
                                         JCFillStyle.STRIPE_45));
dataView.addThreshold(threshold2);
```

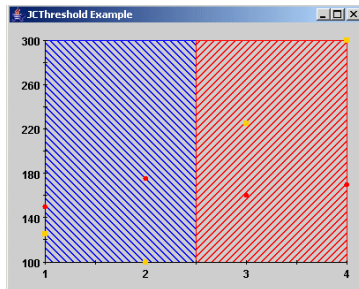


Figure 87 Scatter plot chart showing two thresholds with different fill styles.

### 8.2.4.2 Adding Boundary Lines

You can add boundary lines at the start and end of the threshold area by specifying the `StartLineStyle` and/or the `EndLineStyle` properties of the threshold. By default these properties are set to null, which means no lines are drawn. When you set one of these properties, you construct a `JLineStyle` object to specify the line style. For more information, see `JLineStyle` in [Appendix A.22](#) and in the [JClass API Documentation](#).

The following code snippet demonstrates how to add bounding lines to a threshold using the `StartLineStyle` and `EndLineStyle` properties with `JLineStyle` objects.

```
ChartDataView dataView = chart.getDataView(0);
boolean isAssociatedWithYAxis = true;
JCThreshold threshold = new JCThreshold("Threshold", 300.0, 400.0,
                                         isAssociatedWithYAxis, Color.blue);
threshold.setStartLineStyle(new JLineStyle(2, Color.black,
                                           JLineStyle.DASH_DOT));
threshold.setEndLineStyle(new JLineStyle(2, Color.black,
                                           JLineStyle.DASH_DOT));
dataView.addThreshold(threshold);
```

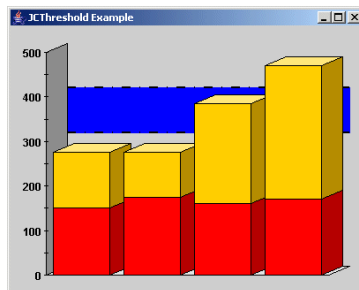


Figure 88 3D stacking bar chart showing a threshold with boundary lines.

### 8.2.4.3 Identifying Thresholds in the Legend

If you want, you can add labels for your thresholds to the legend. Threshold labels are displayed below both series labels and marker labels (if any).

In the following code snippet, the threshold labels are added to the legend by setting the `VisibleInLegend` property to `true`. Note that for clarity, the fill color chosen for the threshold should be different than the colors used for the series. For more information, see Section 8.2.4.1, [Setting the Fill Style for Thresholds](#).

```
ChartDataView dataView = chart.getDataView(0);
chart.getLegend().setVisible(true);
boolean isAssociatedWithYAxis = true;
JCThreshold threshold = new JCThreshold("The Blue Zone", 200.0, 300.0,
                                       isAssociatedWithYAxis, Color.blue);
threshold.setStartLineStyle(new JLineStyle(1, Color.black,
                                           JLineStyle.SOLID));
threshold.setEndLineStyle(new JLineStyle(1, Color.black,
                                           JLineStyle.SOLID));
threshold.setVisibleInLegend(true);
dataView.addThreshold(threshold);
```

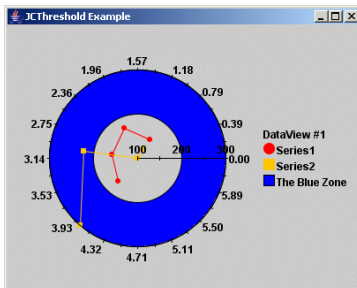


Figure 89 Polar chart with a circular y-axis threshold that is identified in the legend.

### 8.2.4.4 Troubleshooting Missing or Partial Thresholds

If your threshold is missing or is only partially visible on the chart, it may be that its start value and/or end value falls outside the minimum or maximum data bounds for its associated axis. A threshold is not drawn on the chart when its start and end values fall below the minimum data bound or exceed the maximum data bounds for its associated axis. A threshold is only partially visible if its start value falls below the minimum bound while its end value is within data bounds. Similarly, if the end value exceeds the maximum bound while the start value is within bounds, only the start of the threshold is visible.

By default, the data bounds for an axis are calculated based on the range of values for that axis contained in the data set; threshold values are ignored. To include a threshold's start and end values in the data bounds calculations, set the threshold's `IncludedInDataBounds` property to `true`.



**Note:** This property is ignored for axes that have fixed bounds. Axes that have fixed bounds are the x -axis for polar and radar charts and the y-axis for 100% stacking charts. In addition, the y-axis in polar, radar, and area radar charts have a fixed minimum bound of zero (when not reversed).



# Advanced Chart Programming

[Outputting JClass Charts](#) ■ [Batching Chart Updates](#)  
[FastAction](#) ■ [FastUpdate](#) ■ [Programming End-User Interaction](#)  
[Map and Unmap](#) ■ [Pick](#) ■ [Unpick](#) ■ [Using Pick and Unpick](#)  
[Coordinate Conversion Methods](#) ■ [Image-Filled Bar Charts](#)

Controlling the chart in an application program is generally straightforward once you are familiar with the programming basics and the object hierarchy. For most JClass Chart objects, all the information needed to program them can be found in the [API](#). In addition, extensive information on how they can be used can be found in the numerous example and demonstration programs provided with JClass Chart.

This chapter covers more advanced programming concepts for JClass Chart and also looks at more complex chart programming tasks.

## 9.1 Outputting JClass Charts

Many applications require that the user has a way to get an image or a hard copy of a chart. JClass Chart allows you to output your chart as a GIF, PNG, or JPEG image, to either a file or an output stream. If you have JClass PageLayout installed, you can also choose to encode your charts as an EPS, PS, or PDF file. For more information, please see the [JClass PageLayout Programmer's Guide](#).

Located in `com.klg.jclass.util.swing.encode`, the [JCEncodeComponent](#) class is used to encode components into different image file formats. When you include this class in your program, you can call one of two methods that allow you to save the chart image as a GIF, PNG, or JPEG file, sending it to either a file or an output stream.

The parameters of the two methods are the same, except for output.

### 9.1.1 Encode method

The method to output to a file is:

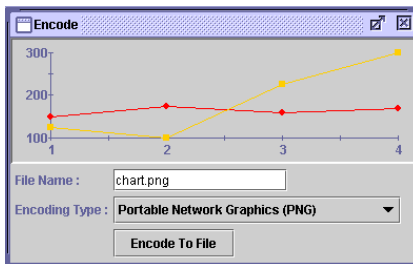
```
public static void encode(JCEncodeComponent.Encoding encoding,
    Component component, File file)
```

The method to output to an output stream is the same, except that the last parameter is `OutputStream output`, that is `...Component component, OutputStream output`)

The `component` parameter refers to the component to encode, that is, the chart; the `encoding` parameter refers to the type of encoding to use (a GIF, PNG, or JPEG; or, if you have JClass PageLayout installed, EPS, PS, or PDF file); and the `output` parameter refers either to the file to which to write the encoding or to the stream to which to write the encoding.

### 9.1.2 Encode example

To see this `encode` method in action, review the Encode example, found in the *Example & Demo Gallery*. This example appears in the Advanced folder.



In this example, you can alter the encoding type by selecting a different encoding type from the drop-down menu. Another option provided is your choice of file name. Also, you can right-click the example to bring up the Property Editor and further manipulate the properties of the chart.

### 9.1.3 Code example

The following code snippet was used to create the example above.

```
public void actionPerformed(ActionEvent evt) {
    int typeIndex = encTypesCB.getSelectedIndex();
    String fileName = encFileTF.getText();

    if (evt.getSource() == encButton) {
        // if encode button pressed, get the encoding type and file name
        // and use them to encoding the chart

        if (typeIndex >= 0 && !(fileName.equals("")) ) {
            // Call chart's encoding method, but make sure to catch
            // possible exception
            try {
                JCEncodeComponent.encode (JCEncodeComponent.ENCODINGS
                                         [typeIndex], chart, new File(fileName));
            }
            catch (EncoderException ee) {
                ee.printStackTrace();
            }
        }
    }
}
```

```

        catch (IOException io) {
            io.printStackTrace();
        }
    }
}

```

## 9.2 Batching Chart Updates

Normally, the chart is repainted immediately after a property is set. To make several changes to a chart before causing a repaint, set the `Batched` property of the `JCChart` object to `true`. Property changes do not cause a repaint until `Batched` is reset to `false`.

The `Batched` property is also defined for the `ChartDataView` object. This `Batched` property is independent of `JCChart.Batched`. It is used to control the update requests sent from the `DataSource` to the chart.

**Note:** It is **highly recommended** that you batch around the creation or updating of multiple chart labels.

## 9.3 FastAction

The `FastAction` property determines whether chart actions will use an optimized mode in which it does not bother to update display axis annotations or gridlines during a chart action. Default value is `false`.

Using `FastAction` can greatly improve the performance of a chart display, because relatively more time is needed to draw such things as axis annotations or gridlines than for simply updating the points on a chart. It is designed for use in dynamic chart displays, such as charts that enable the user to perform translation or rotation actions.

The following line of code shows how `FastAction` can be used in a program:

```
c.getChartArea().setFastAction(true);
```

## 9.4 FastUpdate

The `FastUpdate` property optimizes chart drawing – if possible, only new data that has been added to the `datasource` is drawn when the chart updates, with little recalculating and redrawing of existing points. (Please see [Making Your Own Chart Data Source](#), in Chapter 4, for a guide on how to build an updating chart data source.) However, if the new data goes outside of the current axis boundaries, then a full redraw is done.

Using `FastUpdate` can improve the performance of a chart display, especially with dynamic chart displays.

The following line of code shows how `FastUpdate` can be used in a program:

```
c.getDataView(0).setFastUpdate(true);
```

A chart using the fast update feature will not draw correctly when the chart object is placed within an `JInternalFrame` object or when items from a `JPopupMenu` overlay the chart.

Please see the `FastUpdate` demo, found in *JCLASS\_HOME/demos/chart/fastupd/*, for a demonstration of this feature.

**Note:** This feature is not supported in Area Radar or Radar charts. For Polar charts, there is no need to check the axis bounds in the x-direction. The routines for checking axis bounds can still be used for the y-direction.

## 9.5 Programming End-User Interaction

An end-user can interact with a chart more directly than using the Customizer. Using the mouse and keyboard, a user can examine data more closely or visually isolate part of the chart. `JClass Chart` provides the following interactions:

- moving the chart
- zooming into or out of the chart
- rotation (only for bar or pie charts displaying a 3D effect)
- adding depth cues to the chart
- interactively change data points (using the pick feature)

It is also possible in most cases for the user to reset the chart to its original display parameters. The interactions described here affect the chart displayed inside the `ChartArea`; other chart elements, such as the header, are not affected.

**Note:** The keyboard/mouse combinations that perform the different interactions can be changed or removed by a programmer. The interactions described here may not be enabled for your chart.

A chart action is a user event that causes some interactive action to take place in the control. In `JClass Chart`, actions like zoom, translate and rotate can be mapped to a mouse button and a modifier. For example, it is possible to bind the translate event to the combination of mouse button 2 and the **Control** key. Whenever the user hits **Control** and mouse button 2 and drags the mouse, the chart will move.

### 9.5.1 Event Triggers

An event trigger is a mapping of a mouse operation and/or a key press to a chart action. In the example above, the trigger for translate is a combination of mouse button 2 and the **Control** key.

An event trigger has two parts:

- the modifier, which specifies the combination of meta keys and mouse buttons that will trigger the action
- the action, which specifies the combination of chart action that will occur

Valid actions include `EventTrigger.CUSTOMIZE`, `EventTrigger.DEPTH`, `EventTrigger.EDIT`, `EventTrigger.PICK`, `EventTrigger.PICK_SERIES`, `EventTrigger.ROTATE`, `EventTrigger.TRANSLATE`, and `EventTrigger.ZOOM`.

### 9.5.2 Valid Modifiers

The value of a modifier is specified using *java.AWT.event* modifiers, as shown in the following list:

- `InputEvent.SHIFT_MASK`
- `InputEvent.CTRL_MASK`
- `InputEvent.ALT_MASK`
- `InputEvent.META_MASK`

You can also specify the mouse button using one of the following modifiers:

- `InputEvent.BUTTON1_MASK`
- `InputEvent.BUTTON2_MASK`
- `InputEvent.BUTTON3_MASK`

### 9.5.3 Programming Event Triggers

To program an event trigger, use the `setTrigger` method to add the new action mapping to the collection.

For example, the following tells JClass Chart to add a zoom operation as its first trigger (first trigger denoted by 0) when **Shift** and mouse button are pressed:

```
c.setTrigger(0,newEventTrigger(Event.SHIFT_MASK,
    EventTrigger.ZOOM);
```

### 9.5.4 Removing Action Mappings

To remove an existing action mapping, set the trigger to null, as in the following example:

```
c.setTrigger(0,null);
```

### 9.5.5 Calling an Action Directly

In JClass Chart, it is possible to force some actions by calling a method of `JCCChart`. The following is a list of the methods that can be called upon to force a particular action:

- Translation – `translateStart()`, `translate()`, `translateEnd()`
- Rotation – `rotateStart()`, `rotate()`, `rotateEnd()`

- **Zoom** – `zoomStart()`, `zoom()`, `zoomEnd()`
- **Scale** – `scale()`
- **Reset** – `reset()`

### 9.5.6 Specifying Action Axes

Actions like translation occur with respect to one or more axes. In JClass Chart, the axes can be set using the `HorizActionAxis` and `VertActionAxis` properties of `JCChartArea`, as the following code fragment illustrates:

```
ChartDataView arr = c.getDataView(0);
c.getChartArea().setHorizActionAxis(arr.getXAxis());
c.getChartArea().setVertActionAxis(arr.getYAxis());
```

Note that it is possible to have a null value for an action axis. This means that chart actions like translation do not have any effect in that direction. By default, the `HorizActionAxis` is set to the default x-axis, and the `VertActionAxis` is set to the default y-axis.

## 9.6 Map and Unmap

The `map` and `unmap` are functionally equivalent to the `coordToDataCoord()` and `dataIndexToCoord()` methods. They are provided as convenience methods, and are more in keeping with typical Java terminology than `coordToDataCoord()` and `dataIndexToCoord()`. For more information, see Section 9.10, [Coordinate Conversion Methods](#).

For polar charts, the x- and y-values are interpreted as (*theta*, *r*) coordinates. The X units used will depend on the current value of angle unit. The case for radar and area radar charts is similar, except that x-values will be ignored.

## 9.7 Pick

There are two `pick` methods: `pick()` and `pickSeries()`.

The `pick()` method is used to translate a pixel coordinate on a chart to the data point that is closest to it. The method takes a `Point` object containing a pixel coordinate and an optional `ChartDataView` object to check against, and returns the resulting data point encapsulated in a `JCDataIndex` object.

The `pickSeries()` method is used to translate a pixel coordinate on a chart to the data series line that is closest to it. The method takes a `Point` object containing a pixel coordinate and an optional `ChartDataView` object to check against, and returns the resulting data series and the nearest point on the series line encapsulated in a `JCDataIndex` object. The `pickSeries()` method is suitable for plot, polar, and radar



charts, that is, chart types where the points in a series are plotted and connected by a line. Calling the `pickSeries()` method on other chart types returns the same result as calling the `pick()` method.

For the pick methods to work correctly, the `JCChart` instance must first be laid out. This is automatically done whenever a chart is drawn, such as when the `snapshot()` method is called. Alternately, layout can be accomplished manually by calling the `doLayout()` method of `JCChart`.

### 9.7.1 Pick Methods for Polar and Radar Charts

The `pick()` method for polar and radar charts is implemented in two stages. The data point closest to the pick point is identified in a primary search, thus obeying the specified pick focus rule. In some cases (for example, radar charts with more than one series), there may be two or more data points that have the same x- or y-value. The primary search result may be ambiguous if the pick focus rule is `PICK_FOCUS_X` or `PICK_FOCUS_Y`. To determine which of those points is the desired one, a secondary search is carried out using the `PICK_FOCUS_XY` rule.

For `pickSeries()`, `PICK_FOCUS_XY` is always used.

### 9.7.2 Pick Methods for Area Radar Charts

The pick behavior for area radar charts differs from that of polar or radar charts. If the user clicks on a point within a filled polygon, the search for the closest point (again, obeying the pick focus rule) is limited to the data series represented by that polygon. Pick points within a polygon have the `JCDataIndex.distance` variable set to 0. If the pick point is not within a filled polygon (that is, the user clicked on a point outside of the largest polygon), then the smallest distance from the pick point to the polygon is taken. As with the polar and radar chart types, primary and secondary searches are conducted to resolve ambiguities that may arise for `PICK_FOCUS_X` or `PICK_FOCUS_Y`.

### 9.7.3 Pick Focus

`pick` normally takes an *x,y* coordinate value, but it can take an x- or y-value only, which is useful for specific chart types. This can be specified using the `PickFocus` property of `ChartDataView` which specifies how distance is determined for pick operations. When set to `PICK_FOCUS_XY` (default), a pick operation will use the actual distance between the point and the drawn data. When set to values of `PICK_FOCUS_X` or `PICK_FOCUS_Y`, only the distance along the x-axis or the y-axis is used.

This is particularly useful when using bar charts. In most cases it is desirable to translate a pixel coordinate into the bar directly below the coordinate rather than the nearest bar, which may be to one side. To accomplish this, merely set the `PickFocus` property to carry out only operations against the *y* coordinate as follows:

```
arr.setPickFocus(ChartDataView.PICK_FOCUS_Y);
```

## 9.8 Unpick

The `unpick()` method essentially functions in the opposite manner of `pick`: given a data series and a data point within that series, `unpick` returns the pixel co-ordinates of that point relative to the chart area. It takes two sets of parameters: `pt` for the point index, and `series` for the data series. For bar charts it returns the top-middle location for a given bar, and the middle of an arc for a pie chart. `unpick` can be used to display information at a given point in a chart, and can be used for attaching labels to chart regions.

For `unpick()` to work correctly, the `JCChart` instance must first be laid out. This is automatically done whenever a chart is drawn, such as when the `snapshot()` method is called. Alternately, layout can be accomplished manually by calling the `doLayout()` method of `JCChart`.

## 9.9 Using Pick and Unpick

The `pick` method is used to retrieve an *x,y* coordinate in a chart from user input and then translate that into selecting the data point nearest to it. For example, if a user clicks within a single bar within a bar chart, `pick` takes the coordinates of the mouse-click and selects that bar for any action within the program. Similarly, if a user clicks in an area immediately above a bar chart, `pick` is used to select the bar that is closest to the mouse click.

To use the `pick` or `pick series` listener, you must first set up a `PICK` or `PICK_SERIES` event trigger on the chart. For more information, see Section 9.7, [Pick](#) and Section 9.5.3, [Programming Event Triggers](#).

Consider the following code (taken from the *DrillDown* demo in *JCLASS\_HOME/demos/chart/drilldown/*) that demonstrates how `pick` can be used to “drill down” to reveal more information.

**Note:** This example assumes that a `data.class` file exists that understands different data levels.

```
package demos.chart.drilldown;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Event;
import java.awt.GridLayout;
import javax.swing.JComponent;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JEditorPane;
import javax.swing.BorderFactory;

import java.util.Iterator;
import java.util.List;
```

```

import com.klg.jclass.chart.ChartDataView;
import com.klg.jclass.chart.ChartDataViewSeries;
import com.klg.jclass.chart.EventTrigger;
import com.klg.jclass.chart.JCAxis;
import com.klg.jclass.chart.JCPickListener;
import com.klg.jclass.chart.JCPickEvent;
import com.klg.jclass.chart.JCChartStyle;
import com.klg.jclass.chart.JCLineStyle;
import com.klg.jclass.chart.JCChart;
import com.klg.jclass.chart.ChartText;
import com.klg.jclass.chart.JCDataIndex;
import com.klg.jclass.chart.JCChartArea;
import com.klg.jclass.util.swing.JCExitFrame;
import com.klg.jclass.util.legend.JCLegend;

/*
 * This demo demonstrates using pick to drill down to more
 * refined data
 */
public class DrillDown extends javax.swing.JPanel
    implements JCPickListener {

    protected Data    d = null;
    protected JCChart c = null;

    public DrillDown()
    {
        setLayout(new BorderLayout(10,10));
        setPreferredSize(new Dimension(600,400));
        d = new Data();

        Color Turquoise = new Color(64,224,208);
        Color DarkTurquoise = new Color(0x00,0xce,0xd1);

        c = new JCChart();
        c.setTrigger(0, new EventTrigger(0, EventTrigger.PICK));
        c.setBackground(DarkTurquoise);

        JCChartArea area = c.getChartArea();
        area.getPlotArea().setBackground(Turquoise);
        area.setOpaque(true);
        area.setBorder(BorderFactory.createEtchedBorder());

        JComponent header = c.getHeader();
        header.setBackground(Turquoise);

        ((JLabel)header).setText("<html><center>
                                <font color=black><b>Drill Down
                                Demo</b><P>Independent Comic Book
                                Sales 1996</center>");

        header.setBorder(BorderFactory.createRaisedBevelBorder());
        header.setVisible(true);

        JCLegend legend = c.getLegend();
        legend.setVisible(true);

```

```

legend.setBackground(Turquoise);
legend.setForeground(Color.black);
legend.setBorder(BorderFactory.createLoweredBevelBorder());

ChartDataView dataView = c.getDataView(0);
c.setBatched(false);
dataView.setDataSource(d);
dataView.setChartType(JCChart.BAR);
dataView.setHoleValue(-1000);
dataView.getOutlineStyle().setColor(Color.darkGray);

JComponent footer = c.getFooter();
footer.setVisible(true);

((JLabel)footer).setText("<html><font color=black size=-1>
                                <CENTER><i>Drill Down -> Mouse Down
                                on Bar or Legend<P>Drill Up ->
                                Mouse Down on Other Area of
                                Graph</i></CENTER>");

area.setDepth(10);
area.setElevation(20);
area.setRotation(20);

JCAxis yAxis = area.getYAxis(0);
yAxis.setGridVisible(true);
yAxis.getGridStyle().getLineStyle().setColor(new
    Color(154,154,229));

// Set colors for each data series
setSeriesColor();

// Set up pick and rotate trigger
c.setTrigger(0, new EventTrigger(0, EventTrigger.PICK));
c.setTrigger(1, new EventTrigger(Event.SHIFT_MASK,
                                EventTrigger.ROTATE));
c.setTrigger(2, new EventTrigger(Event.META_MASK,
                                EventTrigger.CUSTOMIZE));
c.setAllowUserChanges(true);

// Add listener for pick events
c.addPickListener(this);

add("Center",c);
}

void setSeriesColor()
{
    // Set colors for each data series
    Color colors[] = {Color.red, Color.blue, Color.white,
                    Color.magenta, Color.green, Color.cyan,
                    Color.orange, Color.yellow};
    ChartDataView dataView = c.getDataView(0);
    List seriesList = dataView.getSeries();

```

```

        Iterator iter = seriesList.iterator();
        for (int i = 0; iter.hasNext(); i++) {
            ChartDataViewSeries series =
                (ChartDataViewSeries)iter.next();
            series.getStyle().setFillColor(colors[i]);
        }
    }

    /**
     * Pick event listener. Upon receipt of a pick event, it either
     * drills up or down to more general or refined data.
     */
    public void pick(JCPickEvent e)
    {
        boolean doLevel = false;
        boolean doUpLevel = true;
        JCDataIndex di = e.getPickResult();
        int srs = 0;

        // If clicked on bar or legend item, drill down. If clicked on
        // any other area of chart, drill up.
        if (di != null) {
            Object obj = di.getObject();
            ChartDataView vw = di.getDataView();
            srs = di.getSeriesIndex();
            int pt = di.getPoint();
            int dist = di.getDistance();

            if (vw != null && srs != -1) {
                if (srs >= 0) {
                    if ((obj instanceof JCLegend) ||
                        (obj instanceof JCChartArea && dist == 0))
                    {
                        doLevel = true;
                        doUpLevel = false;
                    }
                    else {
                        doLevel = true;
                    }
                }
            }
            else {
                doLevel = true;
            }
        }
        else {
            doLevel = true;
        }

        if (doLevel) {
            c.setBatched(true);
            if (doUpLevel) {
                d.upLevel();
            }
        }
    }

```

```

        else {
            d.downLevel(srs);
        }
        setSeriesColor();
        c.setBatched(false);
    }
}

public static void main(String args[])
{
    JCExitFrame f = new JCExitFrame("Basic Drilldown example");
    DrillDown tc = new DrillDown();
    f.getContentPane().add(tc);
    f.pack();
    f.setVisible(true);
}
}

```

When compiled and run, the *DrillDown.class* program displays the following:

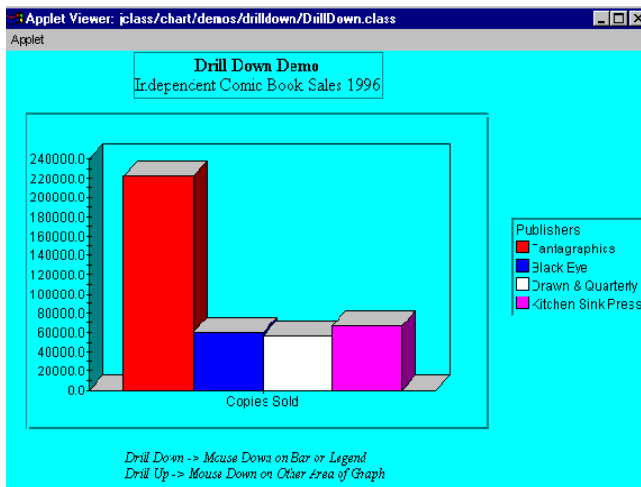


Figure 90 The DrillDown demonstration program displayed.

When a bar or legend within this chart is clicked by the user, the program “drills down” to reveal more refined data comprising that bar. If an area outside of the bars is clicked upon, then the program “drills up” to reveal more general data.

`pick` is key to this program, determining the way the program interacts with the user. `pick` requires an event trigger and listener to work, as the following code fragment shows:

```

c.setTrigger(0, new EventTrigger(0, EventTrigger.PICK));
c.addPickListener(this);
public void pick(JCPickEvent e)

```

```

{
    JCDataIndex di = e.getPickResult();
}

```

When a user clicks in the *DrillDown* demonstration program, the event is triggered, and the  $x,y$  coordinates are passed along to the pick event listener, which in turn takes the information and performs the indicated action. The pick() method returns a JCDataIndex, which encapsulates the point index and data series of the selected point. It is also possible to send a pick event to objects manually.

When the sendPickEvent() method is called, it sends a pick event to all objects listening for it.

## 9.10 Coordinate Conversion Methods

The ChartDataView object in the underlying data model provides methods that enable you to do the following:

- Convert from data coordinates (x- and y-data values) to pixel coordinates (where these data coordinates appear on screen) and vice versa.
- Determine the pixel coordinates of a given data point in a series, or the closest point or series to a given set of pixel coordinates.

Note that for these calls to work, the chart must first be laid out with a call to doLayout().

The following table outlines which method or functional equivalent to use for each action.

Method	Functional equivalent	Action
dataCoordToCoord()	unmap	Converts from data coordinates to pixel coordinates
coordToDataCoord()	map	Converts from pixel coordinates to data coordinates
dataIndexToCoord()	unpick	Determines the pixel coordinates of a given data point in a series
coordToDataIndex()	pick	Determines the closest point in pixels to a given data point in a series
coordToDataSeries()	pickSeries	Determines the closest point in pixels to a series line

## DataCoordToCoord

To convert from data coordinates to pixel coordinates, call the `dataCoordToCoord()` method. For example, the following code obtains the pixel coordinates corresponding to the data coordinates (5.1, 10.2):

```
Point p=c.getDataView(0).dataCoordToCoord(5.1,10.2);
```

This works in the same way as `unmap`. Note that the pixel coordinate positioning is relative to the upper left corner of the `JCChart` component display.

## CoordToDataCoord

To convert from pixel coordinates to data coordinates, call `coordToDataCoord()`. For example, the following converts the pixel coordinates (225, 92) to their equivalent data coordinates:

```
JCDataCoord cd=c.getDataView(0).coordToDataCoord(225,92);
```

This works in the same manner as `map`. So, `coordToDataCoord()` returns a `JCDataCoord` object containing the x- and y-values in the data space.

## DataIndexToCoord

To determine the pixel coordinates of a given data point, call `dataIndexToCoord()`. For example, the following code obtains the pixel coordinates of the third point in the first data series:

```
JCDataIndex di= new JCDataIndex(3,c.getDataView(0).getSeries(0));  
Point cdc=c.getDataView(0).dataIndexToCoord(di);
```

## CoordToDataIndex

To determine the closest data point to a set of pixel coordinates, call `coordToDataIndex()`:

```
JCDataIndex di=c.getDataView(0).coordToDataIndex(225,92,  
ChartDataView.PICK_FOCUSXY);
```

The last argument specifies how the nearest series and point value are determined. This argument can be one of `ChartDataView.PICK_FOCUSXY`, `ChartDataView.PICK_FOCUSX`, `PICK_FOCUSY`, or `ChartDataView.PICK_FOCUS_LOCAL`. Produces the same result as calling the `pick()` method. `JCDataIndex` contains the series and point value corresponding to the closest data point, and also returns the distance in pixels between the pixel coordinates and the point. Returns a `JCDataIndex` instance.

## CoordToDataSeries

To determine the closest series line to a set of pixel coordinates, call `coordToDataSeries()`:

```
JCDataIndex di=c.getDataView(0).coordToDataSeries(225,92,  
ChartDataView.PICK_FOCUSXY);
```



The last argument specifies how the nearest series and point value are determined. This argument can be one of `ChartDataView.PICK_FOCUSXY` or `ChartDataView.PICK_FOCUS_LOCAL`. Produces the same result as calling the `pickSeries()` method. Returns a `JCDataIndex` instance.

## 9.11 Image-Filled Bar Charts

Using the underlying data model, it is possible to use image files as chart elements within a bar chart. This is accomplished by using the `Image` in `JCFillStyle` and iterating through the series. `Image` sets the image used to paint the fill region of bar charts. It takes `img` as a parameter, which is an `AWT Image` class representing the image to be used to paint image fills. If set to null, no image fill is done.

The following code fragment shows how `Image` can be incorporated into a program:

```
String imageStrings[] = {"cd.gif", "tape.gif"};
List seriesList = arr.getSeries();
Iterator iter = seriesList.iterator();
for (int i = 0; iter.hasNext(); i++) {
    ChartDataViewSeries thisSeries = (ChartDataViewSeries) iter.next();
    if (i < seriesLabels.length) {
        if (imageStrings[i] != null) {
            Class cl = getClass();
            URL url = cl.getResource("/examples/chart/intro/"+
                                   imageStrings[i]);
            if (url != null) {
                ImageIcon icon = new ImageIcon(url);

                thisSeries.getStyle().getFillStyle().setImage(icon.getImage());
                thisSeries.getStyle().getFillStyle().setPattern(JCFillStyle.CUSTOM_STACK);
            }
        }
    }
}
```

The effects can be seen in the *ImageBar* example located in the `JCLASS_HOME/examples/chart/intro/` directory.

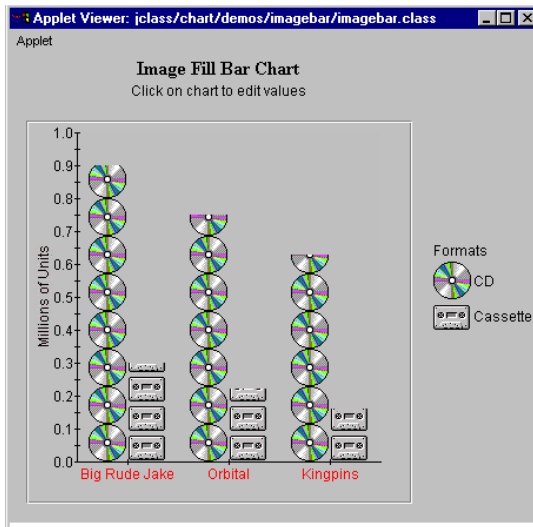


Figure 91 Demonstration of image bars.

The image is clipped at the point of the highest value indicated for the bar chart.

Image only tiles the image along a single axis. For example, if the bars were widened in the above illustration, it would still tile along the vertical y-axis only, and would not fill in the image across the horizontal x-axis. This same principle applies (though along different axes) when the bar chart is rotated 90 degrees.

**Note:** Image can only be used with the image formats that can be used in Java.

# *Part II*

## *Supported Technologies*



# Using JCChartFactory

*Overview of the JCChartFactory Class* ■ *Overview of the LoadProperties Class*  
*Saving Data: The OutputDataProperties Class* ■ *Saving Image Information: The OutputProperties Class*

You can use the `JCChartFactory` class to create and update charts using properties defined in a markup language, as well as save chart properties to a markup language. Currently, JClass Chart supports the following markup languages: HTML and XML.

This chapter describes the `JCChartFactory` and related classes.

The following chapters tell you how to use the factory with HTML and XML:

- Chapter 11, [Loading and Saving Charts Using HTML](#)
- Chapter 12, [Loading and Saving Charts Using XML](#)

## 10.1 Overview of the JCChartFactory Class

The `JCChartFactory` class is a convenience class that provides methods to create, update, and save a chart using a variety of data formats. You can define the chart properties in HTML or XML; the `JCChartFactory` methods have a type parameter where you specify `JCChartFactory.HTML` or `JCChartFactory.XML`, as appropriate.

For example, the following code snippet (taken from the example in Chapter 11, [Loading and Saving Charts Using HTML](#)) creates a chart from properties contained in a file (`inFile`). The last parameter in the method tells the factory that the properties are defined in HTML tags.

```
chart = JCChartFactory.makeChartFromFile(inFile, loadProps,
                                         chartName, JCChartFactory.HTML);
```

Similarly, for XML (taken from Chapter 12, [Loading and Saving Charts Using XML](#)):

```
chart = JCChartFactory.makeChartFromFile(inFile, loadProps,
                                         chartName, JCChartFactory.XML);
```

The `loadProps` parameter is an instance of `LoadProperties`. For more information, see Section 10.2, [Overview of the LoadProperties Class](#).

The following table summarizes the methods in `JCChartFactory` for creating, updating, and saving charts. The format-specific make and update methods, with the exception of `updateChartWithData()`, call `updateChart()` under the hood. These methods are provided for convenience. Similarly, the save methods call `saveChart()`. For a list of the method parameters, look up the `JCChartFactory` class in the *API Documentation*.

Create Methods	Update Methods	Save Methods
<code>makeChartFromFile()</code>	<code>updateChart()</code> <sup>a</sup>	<code>saveChart()</code> <sup>b</sup>
<code>makeChartFromReader()</code>	<code>updateChartFromFile()</code>	<code>saveChartToFile()</code>
<code>makeChartFromStream()</code>	<code>updateChartFromReader()</code>	<code>saveChartToStream()</code>
<code>makeChartFromString()</code>	<code>updateChartFromStream()</code>	<code>saveChartToString()</code>
	<code>updateChartFromString()</code>	
	<code>updateChartWithData()</code>	

a. You can use this method instead of one of the convenience methods. The input source can be a `String` (interpreted as a file name), `URL`, `InputStream`, or `Reader`.

b. You can use this method instead of one of the convenience methods. The output target can be a `String` (interpreted as a file name), `OutputStream`, or `Writer`.

## 10.2 Overview of the LoadProperties Class

The `LoadProperties` class is responsible for the following tasks:

- Telling the chart how to access data files and image files based on their `fileAccess` properties defined in the HTML or XML source. For more information, see Section 10.2.1, [LoadProperties Class and the fileAccess Property](#).
- Passing user-defined objects to an external Java class when `<external-java-code>` elements are defined in the XML source. For more information, see [external-java-code](#) in Appendix C, [XML DTD](#).
- Identifying what to do when there is an error in reading external data or an image from its source. Normally, the chart throws a `JCIOException` when this happens. However, you can ignore these exceptions and continue loading the chart by setting the `ignoreExternalResourceExceptions` property to `true`.

The `JCChartFactory` class automatically creates a default `LoadProperties` object if you do not define one. In many cases, the default object is sufficient. However, there are times when you will need to define a `LoadProperties` object. For more information, see Section 10.2.2, [When to Define a LoadProperties Object](#).

### 10.2.1 LoadProperties Class and the fileAccess Property

When you define a data file or an image file in HTML or XML, you specify a `fileName` property and a `fileAccess` property. The `fileAccess` property is used by the `LoadProperties` object to determine how to access the data file or image file named in the `fileName` property.

The following table summarizes the valid values for the `fileAccess` property and describes how the `LoadProperties` object interprets the `fileName` property.

fileAccess Value	Description
Default	The default access is Absolute.
Absolute	Interprets the value of <code>fileName</code> as an absolute name.
Url	Interprets the value of <code>fileName</code> as a URL.
Relative_Url	Interprets the value of <code>fileName</code> as a URL after adding a prefix to the beginning of it. You specify the prefix by setting the <code>relativeURLPrefix</code> property of the <code>LoadProperties</code> object. For more information, see Section 10.2.2, <a href="#">When to Define a LoadProperties Object</a> .
Resolving_Class	Requires a resolving class <code>Class</code> object to load the file. The <code>ClassLoader</code> of the resolving class is used to resolve the String set in the <code>fileName</code> property through a call to <code>getResource(filename)</code> . In the resolution process, if the String starts with “/”, it is unchanged; otherwise, the package name of the resolving <code>Class</code> is added to the beginning of the String, after converting “.” to “/”. You specify the resolving class by setting the <code>resolvingClass</code> property of the <code>LoadProperties</code> object. For more information, see Section 10.2.2, <a href="#">When to Define a LoadProperties Object</a> .

The same values are used by the chart when saving to HTML or XML so that, if the chart is reloaded, the chart knows how to access the data and images.

### 10.2.2 When to Define a LoadProperties Object

In simple cases, you can use the default `LoadProperties` object created by the factory or you can use the no-arguments `LoadProperties` constructor to create a `LoadProperties` object that uses null values for all its properties. In some cases, however, you need to specify some `LoadProperties` properties.

In the following circumstances, you need to set the specified `LoadProperties` property:

- When you define a data file or image file with `fileAccess=Resolving_Class`, you need to set the `resolvingClass` property of the `LoadProperties` object to specify the `Class` object that is used to resolve the location of file.
- When you define a data file or image file with `fileAccess=Relative_Url`, you need to set the `relativeURLPrefix` property of the `LoadProperties` object to specify the `String` to prepend to the URL. (Recall that in this case the `fileName` attribute of the `<image-file>` is interpreted as a URL.)
- When you specify an `<external-java-code>` tag in XML, you need to set the `userObject` property to specify the `Object` and the `storeUserObject` property to determine if the `Object` is stored to the `userObject` property of the chart.

The following example shows a `LoadProperties` constructor with some properties set.

```
Class myResolvingClass = new Class(...);
Object myObject = new Object(...);

// Create a LoadProperties object and set properties
LoadProperties loadProps = new LoadProperties(
    myResolvingClass,    // resolvingClass
    "",                 // relativeURLPrefix (default is empty String)
    myObject,           // userObject
    true);              // storeUserObject
```

Alternatively, you can set these properties using the `LoadProperties` object's `set*(())` methods. For more information, look up `com.klg.jclass.util.io.LoadProperties` in the *API documentation*.

## 10.3 Saving Data: The `OutputDataProperties` Class

The `OutputDataProperties` class is responsible for controlling whether or not the chart data is saved when the chart properties are saved, and if it is, whether the data is embedded in the chart properties file or saved to a separate file. It also specifies how the data should be read in when the HTML or XML chart properties are loaded into a chart.

The following code creates a sample `OutputDataProperties` object.

```
// Create an instance of OutputProperties
OutputProperties dataOutputProps = new OutputProperties(
    "chartdataout.xml",           // outputFileName
    "http://www.mysite.com/chartdataout.xml", // propertyName
    OutputDataProperties.DATA_FILE_XML, // saveType
    Properties.URL);              // fileAccess
```



The `saveType` determines whether or not data is saved, and if so, how it is saved. The following table summarizes the valid values for the `saveType` property:

saveType Value	Description
<code>OutputDataProperties.NO_DATA</code>	Data is not saved. This <code>ChartDataView</code> will have no data when the saved chart properties are loaded into a chart.
<code>OutputDataProperties.EMBED_DATA</code>	The data is embedded in the chart properties file. No data file is required.
<code>OutputDataProperties.DATA_FILE_TXT</code>	The data is saved to a file in text data format (see Section 4.8, <a href="#">Text Data Formats</a> ). In the chart properties files, the data file's <code>fileType</code> attribute is set to <code>Text</code> .
<code>OutputDataProperties.DATA_FILE_XML</code>	The data is saved to a file in XML data format (see Section 4.7, <a href="#">Loading Data from an XML Source</a> ). In the chart properties file, the data file's <code>fileType</code> attribute is set to <code>Xml</code> .

The `outputFileName` is an absolute file name. If the `saveType` is either `DATA_FILE_TXT` or `DATA_FILE_XML`, the data is saved to the specified file name.

The `propertyName` property specifies the data file using a text string. The text is saved to the chart properties file and used to later reload the data. The `propertyName` and `fileAccess` parameters correspond to the data file's `fileName` and `fileAccess` properties in the saved chart properties file. Recall that the `LoadProperties` object uses the `fileAccess` property to interpret the `fileName` attribute. For more information, see Section 10.2.1, [LoadProperties Class and the fileAccess Property](#).

For more information, see [Saving a Chart to HTML](#), in Chapter 11 and [Saving a Chart to XML](#), in Chapter 12. See also `com.klg.jclass.util.io.OutputDataProperties` in the *API Documentation*.

## 10.4 Saving Image Information: The `OutputProperties` Class

Images are not saved when a chart is saved to HTML or XML. You can, however, choose to save information about the images so that if you reload the chart the images can be located and displayed. Image information includes the file name and how to access it.

Every image in your chart whose information you want to save requires an `OutputProperties` object.

### 10.4.1 Constructing an OutputProperties Object

The following code creates an `OutputProperties` object for use with an image in a `JCFillStyle` object associated with the chart.

```
// Create an instance of OutputProperties
OutputProperties imageOutputProps = new OutputProperties(
    null,                               // outputFileName (not used here)
    "images/bgimage.jpg",              // propertyName
    null,                               // saveType (not used here)
    OutputProperties.RELATIVE_URL);     // fileAccess
```

The `propertyName` property specifies the file name and location of the image. The `fileAccess` property specifies how to interpret the `propertyName`.

- In HTML, `propertyName` is stored as `data.seriesn.fill.image.fileName` and `fileAccess` is stored as `data.seriesn.fill.image.fileAccess`.
- In XML, `propertyName` is stored in the `<image-file>` `fileName` attribute and `fileAccess` is stored in the `<image-file>` `fileAccess` attribute.

For more information, see [Saving Image Information to HTML](#), in Chapter 11 and [Saving Image Information to XML](#), in Chapter 12.

The `outputFileName` and `saveType` properties are null, because this class has other uses and these properties are not required for images. For more information, look up the `com.klg.jcClass.util.io.OutputProperties` class in the *API Documentation*.

### 10.4.2 Setting Output Properties on an Image

Images are specified in fill styles. To prepare to save information about an image to a markup language, you need to set the `OutputProperties` property of the fill style and specify its `OutputProperties` object.

```
// Define an image
String URLString = "http://www.my_site.com/snowflakes.jpg";
URL url = new URL(URLString);

// Load the image (where loadImageFromURL is some method
// that creates an image from a URL)
Image inputImage = loadImageFromURL(url);

// Set the image in the fill style
ChartDataView dv = chart.getDavaView(0);
JCChartStyle cs = dv.getChartStyle(0);
JCFillStyle fs = cs.getFillStyle();
fs.setImage(inputImage);

// Set the output properties for the image
OutputProperties imageOutputProps = new OutputProperties(
    null, URLString, null, OutputProperties.URL);

fs.setOutputProperties(imageOutputProps);
```

# Loading and Saving Charts Using HTML

*Overview of HTML for JClass Chart* ■ *Creating a Chart from HTML*  
*Updating a Chart Using HTML* ■ *Saving a Chart to HTML*

This chapter describes how to create, update, and save a chart using HTML tags to define the chart properties. For more information on the syntax for chart properties, see Appendix B, [HTML Syntax](#).

## 11.1 Overview of HTML for JClass Chart

JClass Chart defines HTML syntax for most chart properties and ships with some HTML examples.

### HTML Syntax for Chart Properties

Most, though not all, chart properties can be specified in HTML. To specify chart properties in HTML, you provide name/value pairs using the syntax and value types outlined in Appendix B, [HTML Syntax](#).

For example, you can specify the chart type using the name `data.chartType` and a value that represents any of the enumerations defined for chart types, such as `AREA`, `BAR`, or `PIE`. The format in which you provide the name/value pairs is dictated by the input medium. The `JCCChartFactory` class contains methods to create a chart using properties read from a file, reader, stream, URL, or string.

For example, when specifying properties in an HTML file, you use the `<PARAM>` tag. The chart property is specified in the `NAME` element and its value in the `VALUE` element. The following properties create a bar chart using data read from a file called *bar.dat*:

```
<PARAM NAME="width" VALUE="400">
<PARAM NAME="height" VALUE="400">
<PARAM NAME="dataFile" VALUE="bar.dat">
<PARAM NAME="data.chartType" VALUE="BAR">
<PARAM NAME="xaxis.annotationMethod" VALUE="Point_Labels">
```

The *bar.dat* file contains the data and the point label annotations:

```
# The data is in array format
# It has one set of x-axis data, multiple sets of y-axis data
# It can have names for the data view and for the x and y series
# Each x point can have a point label
ARRAY `` 5 4
'Network' 'Cache' 'Connect' 'System'
`` 0.0 90.0 180.0 270.0
'Planning' 8.2 17.3 28.4 26.2
'Finance' 35.2 36.1 23.1 38.5
'Research' 35.1 19.7 21.2 20.1
'Sales' 15.1 4.1 8.2 16.1
'Contracts' 16.5 28.1 18.2 0.9
```

## HTML Examples

Many example HTML files are located in the *JCLASS\_HOME/demos/chart/applet* and *JCLASS\_HOME/examples/chart/applet* directories. A pre-built applet, called *JCChartApplet.class*, is also provided. For more information, lookup `com.klg.jclass.chart.property.html.JCChartApplet` in the *API Documentation*.

## 11.2 Creating a Chart from HTML

The *JCChartFactory* class has methods that create a *JCChart* instance from a file, reader, stream, URL, or String. The following sections demonstrate how to create a chart from an HTML file using the *makeChartFromFile()* method.

The examples used in this section create the chart displayed in the following figure:

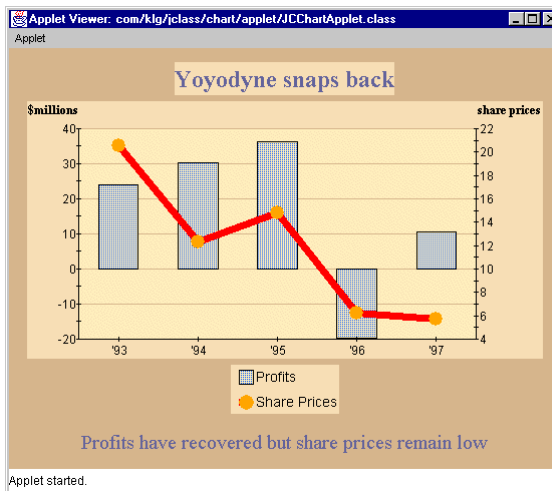


Figure 92 Chart created from HTML tags

### 11.2.1 Specifying JClass Chart Properties Using HTML Tags

When specifying chart properties using HTML, you need to encode the properties within HTML `<PARAM>` tags. The `NAME` element of the `<PARAM>` tag specifies the property name; the `VALUE` element specifies the property value to set. When the file is read in, only the properties inside `<PARAM>` tags are used; all other HTML tags are ignored.

For example, the following line of code shows how to specify the name of the file that contains the chart data:

```
<PARAM NAME="chart.dataFile" VALUE="sample_1.dat">
```

Appendix B, [HTML Syntax](#), contains a list of chart properties by class. Each entry states the HTML syntax for the property (`NAME` element) and its value type (`VALUE` element). The easiest way to create a set of HTML properties is to use the JClass Chart Customizer to save the property values to an HTML file. For more details, see [JClass Chart Customizer](#), in Chapter 1.

For example, the following chart properties were used to create the chart shown at the beginning of this section (Figure 92). The data is embedded in the file. This excerpt is taken from the *yoyodyne.html* demo located in the *JCLASS\_HOME/demos/chart/applet/* directory.

```
<PARAM NAME=background VALUE="210-180-140">
<PARAM NAME=foreground VALUE="black">
<PARAM NAME=font VALUE="Dialog-PLAIN-12">
<PARAM NAME=CustomizeTrigger VALUE="Meta">
<PARAM NAME=allowUserChanges VALUE="true">
<PARAM NAME=footer.y VALUE="55">
<PARAM NAME=footer.font VALUE="TimesRoman-PLAIN-20">
<PARAM NAME=footer.text VALUE="Profits have recovered but share
prices remain low">
<PARAM NAME=footer.visible VALUE="true">
<PARAM NAME=header.border VALUE="bevel|raised">
<PARAM NAME=header.font VALUE="TimesRoman-BOLD-24">
<PARAM NAME=header.background VALUE="245-222-180">
<PARAM NAME=header.text VALUE="Yoyodyne snaps back">
<PARAM NAME=header.visible VALUE="true">
<PARAM NAME=legend.y VALUE="345">
<PARAM NAME=legend.border VALUE="etched|raised">
<PARAM NAME=legend.font VALUE="Dialog-PLAIN-14">
<PARAM NAME=legend.background VALUE="245-222-180">
<PARAM NAME=legend.visible VALUE="true">
<PARAM NAME=legend.anchor VALUE="South">
<PARAM NAME=legend.orientation VALUE="Horizontal">
<PARAM NAME=chartArea.y VALUE="90">
<PARAM NAME=chartArea.border VALUE="bevel|lowered">
<PARAM NAME=chartArea.background VALUE="245-222-180">
<PARAM NAME=chartArea.plotArea.background VALUE="255-232-190">
<PARAM NAME=xaxis.annotationMethod VALUE="Value_Labels">
<PARAM NAME=xaxis.placement VALUE="Min">
<PARAM NAME=xaxis.placementAxis VALUE="yaxis">
<PARAM NAME=xaxis.grid.Color VALUE="210-180-140">
```

```

<PARAM NAME=xaxis.valueLabels VALUE="1.0; '93; 2.0; '94; 3.0;
'95; 4.0; '96; 5.0; '97">
<PARAM NAME=xaxis.title.visible VALUE="false">
<PARAM NAME=yaxis.placement VALUE="Min">
<PARAM NAME=yaxis.grid.visible VALUE="true">
<PARAM NAME=yaxis.grid.Color VALUE="210-180-140">
<PARAM NAME=yaxis.title.font VALUE="TimesRoman-BOLD-12">
<PARAM NAME=yaxis.title.text VALUE="$millions">
<PARAM NAME=chartArea.yaxisName1 VALUE="yaxis1">
<PARAM NAME=yaxis1.placement VALUE="Max">
<PARAM NAME=yaxis1.min VALUE="4.0">
<PARAM NAME=yaxis1.max VALUE="22.0">
<PARAM NAME=yaxis1.grid.Color VALUE="black">
<PARAM NAME=yaxis1.title.font VALUE="TimesRoman-BOLD-12">
<PARAM NAME=yaxis1.title.text VALUE="share prices ">
<PARAM NAME=data.chartType VALUE="BAR">
<PARAM NAME=data.line.color VALUE="black">
<PARAM NAME=data.series1.line.colorIndex VALUE="0">
<PARAM NAME=data.series1.line.width VALUE="8">
<PARAM NAME=data.series1.fill.colorIndex VALUE="0">
<PARAM NAME=data.series1.fill.color VALUE="0-84-255">
<PARAM NAME=data.series1.fill.pattern VALUE="Per_25">
<PARAM NAME=data.series1.symbol.colorIndex VALUE="0">
<PARAM NAME=data.series1.symbol.color VALUE="255-165-0">
<PARAM NAME=data.series1.symbol.size VALUE="7">
<PARAM NAME=data.series1.label VALUE="Profits">
<PARAM NAME=data.Bar.clusterWidth VALUE="50">
<PARAM NAME=data VALUE="
    ARRAY ' ' 1 5 1.0 2.0 3.0 4.0 5.0 24.0 30.2 36.4 -19.8 10.6">
<PARAM NAME=dataName1 VALUE="data1">
<PARAM NAME=data1.outlineColor VALUE="black">
<PARAM NAME=data1.series1.line.colorIndex VALUE="1">
<PARAM NAME=data1.series1.line.color VALUE="red">
<PARAM NAME=data1.series1.line.width VALUE="7">
<PARAM NAME=data1.series1.fill.colorIndex VALUE="1">
<PARAM NAME=data1.series1.symbol.colorIndex VALUE="1">
<PARAM NAME=data1.series1.symbol.color VALUE="255-165-0">
<PARAM NAME=data1.series1.symbol.shape VALUE="Dot">
<PARAM NAME=data1.series1.symbol.size VALUE="14">
<PARAM NAME=data1.series1.label VALUE="Share Prices">
<PARAM NAME=data1.yaxis VALUE="yaxis1">
<PARAM NAME=data1 VALUE="
    ARRAY ' ' 1 5 1.0 2.0 3.0 4.0 5.0 20.5 12.3 14.8 6.2 5.75">

```

## 11.2.2 Creating the Chart and Loading HTML-based Properties

This section demonstrates how to load properties from a file. The file is called *chart.in.html*.

```

LoadProperties loadProps = new LoadProperties();

String inFile = "chart.in.html";
String chartName = "myChart";
JCCChart chart = null;

```

```

try {
    chart = JCChartFactory.makeChartFromFile(inFile,
                                             loadProps, chartName, JCChartFactory.HTML);
}
catch (JCIOException e) {
    System.out.println("Error accessing external file:" +
                      e.getMessage());
}
catch (JCParseException e) {
    System.out.println("Error parsing file:" +
                      e.getMessage());
}
catch (IOException e) {
    System.out.println("Error reading " + inFile + ":" +
                      e.getMessage());
}
}

```

where

- `inFile` is an input file containing JClass Chart properties in HTML.
- `loadProps` is a `LoadProperties` object containing properties that specify how to load the chart (see [Overview of the LoadProperties Class](#), in Chapter 10).
- `chartName` is the name of the chart, and is the name with which each of the properties begins. If there is more than one chart in the HTML file, only the parameters beginning with that name are assigned to the chart. If there is only one set of chart parameters stored in the file, the name can be dropped and an empty String passed as the third parameter.
- `JCChartFactory.HTML` specifies HTML as the markup language used.

## 11.3 Updating a Chart Using HTML

You can update an existing chart with new chart properties or with a new data set.

### 11.3.1 Updating a Chart with New Chart Properties

You can update existing charts with new chart properties using the update methods in the `JCChartFactory` class. The new properties can come from a file, reader, stream, URL, or a String. Properties that are unspecified retain their existing values.

For example, the following code sample updates a chart using properties defined in an HTML file:

```

String updateFile = "chart.update.html";

try {
    JCChartFactory.updateChartFromFile(chartName,
                                       updateFile, loadProps, name, JCChartFactory.HTML)
}

```

```

catch (JCIOException e) {
    System.out.println("Error accessing external file:" +
        e.getMessage());
}
catch (JCParseException e) {
    System.out.println("Error parsing file:" + e.getMessage());
}
catch (IOException e) {
    System.out.println("Error reading " + updateFile + ":" +
        e.getMessage());
}

```

where

- `chartName` is the chart to update.
- `updateFile` is an input file containing JClass Chart properties in HTML.
- `loadProps` is a `LoadProperties` object containing properties that specify how to load the chart (see [Overview of the LoadProperties Class](#), in Chapter 10).
- `name` of the chart being read from parameters. If non-null, only parameters of the form `name.parameter` are applied to this chart.
- `JCChartFactory.HTML` specifies HTML as the markup language used.

For more information, lookup `JCChartFactory` in the *API Documentation* and review the update methods.

### 11.3.2 Updating a Chart with a New Data Set

`JCChartFactory` also has a method, `updateChartWithData()`, that updates a chart with a new data set. In the given data view, the old data set is replaced by the new data set, with information provided by a file, reader, or an input stream.

**Note:** You need to use the underlying data model to complete this task. For more information, see Chapter 4, [Adding Data with the Underlying Data Model](#) and the `ChartDataView` and `JCChartFactory` classes in the *API Documentation*.

For example, the following code sample updates a chart with data defined in a text file:

```

String updateDataFile = "newchartdata.txt";

try{
    JCChartFactory.updateChartWithData(chartName,
        JCChartFactory.DATA_FILE_TEXT, updateDataFile,
        dataViewIndex, loadProps);
}

catch (IOException e) {
    System.out.println("Error reading " + updateDataFile + ":" +
        e.getMessage());
}

```



where

- `chartName` is the chart to update.
- `JCChartFactory.DATA_FILE_TEXT` specifies that the data is in a text file. Alternatively, you can specify an XML data file using `JCChartFactory.DATA_FILE_XML`.
- `updateDataFile` is the name of the file containing the data. Alternatively, data could be coming from a Reader or an `InputStream`.
- `dataViewIndex` is the index of the `ChartDataView` on which the data is to be set (there is also a method where the `ChartDataView` can be specified by name).
- `loadProps` is a `LoadProperties` object containing properties that specify how to load the chart (see [Overview of the LoadProperties Class](#), in Chapter 10).

This method creates a data source from the data object, and sets this new data source on the appropriate `ChartDataView` on the chart.

## 11.4 Saving a Chart to HTML

The `JCChartFactory` class also has methods that save a `JCChart` instance to a stream, file, writer, or `String`.

**Note:** You need to use the underlying data model to complete this task. For more information, see Chapter 4, [Adding Data with the Underlying Data Model](#) and the `ChartDataView` and `JCChartFactory` classes in the *API Documentation*.

The following example code saves chart properties to a file called *chart.out.html* and the chart data to a file called *chart.dat*.

```
String outFile      = "chart.out.html";
String outDataFile = "chart.dat";
ChartDataView dv = chart.getDataView(0);

if (dv != null) {
    OutputDataProperties outProps = dv.getOutputDataProperties();
    if (outProps == null) {
        outProps = new OutputDataProperties();
    }

    // Save data to a text file, which can be accessed via a URL
    outProps.setOutputFileName(outDataFile);
    outProps.setPropertyName("file:///C:/jclass_home/" + outDataFile);
    outProps.setSaveType(OutputDataProperties.DATA_FILE_TXT);
    outProps.setFileAccess(OutputDataProperties.URL);
    dv.setOutputDataProperties(outProps);
}
```

```

try {
    JCChartFactory.saveChartToFile(chart, outFile,
                                   JCChartFactory.HTML);
}
catch (IOException e) {
    System.out.println("Error writing to " + outFile + ":" +
                       e.getMessage());
}

```

### 11.4.1 Saving Data When a Chart is Saved to HTML

In the above example, `outProps` is an instance of `OutputDataProperties`. The `OutputDataProperties` instance specifies that the data is to be saved to a file named *chart.dat* in text format. When loading the chart, the chart can access the data via the URL *file:///C:/jclass\_home/*.

For more information, see [Saving Data: The OutputDataProperties Class](#), in Chapter 10.

### 11.4.2 Saving Image Information to HTML

When a chart containing an image is saved to HTML, the information about the image that is contained in its `OutputProperties` object is also saved. If an image does not have an `OutputProperties` object associated with it, the image is ignored.

For example, consider an image that is used as a fill style for a threshold. If the image has the following `OutputProperties` object associated with it:

```

OutputProperties imageOutputProps = new OutputProperties(
    null, "threshold.jpg", null, OutputProperties.ABSOLUTE);

```

the information about the image is stored as:

```

<PARAM NAME=data.thresholdn.fill.image.fileName VALUE="threshold.jpg">
<PARAM NAME=data.thresholdn.fill.image.fileAccess VALUE="Absolute">

```

where *data* is the name of the dataset.

The image file *threshold.jpg* will be interpreted an absolute file name when the properties in the HTML file are loaded back into a chart.

Note that HTML elements map to the `OutputProperties` object as follows:

- *data.thresholdn.fill.image.fileName* maps to the `propertyName` parameter.
- *data.thresholdn.fill.image.fileAccess* maps to the `fileAccess` parameter.

For more information, see [Saving Image Information: The OutputProperties Class](#), in Chapter 10.

# Loading and Saving Charts Using XML

*Background XML Information ■ Overview of XML for JClass Chart  
Creating a Chart Using XML ■ Updating a Chart Using XML ■ Saving a Chart to XML  
Internationalizing Your XML-based Chart*

This chapter describes how to create a chart using XML tags to define chart properties. The first two sections provide some general XML information followed by an overview of how JClass Chart implements XML. The next three sections describe how to create, update, and save a chart. The last section deals with the topic of internationalizing XML-based charts. For more information, see Appendix C, [XML DTD](#).

## 12.1 Background XML Information

### XML Primer

XML – eXtensible Markup Language – is a scaled-down version of SGML (Standard Generalized Markup Language), the standard for creating a document structure. XML was designed especially for web documents, and allows designers to create customized tags (“extensible”), thereby enabling common information formats for sharing both the format and the data on the Internet, intranets, et cetera.

XML is similar to HTML in that both contain markup tags to describe the contents of a page or file. But HTML describes the content of a web page (mainly text and graphic images) only in terms of how it is to be displayed and interacted with. XML, however, describes the content in terms of what data is being described. This means that an XML file can be used in various ways. For instance, an XML file can be utilized as a convenient way to exchange data across heterogeneous systems. As another example, an XML file can be processed (for example, via XSLT [Extensible Stylesheet Language Transformations]) in order to be visually displayed to the user by transforming it into HTML.

In XML, certain special characters need to be “escaped” if you want them to be displayed. For example, you cannot simply put an ampersand (&) or a greater than sign (>) into a block of text; these special characters are represented as `&amp;` and `&gt;` respectively. See [http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/sax/4\\_refs.html#chars](http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/sax/4_refs.html#chars).

## DTD Primer

The document type definition (DTD) file has one purpose: to specify the structure of an XML file. The DTD describes, in XML Declaration Syntax, the particular type of document, and sets out what names are to be used for various elements, where these elements may occur, and how they work together.

## Further Information About XML

Here are links to more information on XML.

<http://www.w3.org/XML/> – another W3C site; contains information on standards

<http://www.ucc.ie/xml> – an extensive FAQ devoted to XML

<http://java.sun.com/docs/index.html> – Sun’s XML site

## 12.2 Overview of XML for JClass Chart

JClass Chart ships with the following DTDs and examples for XML.

### DTDs

DTD files are located in *JCLASS\_HOME/xml-dtd*.

- *Chart.dtd* – Defines chart elements.
- *JCChartData.dtd* – Defines chart data elements. Requires *Chart.dtd*.

In JClass Chart, the elements, sub-elements, and attributes in the DTDs, for the most part coincide with the objects, sub-objects, and properties within the chart component. Properties are specified as Strings in the XML file. The Strings are converted to the appropriate type by the JClass Chart XML handler. For more information, see Appendix C, [XML DTD](#).

### XML Examples

The *XMLCharts* demo and its sample XML files are in *JCLASS\_HOME/demos/chart/xml*. You can also load and save XML files using the JClass Chart Customizer. For more information, see [JClass Chart Customizer](#), in Chapter 1.

## 12.3 Creating a Chart Using XML

The `JCChartFactory` class has methods that create a `JCChart` instance from a file, reader, stream, URL, or String. The following sections demonstrate how to create a chart from an XML file using the `makeChartFromFile()` method.

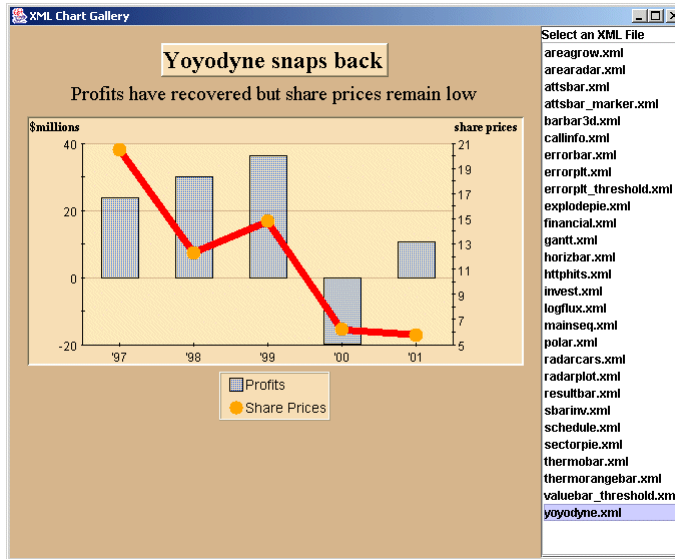


Figure 93 Chart created using the XMLCharts demo with the yoyodyne.xml file selected.

### 12.3.1 Specifying JClass Chart Properties Using XML Elements

The XML elements for JClass Chart are defined in the DTD files that ship with JClass DesktopViews. For your convenience, the elements and subelements are also listed in Appendix C, [XML DTD](#). The easiest way to create a set of XML chart properties is to use the JClass Chart Customizer to save the property values to an XML file. For more details, see [JClass Chart Customizer](#), in Chapter 1.

The following example shows the XML elements used to define the chart in Figure 93. This code is from the *yoyodyne.xml* file located in the *JCLASS\_HOME/demos/chart/xml/* directory. It runs with the *XMLCharts* demo located in the same directory.

```
<?xml version="1.0"?>
<!DOCTYPE chart SYSTEM "Chart.dtd">
<chart name=""
      allowUserChanges="true"
      width="550"
      height="420">
  <component background="210-180-140"
             foreground="black"
             font="Dialog-PLAIN-12" />
  <event-trigger trigger="Customize"
                modifier="Meta" />
  <event-trigger trigger="Customize"
                modifier="Meta" />
  <header text="Yoyodyne snaps back">
    <component background="245-222-180">
```

```

        opaque="true"
        font="TimesRoman-BOLD-24"
        visible="true">
    <bevel-border soft="false"
        type="Raised"
        highlightColor="white"
        shadowColor="119-108-87" />
</component>
</header>
<footer text="Profits have recovered but share prices remain low">
    <component font="TimesRoman-PLAIN-20"
        visible="true" />
    <layout-hints y="55" />
</footer>
<legend anchor="South"
    orientation="Horizontal">
    <component background="245-222-180"
        opaque="true"
        foreground="black"
        font="Dialog-PLAIN-14"
        visible="true">
        <etched-border type="Raised"
            highlightColor="white"
            shadowColor="171-155-125" />
    </component>
    <layout-hints y="345" />
</legend>
<chart-area>
    <component background="245-222-180"
        opaque="true"
        foreground="black"
        font="Dialog-PLAIN-12">
        <bevel-border soft="false"
            type="Lowered"
            highlightColor="white"
            shadowColor="119-108-87" />
    </component>
    <layout-hints y="90" />
    <plot-area foreground="black"
        background="255-232-190" />
    <axis type="XAxis"
        name="xaxis"
        annotationMethod="Value_Labels"
        placement="Min"
        placementAxis="yaxis">
    <chart-interior-region font="Dialog-PLAIN-12"
        foreground="black"
        background="245-222-180">
    </chart-interior-region>
    <axis-title>
        <chart-interior-region font="Dialog-PLAIN-12"
            foreground="black"
            background="245-222-180">
        </chart-interior-region>
    </axis-title>
    <value-label value="1.0">&apos;97</value-label>

```

```

        <value-label value="2.0">&apos;98</value-label>
        <value-label value="3.0">&apos;99</value-label>
        <value-label value="4.0">&apos;00</value-label>
        <value-label value="5.0">&apos;01</value-label>
        <line-style color="210-180-140" />
    </axis>
    <axis type="YAxis"
        name="yaxis"
        placement="Min"
        gridVisible="true">
        <chart-interior-region font="Dialog-PLAIN-12"
            foreground="black"
            background="245-222-180">

        </chart-interior-region>
        <axis-title text="$millions">
            <chart-interior-region font="TimesRoman-BOLD-12"
                foreground="black"
                background="245-222-180"
                visible="true">

            </chart-interior-region>
        </axis-title>
        <line-style color="210-180-140" />
    </axis>
    <axis type="YAxis"
        name="yaxis1"
        placement="Max">
        <chart-interior-region font="Dialog-PLAIN-12"
            foreground="black"
            background="245-222-180">

        </chart-interior-region>
        <axis-title text="share prices ">
            <chart-interior-region font="TimesRoman-BOLD-12"
                foreground="black"
                background="245-222-180"
                visible="true">

            </chart-interior-region>
        </axis-title>
        <line-style color="black" />
    </axis>
</chart-area>
<chart-data-view chartType="Bar"
    name=" ">
    <chart-data name=" " hole="max">
        <data-series>
            <x-data>1.0</x-data>
            <x-data>2.0</x-data>
            <x-data>3.0</x-data>
            <x-data>4.0</x-data>
            <x-data>5.0</x-data>
            <y-data>24.0</y-data>
            <y-data>30.2</y-data>
            <y-data>36.4</y-data>
            <y-data>-19.8</y-data>
            <y-data>10.6</y-data>
        </data-series>
    </chart-data>

```

```

<bar-format clusterWidth="50" />
<line-style color="black" />
<chart-data-view-series label="Profits">
  <chart-style>
    <line-style color="red"
      width="8" />
    <fill-style color="0-84-255"
      pattern="Per_25" />
    <symbol-style color="255-165-0"
      shape="Dot"
      size="7" />
  </chart-style>
</chart-data-view-series>
</chart-data-view>
<chart-data-view name=" "
  yaxis="yaxis1">
  <chart-data name=" " hole="max">
    <data-series>
      <x-data>1.0</x-data>
      <x-data>2.0</x-data>
      <x-data>3.0</x-data>
      <x-data>4.0</x-data>
      <x-data>5.0</x-data>
      <y-data>20.5</y-data>
      <y-data>12.3</y-data>
      <y-data>14.8</y-data>
      <y-data>6.2</y-data>
      <y-data>5.75</y-data>
    </data-series>
  </chart-data>
  <line-style color="black" />
  <chart-data-view-series label="Share Prices">
    <chart-style>
      <line-style color="red"
        width="7" />
      <fill-style color="orange" />
      <symbol-style color="255-165-0"
        shape="Dot"
        size="14" />
    </chart-style>
  </chart-data-view-series>
</chart-data-view>
</chart>

```



### 12.3.2 Creating the Chart and Loading XML-based Properties

The following code sample creates a chart using properties defined in an XML file. The file is called *chart.in.xml*.

```
public CreateXMLChart()
{
    LoadProperties loadProps = new LoadProperties();

    String inFile = "chart.in.xml";
    String chartName = "myChart";
    JCCChart chart = null;
    try {
        chart = JCCChartFactory.makeChartFromFile(inFile, loadProps,
            chartName, JCCChartFactory.XML);
    }
    catch (JCIOException e) {
        System.out.println("Error accessing external file:" +
            e.getMessage());
    }
    catch (JCParseException e) {
        System.out.println("Error parsing file:" + e.getMessage());
    }
    catch (IOException e) {
        System.out.println("Error reading " + inFile + ":" +
            e.getMessage());
    }

    add(chart);
}
```

where

- `inFile` is an input file containing JClass Chart properties in XML.
- `loadProps` is a `LoadProperties` object containing properties that specify how to load the chart (see [Overview of the LoadProperties Class](#), in Chapter 10).
- `chartName` is set as the name property of the chart.
- `JCCChartFactory.XML` specifies XML as the markup language used.

## 12.4 Updating a Chart Using XML

You can update an existing chart with new chart properties or with a new data set.

### 12.4.1 Updating a Chart with New Chart Properties

You can update existing charts with new chart properties using the update methods in the `JCCChartFactory` class. The new properties can come from a file, reader, stream, URL, or a `String`. Properties that are unspecified retain their existing values.

For example, the following code sample updates a chart using properties defined in an XML file.

```
try {
    JCCartFactory.updateChartFromFile(chartName, updateFile,
                                      loadProps, null, JCCartFactory.XML);
}
catch (JCIOException e) {
    System.out.println("Error accessing external file:" +
                      e.getMessage());
}
catch (JCParseException e) {
    System.out.println("Error parsing file:" + e.getMessage());
}
catch (IOException e) {
    System.out.println("Error reading " + updateFile + ":" +
                      e.getMessage());
}
```

where

- `chartName` is the name of the chart to update.
- `updateFile` is an input file containing JClass Chart properties in XML.
- `loadProps` is a `LoadProperties` object containing properties that specify how to load the chart (see [Overview of the LoadProperties Class](#), in Chapter 10).
- `name` parameter is only used in HTML.
- `JCCartFactory.XML` specifies XML as the markup language used.

## 12.4.2 Updating a Chart with a New Data Set

`JCCartFactory` also has a method, `updateChartWithData()`, that updates a chart with a new data set. In the given data view, the old data set is replaced by the new data set, with information provided by a file, reader, or an input stream.

**Note:** You need to use the underlying data model to complete this task. For more information, see Chapter 4, [Adding Data with the Underlying Data Model](#) and the `ChartDataView` and `JCCartFactory` classes in the *API Documentation*.

For example, the following code sample updates a chart with data from a data file:

```
try{
    JCCartFactory.updateChartWithData(chartName,
                                      JCCartFactory.DATA_FILE_TEXT, updateDataFile
                                      dataViewIndex, loadProps);
}

catch (IOException e) {
    System.out.println("Error reading " + updateDataFile + ":" +
                      e.getMessage());
}
```

where

- `chartName` is the chart to update.
- `JCChartFactory.DATA_FILE_TEXT` specifies that the data is in a text file. Alternatively, you can specify an XML data file using `JCChartFactory.DATA_FILE_XML`.
- `updateDataFile` is the name of the file containing the data. Alternatively, data could be coming from a Reader or an InputStream.
- `dataViewIndex` is the index of the `ChartDataView` on which the data is to be set (there is also a method where the `ChartDataView` can be specified by name).
- `loadProps` is a `LoadProperties` object containing properties that specify how to load the chart (see [Overview of the LoadProperties Class](#), in Chapter 10).

This method creates a data source from the data object, and sets this new data source on the appropriate `ChartDataView` on the chart.

## 12.5 Saving a Chart to XML

The `JCChartFactory` class has methods that save a `JCChart` instance to a stream, file, or String. The following code sample saves a chart to the file *chart.out.xml* and the data to a file called *chart.dat.xml*.

```
public class SaveXMLChart extends JPanel
{
    private JCChart chart = null;

    public SaveXMLChart()
    {
        chart = new JCChart();
        chart.setPreferredSize(new Dimension(400, 400));
        ChartDataView dv = chart.getDataView(0);
        dv.setDataSource(new JCDefaultDataSource());
        add(chart);
    }

    public void saveChart()
    {
        ChartDataView dv = chart.getDataView(0);
        String outFile = "chart.out.xml";
        String outDataFile = "chart.dat.xml";

        // Set properties for saving the chart's data
        if (dv != null) {
            OutputDataProperties outProps = dv.getOutputDataProperties();
            if (outProps == null) {
                outProps = new OutputDataProperties();
            }
        }
    }
}
```

```

        // Save data to an XML file, which can be accessed using
        // an absolute file name
        outProps.setOutputFileName(outDataFile);
        outProps.setSaveType(OutputDataProperties.DATA_FILE_XML);
        outProps.setFileAccess(OutputDataProperties.ABSOLUTE);
        dv.setOutputDataProperties(outProps);
    }

    // Save the chart and the data
    try {
        JCChartFactory.saveChartToFile(chart, outFile,
                                       JCChartFactory.XML);
    }
    catch (IOException e) {
        System.out.println("Error writing to " + outFile + ":" +
                           e.getMessage());
    }
}

public static void main(String args[])
{
    JCExitFrame f = new JCExitFrame("Save JClass Chart XML Example");
    f.setSize(new Dimension(450, 450));
    SaveXMLChart s = new SaveXMLChart();
    f.getContentPane().add(s);
    f.setVisible(true);

    s.saveChart();
}
}

```

### 12.5.1 Saving Data When a Chart is Saved to XML

In the above example, `outProps` is an instance of `OutputDataProperties`. The `OutputDataProperties` instance specifies that the data is to be saved to a file named *chart.dat.xml* in XML data format. When loading the chart, the chart can use the file name for the data file as an absolute file name.

For more information, see [Saving Data: The `OutputDataProperties` Class](#), in Chapter 10.

### 12.5.2 Saving Image Information to XML

When a chart containing an image is saved to XML, the information about the image that is contained in its `OutputProperties` object is stored in an `<image-file>` element. An `<image-file>` element is nested within the `<fill-style>` element. If an image does not have an `OutputProperties` object associated with it, the image is ignored.

For example, consider an image that is used as a fill style for a threshold. If the image has the following `OutputProperties` object associated with it:

```
OutputProperties imageOutputProps = new OutputProperties(
    null, "threshold.jpg", null, OutputProperties.ABSOLUTE);
```

information about the image is stored as:

```
<chart>
...
  <threshold...>
    <fill-style...>
      <image-file fileName="threshold.jpg"
                  fileAccess="Absolute"/>
    </fill-style>
  </threshold>
</chart>
```

The image file *threshold.jpg* will be interpreted an absolute file name when the properties in the HTML file are loaded back into a chart.

Note that the attributes in `<image-file>` map to the `OutputProperties` object as follows:

- `fileName` attribute maps to the `propertyName` parameter.
- `fileAccess` attribute maps to the `fileAccess` parameter.

For more information, see [Saving Image Information: The OutputProperties Class](#), in Chapter 10.

## 12.6 Internationalizing Your XML-based Chart

If you need to offer your XML-based chart in multiple languages, you can replace the text strings with variables and provide a resource bundle containing properties files or `ResourceBundle` classes for each language that you support. When your client's browser requests the chart, the browser's locale or, for JSF, the client's operating system locale, determines which language is displayed by default. The following sections describe how to add variables to your chart, create the resource bundle, and use your resource bundle in different environments.

### 12.6.1 Using Variables

Wherever text appears on your chart, you need to replace the text string with a variable in your XML file. For example, you can use variables for the header, footer, axes titles, data view name, series labels, data point labels, and text in a URL. Variables take the form `${KEY}`, where `KEY` is a unique variable name. Variable names are case-sensitive and can be uppercase, lowercase, or mixed case. Try to use meaningful names so that, when you create your resource bundle, it is easier to map the correct text strings to the variables.

**Note:** If you want, you can embed a variable within a text string. For example, you could specify a value such as “This is a \${KEY}”. Mixing text and variables, however, is not generally recommended; you usually want to provide charts with the text entirely in your client’s language.

### 12.6.2 Creating a Resource Bundle

Depending on your needs, you can use the class `ResourceBundle` or either of its subclasses – `PropertyResourceBundle` or `ListResourceBundle` – to assign text strings to the variables that you used in your XML file. `PropertyResourceBundle` looks for the localized strings in properties files, while `ResourceBundle` and `ListResourceBundle` looks for them in your code. For more information on how to use these classes, locate `java.lang.ResourceBundle` in the *Java API documentation*.

Whichever method you choose, your resource bundle must include a default locale – the language used when a locale is not specified by the browser – plus a properties file or `ResourceBundle` class for each of the other languages that you want to support. The default locale uses the base name of your resource bundle, while all other locales should follow the I18N naming conventions for language and country, though the country code is optional if there is no chance of confusion. For example, your base name and default locale could be called *myresources*, while U.S. English would be *myresources\_en\_US*, and German would be *myresources\_de\_DE* (or *myresources\_de*). For more information, see [Internationalization](#), in Chapter 1.

### 12.6.3 Using Resource Bundles

To load a localized chart, you need to implement a locale handler using the `com.klg.jclass.util.LocaleHandler` interface. You can use the implementation provided with JClass Chart, called `LocaleBundle`, or you can create your own locale handler by implementing the `LocaleHandler` interface.

To use `LocaleBundle`, you create an instance of `LocaleBundle` and specify the fully qualified pathname to the base name of your resource bundle and a `Locale` object for the locale you want to use. To tell the factory how to find your resource bundle, you set the `localeHandler` property of your `LoadProperties` object to point to your `LocaleBundle` object.

In the following example, the programmer creates a `LocaleBundle` object called `localeBundle` and specifies the location of the bundle and a `Locale` object. The location of the `localeBundle` is in a fictitious *JCLASS\_HOME/demos.chart.xml.resources* directory and the bundle base name is *XMLLocaleInfo*. The `Locale` object is defined elsewhere and passed as a parameter to the `createChart()` method.

The instance of `LocaleBundle` is then passed to the `setLocaleHandler()` method for the `loadProperties` object. Finally, the `loadProperties` object is passed to the factory’s `makeChartFromFile()` method.

```

import com.klg.jclass.chart.JCChartFactory;
import com.klg.jclass.chart.JCChart;
...
import com.klg.jclass.util.LocaleBundle;
...
import java.util.Locale;

public class XMLExample extends JPanel implements ActionListener
{
    ...
    public JCChart createChart(Locale locale)
    {
        JCChart chart = null;
        // Create the chart
        try {
            LoadProperties loadProperties = new LoadProperties();
            LocaleBundle localeBundle = new LocaleBundle(
                "demos.chart.xml.resources.XMLLocaleInfo", locale);
            loadProperties.setLocaleHandler(localeBundle);
            URL url = getClass().getResource("chart.xml");
            chart = JCChartFactory.makeChartFromFile(
                url, loadProperties, JCChartFactory.XML);
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        return chart;
    }
    ...
}

```





## JClass Chart Beans

*Choosing the Right Bean ■ Standard Bean Properties ■ Data-Loading Methods*

This chapter is a reference for JClass Chart beans and their properties. For basic JavaBean concepts and a tutorial, see the [SimpleChart Bean Tutorial](#), in Chapter 14.

### 13.1 Choosing the Right Bean

When creating new applications in an IDE, you can use either `MultiChart` or `SimpleChart`. We recommend using `MultiChart`, both for learning JClass Chart's features and for creating new applications.

#### The MultiChart Bean

`MultiChart` is JClass Chart's most powerful Bean. It contains a richer set of features than previous Beans, highlighting the superiority of JClass Chart as a charting application tool. Among its features are the ability to handle multiple data sources and multiple axes. For more information, see [MultiChart Bean](#), in Chapter 15.

#### SimpleChart

`SimpleChart` was designed for quick chart development in any IDE environment. It exposes the most commonly used charting properties, and presents them in easy-to-use property editors. `SimpleChart` can load data from a file or a design-time editor.

`SimpleChart` and the data-binding Beans share a common set of properties that are covered in this chapter. `SimpleChart` and the data-binding Beans only differ in how they load data. Therefore, this chapter is divided into [Standard Bean Properties](#) and [Data-Loading Methods](#).

### 13.1.1 JClass Chart Beans

The following table shows all of the available JClass Beans and their uses:

JClass Chart Bean	Description
MultiChart	<p>The most powerful charting Bean.</p> <ul style="list-style-type: none"><li>■ Charts data from two data sources and plots them against multiple axes.</li><li>■ Data sources can be a file, or data entered at design-time. Also supports using Swing <code>TableModel</code> objects as data sources.</li><li>■ Compatible with all IDEs.</li></ul> <p>See <a href="#">MultiChart Bean</a>, in Chapter 15, for complete details.</p>
SimpleChart	<p>Charts data from a file or data entered at design-time. Also supports a Swing <code>TableModel</code> object as a data source.</p> <p>Compatible with all IDEs.</p>

### 13.1.2 JClass Chart Beans and JCChart

All JClass Chart Beans are subclasses of the main chart object, `JCChart`. This means that the entire JClass Chart API is available to any developer using any of the Beans.

## 13.2 Standard Bean Properties

`SimpleChart` has a set of standard properties that allow you to control the appearance and behavior of your charts.

### 13.2.1 Axis Properties

JClass Chart Beans set up basic axis properties for you automatically, and adjust these properties to your data. You can also customize your axes with the axes property editors. You have control over the following axis properties:

- Axis Titles
- Annotation Method
- Axis Number Intervals
- Axis Range
- Axis Grids
- Axis Hiding
- Logarithmic Notation
- Axis Orientation

## Axis Titles

Enter X- and Y-axis titles in the `xAxisTitleText` and `yAxisTitleText` property editors:

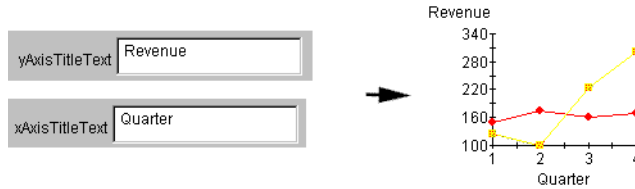


Figure 94 Axis titles.

## Annotation Method

Set the annotation method for the axes using the `xAnnotationMethod` and `yAnnotationMethod` editors. By default, Value annotation is used for both:

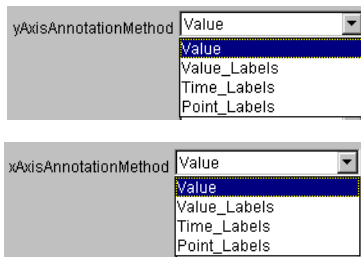


Figure 95 Annotation methods for x and y axes.

`Value_Labels` notation can only be added programmatically or by using HTML parameters; therefore, it is not very useful for Bean programming. The following examples show the three applicable annotation methods as applied to the X-axis:

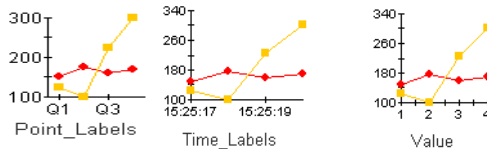


Figure 96 Annotation methods.

### Axis Number Intervals

To specify the number interval on the axes, enter the interval into the `yAxisNumSpacing` or `xAxisNumSpacing` property editors:



Figure 97 Axis number spacing.

### Axis Range

The axis number range is determined by the minimum and maximum values of the axes. By default, these values are set automatically, based on the available data. You can specify the range by using the `xAxisMinMax` and `yAxisMinMax` property editors. Enter the minimum value on the left of the comma, and the maximum on the right:

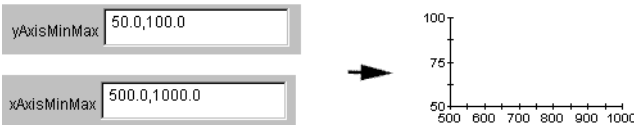


Figure 98 Axis range.

### Logarithmic Notation

You can specify that one or both of the axes are logarithmic by setting the `xAxisLogarithmic` or `yAxisLogarithmic` properties to true:

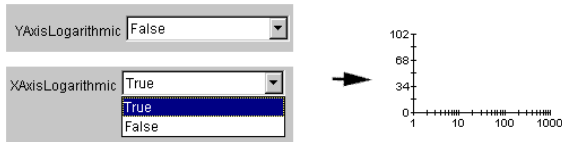


Figure 99 Logarithmic notation.

### Hiding Axes

By default, both the X- and Y-axes are displayed. You can hide them by setting the `xAxisVisible` or `yAxisVisible` properties to false. The following example hides the Y-axis:

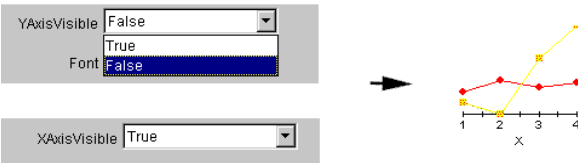


Figure 100 Hidden axes

## Showing Grids

Display gridlines for one or both axes by setting the `xAxisGridVisible` or `yAxisGridVisible` properties to `true`. By default, the grids are hidden. The following example sets both axes to display gridlines:



Figure 101 Gridlines.

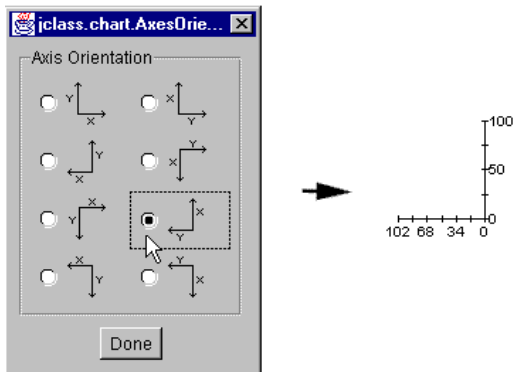
## Axis Orientation

Axis orientation determines how the axes are positioned on the chart. By default, the axes are positioned with the Y-axis left/vertical and the X-axis right/horizontal. Use the axis orientation custom editor to change how your axes are oriented. To launch the custom editor, click the `axisOrientation` property:



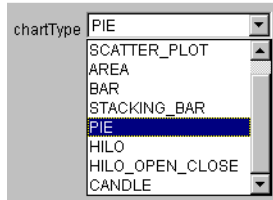
Figure 102 Axis orientation.

Select the desired orientation and click **Done**.



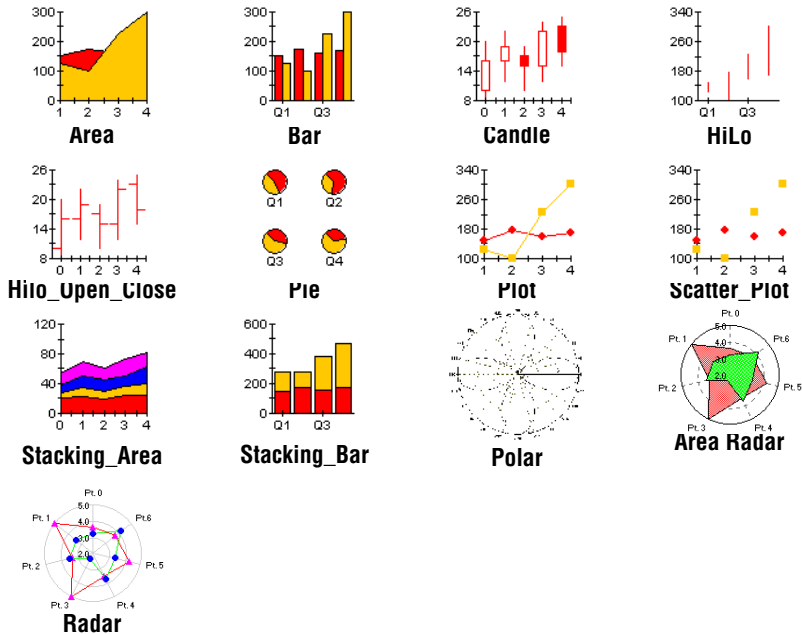
## 13.2.2 Chart Types

By default, JClass Chart Beans use the Plot chart type to display data. To change to another type, use the `chartType` property editor. The following example selects the `PIE` type:



### Data Interpretation

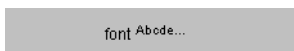
The following examples show how data is displayed by the different chart types:



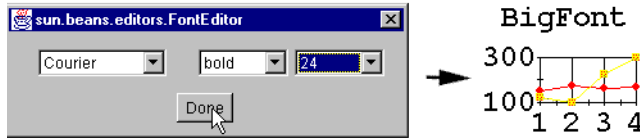
## 13.2.3 Display Properties

### Font

Set the size and style of text on your chart by clicking the `font` property:



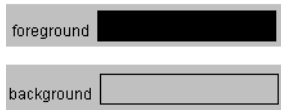
The font you choose will apply to all text on the chart simultaneously with the exception of the header and footer. Note that the font editor that appears in your IDE may be different from the example below. The following example sets the font to **Courier, Bold, 24 point**, with the BeanBox font editor:



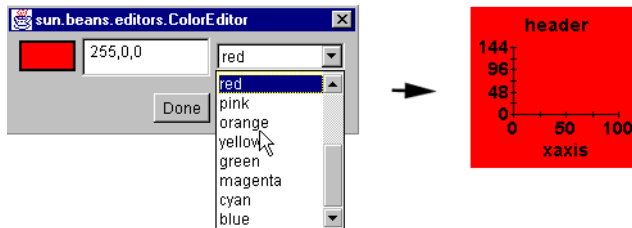
**Note:** The different font properties all work in the same way. Font affects all text on the chart area and legend. Header font affects the header, and Footer Font affects the footer.

### Foreground and Background Colors

Click the foreground and background properties to set the foreground and background colors of your chart. A color editor will appear. By default, the colors are black foreground and light-gray background:



Most IDEs have their own color editors that differ from the BeanBox. The following example sets the background color to red:



### 3D Effects

To add 3D effects to your chart, click the View3D property:

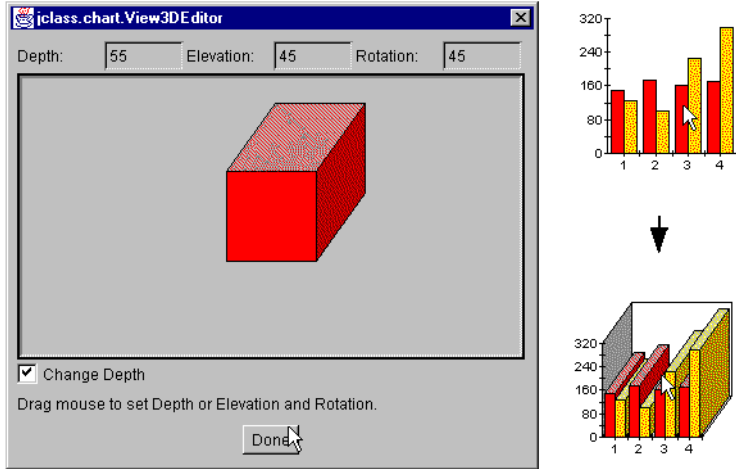


This will bring up the **View3DEditor**. There are two main settings in the **View3DEditor**: depth, and combined elevation and rotation.

You can add 3D effects either by typing a value in the editable box next to the Depth, Elevation, and Rotation settings, or by dragging the red square in the editor until it has the desired Elevation and Rotation. Then, check the **Change Depth** option box, and

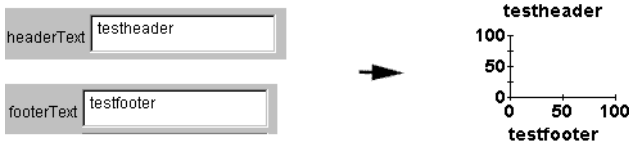
drag the red square until it has the Depth you want; alternatively, simply type in the value in the editable box next to this setting.

The degree of depth, elevation, and rotation is displayed in numbers at the top of the editor. Click **Done** to set the changes:



### 13.2.4 Headers and Footers

Add a header, footer, or both with the `headerText` and `footerText` property editors. The following example sets both:



The font characteristics of the header and footer are determined by the Header Font and Footer Font properties. See Section 13.2.3, [Display Properties](#), for more details.

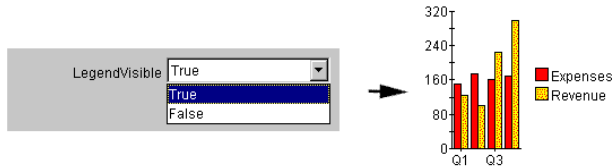
### 13.2.5 Legends

You can add a legend, position it, and select its layout. The legend is set up from information in the data source. For information on how to set up legend items in the data source, see [Text Data Formats](#), in Chapter 4.



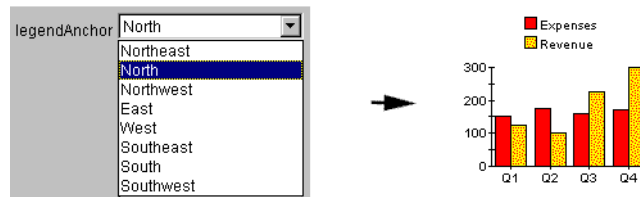
## Showing the Legend

To show the legend, set the `legendVisible` property to `true`:



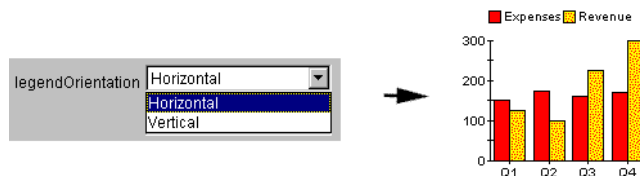
## Legend Placement

Specify where the legend will be anchored in the chart area by selecting a compass direction from the `legendAnchor` property options. By default, legends are anchored on the East. The following example anchors the legend North:



## Legend Layout

Legend items can be laid out vertically or horizontally. By default the legend has a vertical layout. To specify a horizontal layout, set the `legendOrientation` property to `Horizontal`:



## 13.3 Data-Loading Methods

This section covers the data-loading methods of `SimpleChart`. For `MultiChart` data-

loading details, see [MultiChart Bean](#), in Chapter 15.

JClass Chart Bean	Data Source & IDE Compatibility
SimpleChart	<ul style="list-style-type: none"><li>■ Formatted file or design-time editor.</li><li>■ Also supports using a Swing <code>TableModel</code> object as the data source.</li><li>■ All IDEs.</li></ul>

### 13.3.1 SimpleChart: Loading Data from a File

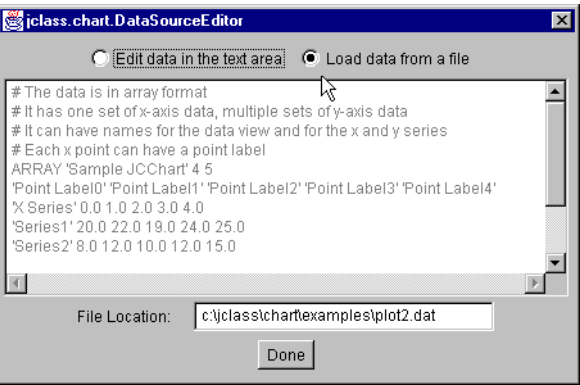
There are two ways of loading data with the `SimpleChart` Bean: from a `.dat` file, or by entering data directly into the custom editor. Both methods are managed by the **DataSourceEditor**. To bring up the **DataSourceEditor**, click on the `data` property:

data

The `DataSource` Editor will appear (see below).

#### Loading Data from a `.dat` File

To load data from a file, click **Load data from a file**, enter the name of the file in the **File Location** field, and click **Done**:



Specify the full path of the file. The file must be pre-formatted to the JClass Chart Standard (see [Text Data Formats](#), in Chapter 4). Sample data files are located in the `JCLASS_HOME/examples/intro/chart2.dat` directory.

#### Editing the Default Data

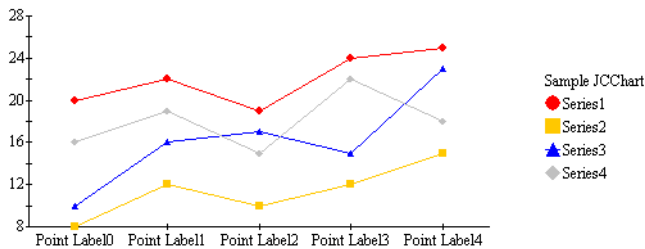
You can use the data provided in the editor as is, or you can modify it. To use existing data, just check the **Edit data in the text area** radio button, and click **Done**. Change

data by deleting and inserting text in the area provided. Be careful to preserve the punctuation surrounding the original text:

```
# The data is in array format
# It has one set of x-axis data, multiple sets of y-axis data
# It can have names for the data view and for the x and y series
# Each x point can have a point label
ARRAY 'Sample JCChart' 4 5
'Point Label0' 'Point Label1' 'Point Label2' 'Point Label3' 'Point Label4'
'X Series' 0.0 1.0 2.0 3.0 4.0
'Series1' 20.0 22.0 19.0 24.0 25.0
'Series2' 8.0 12.0 10.0 12.0 15.0
```

The chart below shows how the default data appears as a plot. Notice where the different elements are positioned. Each point on the X-axis is labelled with the names specified in the default data. The name of each series of y-values appears in the legend. The name of the data view is positioned directly above the legend.

In order for the default data to display this way, you must first set the `xAxisAnnotation` property to `Point_Labels`, and the `legendVisible` property to `true`.



### 13.3.2 SimpleChart: Using Swing TableModel Data Objects

Your (Swing) application may have the data you want to chart contained in a `Swing TableModel`-type data object. You can use this object as your data source instead of using the `JClass Chart` built-in data sources if your IDE supports a `TableModel` editor.

Use the `SwingDataModel` property to specify an already-created `Swing TableModel` object to use as the chart's data source.



# SimpleChart Bean Tutorial

*Introduction to JavaBeans* ■ *SimpleChart Bean Tutorial*

## 14.1 Introduction to JavaBeans

JClass Chart components are JavaBean-compliant. The JavaBeans specification makes it very easy for a Java Integrated Development Environment (IDE) to “discover” the set of properties belonging to an object. The developer can then manipulate the properties of the object easily through the graphical interface of the IDE when constructing a program.

The three main characteristics of a Bean are:

- the set of properties it exposes
- the set of methods it allows other components to call; and
- the set of events it fires

Properties control the appearance and behavior of the Bean. Bean methods can also be called from other components. Beans fire events to notify other components that an action has happened.

### 14.1.1 Properties

“Properties” are the named method attributes of a class that can affect its appearance or behavior. Properties that are readable have a “get” (or “is” for booleans) method, which enables the developer to read a property’s value, and those properties that are writable have a “set” method, which enables a property’s value to be changed.

For example, the `JCAxis` object in JClass Chart has a property called `AnnotationMethod`. This property is used to control how an axis is labelled. To set the property value, the `setAnnotationMethod()` method is used. To get the property value, the `getAnnotationMethod()` method is used.

For complete details on how object properties are organized, see [JClass Chart Object Containment](#) and [Setting and Getting Object Properties](#), in Chapter 1.

#### Setting Bean Properties at Design-Time

One of the features of any JavaBean component is that it can be manipulated interactively in a visual design tool (such as a commercial Java IDE) to set the initial property values

when the application starts. Consult the IDE documentation for details on how to load third-party Bean components into the IDE.

You can also refer to the “JClass and Your IDE” chapter in the [JClass Desktop Views Installation Guide](#).

Most IDEs list a component’s properties in a property sheet or dialog. Simply find the property you want to set in this list and edit its value. Again, consult the IDE’s documentation for complete details.

## 14.2 SimpleChart Bean Tutorial

This tutorial guides you through the development of an application that uses `SimpleChart` to chart the financial information of “Michelle’s Microchips”. It is a good starting point for learning basic JClass Chart features. To explore more advanced features of JClass Chart, however, we recommend that you use the [MultiChart Bean](#) bean.

The tutorial does not cover all of the properties available in `SimpleChart`. For a complete reference, see [JClass Chart Beans](#), in Chapter 13. The screen captures have all been taken from Sun’s BeanBox and will differ slightly from your IDE’s appearance.

### 14.2.1 Steps in this Tutorial

This tutorial has eight steps:

1. Create a new application in your IDE and add a container.
2. Put a `SimpleChart` object into the container.
3. Load the data for Michelle’s Microchips.
4. Add a header, footer, and legend.
5. Add point labels to the X-axis.
6. Change the background color to white.
7. Set the chart type to bar, and add 3D effects.
8. Compile and run the application.

#### Step 1: Create the ‘Michelle’ Application

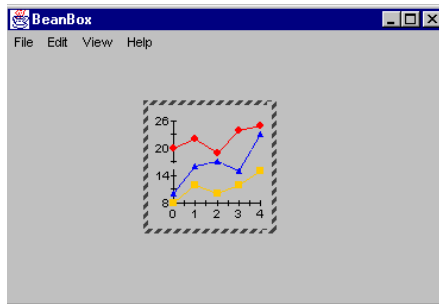
Create a new application in your IDE and add a container to hold a `SimpleChart` object. In most IDEs this will be a panel. See your IDE’s documentation for instructions on creating a basic application and adding a container.

## Step 2: Put a Chart Object into the Container

With the container displayed in design mode, click the `SimpleChart` icon and place a `SimpleChart` object into the container's area. See your IDE's documentation for details on placing objects into a container. The `SimpleChart` icon looks like this:



In your container object, you should now see a basic chart area with an X- and Y-axis, like this:



If you open your property list (the window that displays the Bean's properties) with the `SimpleChart` area selected, you should see the property editors that are available in `SimpleChart`.

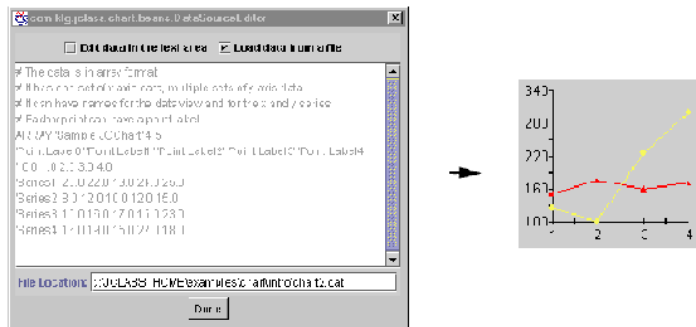
## Step 3: Load Data from a File

This tutorial uses data from a file named *chart2.dat* contained in the `JCLASS_HOME/examples/chart/intro/chart2.dat` directory. To load *chart2.dat* into `SimpleChart`, bring up the custom data source editor by clicking on the `data` property:

data

The data source editor provides two methods for loading data: editing data in the text area, or loading data from a file. For Michelle's Microchips, click the **Load data from a**

**file** radio button. Then, enter the full path name of *chart2.dat* in the **File Location** field. After you click **Done**, you should see the data displayed in the chart area as follows:



### What's in *chart2.dat*?

*chart2.dat* has financial information for Michelle's Microchips, formatted for the file data source method of data loading. SimpleChart accepts only *.dat* files, or modifications to the default data in the editor. For more information on creating a file data source, see [Loading Data from a File](#), in Chapter 4.

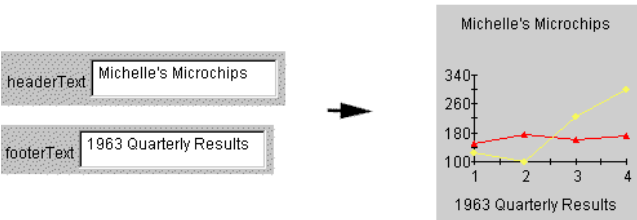
The content of *chart2.dat* is:

```
ARRAY " 2 4
'Q1' 'Q2' 'Q3' 'Q4'
" 1.0 2.0 3.0 4.0
'Expenses' 150.0 175.0 160.0 170.0
'Revenue' 125.0 100.0 225.0 300.0
```

JClass Chart also has other Beans which allow you to chart data from a database easily. See [JClass Chart Beans](#), in Chapter 13, for more information.

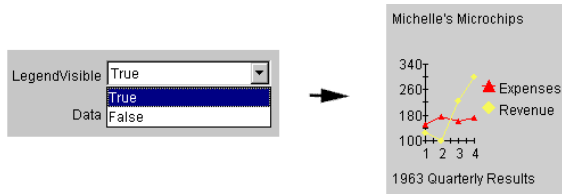
### Step 4: Add a Header, Footer, and Legend

Enter “Michelle's Microchips” in the headerText property editor and “1963 Quarterly Results” in the footerText property editor:





To add the legend, set the `legendVisible` property to `true`. The legend text is taken from information in the data source. Notice how the plot area is resized to accommodate the legend. You may have to resize your chart area to accommodate the changes:

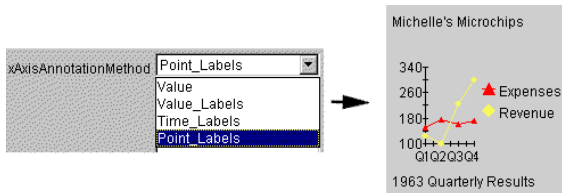


For more information on legend properties, see [Legends](#), in Chapter 13.

### Step 5: Add Point Labels to the X-axis

By default, `SimpleChart` annotates the axes with values. You can change the annotation to show point labels or time labels.

For Michelle's Microchips, change the X-axis annotation from values to point labels. Do this by setting the `xAxisAnnotationMethod` property to `Point_Labels`:



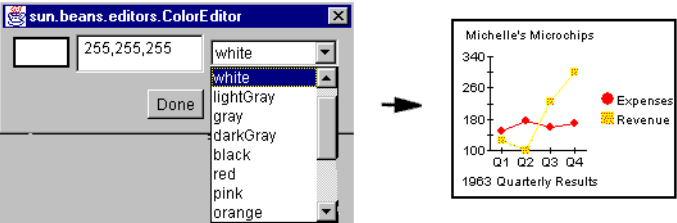
You should now see “Q1”, “Q2”, “Q3”, and “Q4” on the X-axis. These labels are contained in the *chart2.dat* file, and come up automatically when `Point_Labels` is selected. For more information on axis annotation, see [Axis Properties](#), in Chapter 13.

### Step 6: Change the Background Color

To change the background color to white, click the `background` property to bring up your color editor:

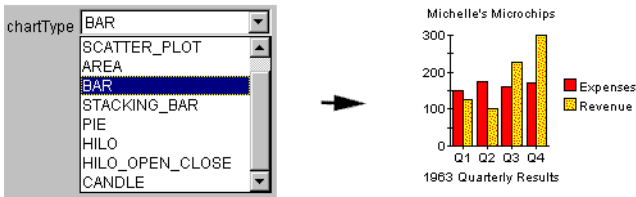


The custom color editor used by your IDE will differ from the BeanBox. Select pure white from the options on your color editor:



### Step 7: Change to Bar Chart and add 3D Effects

You can select from 13 chart types using the `chartType` property editor (see [Chart Types](#), in Chapter 13, for a complete list). For Michelle's Microchips, select the `BAR` type:



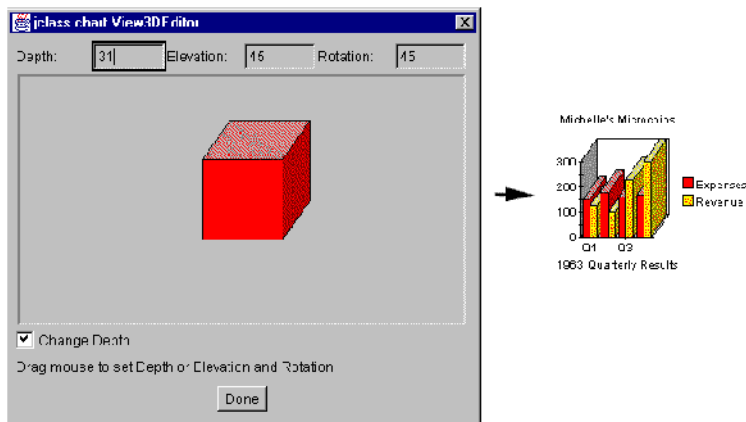
To add three-dimensional visuals to your chart, click the `view3D` property to bring up the **View3DEditor**:



There are two main settings in the **View3DEditor** (below): depth, and combined elevation and rotation. They are both set either by dragging the box in the editor with a mouse or by typing in the value in the editable box next to these settings.

First, drag the square with your mouse until you have an Elevation of 45 and a Rotation of 45, or simply type "45" in the editable box next to these settings. Second, check the

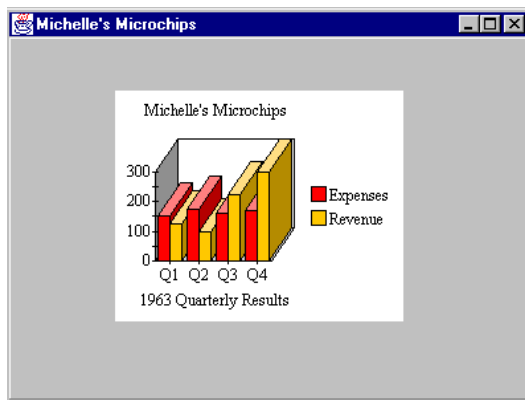
**Change Depth** box, and drag the red square until it has a depth of 31, or simply type “31” in the editable box next to Depth. Click **Done** to set the changes:



### Step 8: Compile and Run the Application

For the last step, compile and run the application. See your IDE’s documentation for details. When you run the application, you should have a window with a chart, displaying Michelle’s Microchips’ financial information.

The following example illustrates how the application appears when run:





## MultiChart Bean

[Introduction to MultiChart](#) ■ [Getting Started with MultiChart](#) ■ [MultiChart Property Reference](#)

### 15.1 Introduction to MultiChart

MultiChart is the next generation charting Bean from JClass Chart. It contains a richer set of features than previous Beans, highlighting the superiority of JClass Chart as a charting application tool.

The MultiChart icon:

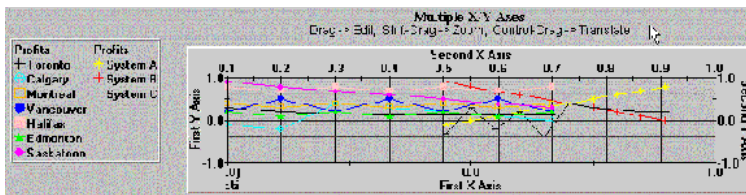


#### Highlights of the MultiChart Bean

- Handles multiple data sources.
- Plots data against multiple X- and Y-axes.
- Fully customizable axes.
- Extensive control of font, colors, borders, and styles for each chart element.

#### 15.1.1 Multiple Axes

MultiChart can have two X- and two Y-axes, as in the example below:



### Setting Properties on Multiple Axes

Axis properties can be set for each axis individually. At the top of each axis editor you will see four radio buttons:



When a radio button is selected, all that follows below will apply to that axis.

### 15.1.2 Multiple Data Views

MultiChart allows you to load data from two different sources at the same time. When loading data from two different sources, they are each assigned to a separate data view.



By default, both data views are showing, but you can hide or reveal data views depending on your application's needs. Both sets of data can be mapped to the same set of X- and Y-axes, or, mapped to different axes.

**Note:** Radar, area radar, and pie charts do not support multiple data views. For more information on data views, see [Understanding the Underlying Data Model](#), in Chapter 4.

### 15.1.3 Intelligent Defaults

MultiChart has a sophisticated set of dynamic default settings in the custom property editors. You can override these defaults to suit your needs. When you override a default value in a text editor, it becomes static, and will not automatically adjust anymore.

#### Returning to Default Values

If you want to return to default settings in the custom editors after overriding them, all you have to do is delete the contents of the changed field, and leave it blank. The next time you bring the editor you will see that the automatic values have returned.

## 15.2 Getting Started with MultiChart

MultiChart has a sophisticated set of dynamic default settings that adjust to your data and other settings. This means that you only have to make a minimum of settings to have a respectable chart. The following list describes the most common start-up tasks and the editors used for them:

- **Load Data.** To load data in the chart, use the [DataSource](#) editor. This editor allows you to load data from one or two sources. There is also a default set of data built-in that you can use to experiment with. Alternately, you can use a `Swing TableModel` data object as the chart's data source using the `SwingDataModel` property.

- **Select Chart Types.** For each data view, you can select a chart type and the axes that the data will be plotted against with the [DataChart](#) editor.
- **Set BackGround Color.** Use [ChartAppearance](#) to set the color of the chart background.
- **Set Axis Annotation.** By default, `MultiChart` uses values to annotate the axes. You can also use value labels, point labels, or time labels by setting the annotation type with the [AxisAnnotation](#) editor.
- **Add a Legend.** Add a legend by checking the Visible box in the [LegendAppearance](#) editor.
- **Add a Header and Footer.** To add a header, use [HeaderText](#) to add the text, and then select the Visible check box in [HeaderAppearance](#). The footer is the same, but uses the [FooterText](#), and [FooterAppearance](#) editors
- **Add Extra Axes.** By default a standard X-Y axis set is displayed. If you require, you can display a second X- or Y-axis. Display them with the [AxisMisc](#) editor's **Visible** property. Then use the many axis editors, such as [AxisPlacement](#), to set up and align the axes.

## 15.3 MultiChart Property Reference

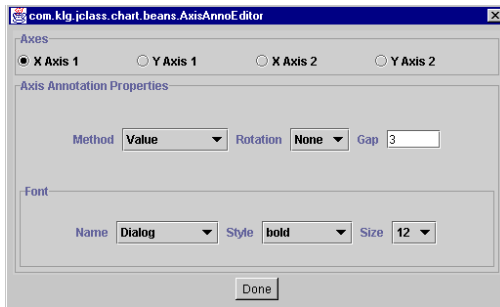
The following property reference section covers all of `MultiChart`'s features.

### 15.3.1 Axis Controls

This group of editors sets up the axes. `MultiChart` has a sophisticated set of automatic default values that adjust to your data. This makes chart development fast and easy. `MultiChart` is also extremely flexible, and every aspect of the axes can be adjusted.

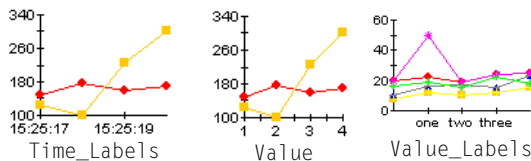
#### **AxisAnnotation**

With the `AxisAnnotation` editor, you can set the annotation type for each axis, and control how they look. Axis annotations are numbers or text that appear along the axes. Options in the **Method** menu are: `Value`, `Time_Labels`, `Point_Labels`, and `Value_Labels`.

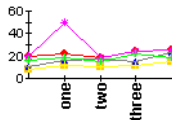


For each of the labelling methods, there is a corresponding editor that gives you more control over the behavior and appearance. For Value, use [AxisScale](#), for Point\_Labels, use [AxisPointLabels](#), for Time\_Labels, use [AxisTimeLabels](#), and for Value\_Labels, use [AxisValueLabels](#).

The following examples illustrate the different label types:



With the **Rotation** property, you can rotate the labels on the axis. The following example shows Value\_Labels, rotated by 270 degrees and with bold, 12pt font:

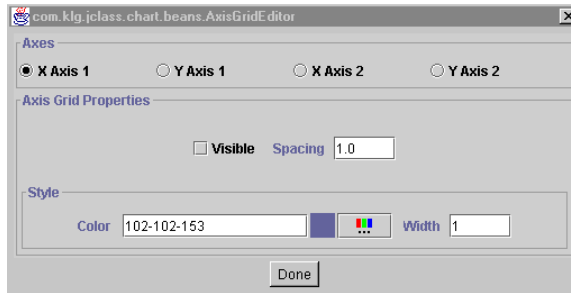


**Gap** controls the space between annotations, in pixels. If, for example, you used point labels, you could use the **Gap** property to make sure they have enough room to display properly.

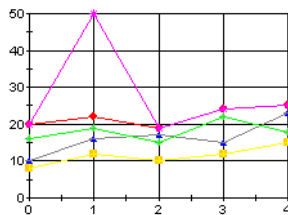
### AxisGrid

Use the [AxisGrid](#) editor to set up gridlines on each of the axes. There are also controls for color, line spacing, and line width of the gridlines.





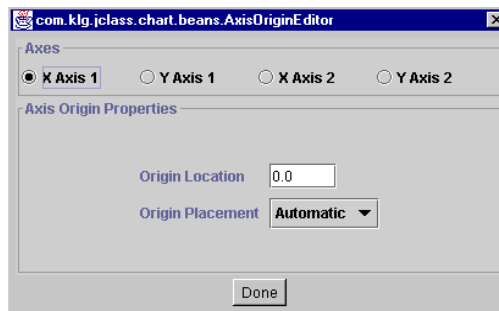
The following example sets X Axis 1 grid and Y Axis 1 grid to **Visible** with **Spacing** = 1 and **Width** = 1 for the X Axis, and with **Spacing** = 1 and **Width** = 10 for the Y Axis:



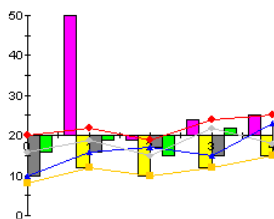
## AxisOrigin

The `AxisOrigin` editor allows you to specify an origin by coordinates, or by choosing an option from a pull down menu. By default, axes origins are set automatically, based on the available data.

To place the origin, you can select one of the locations from the pull-down menu, such as Min or Max. If you want to set the origin to a specific value on the axis, select `Value_Anchored` from the menu and then enter the value in the **Origin** field:



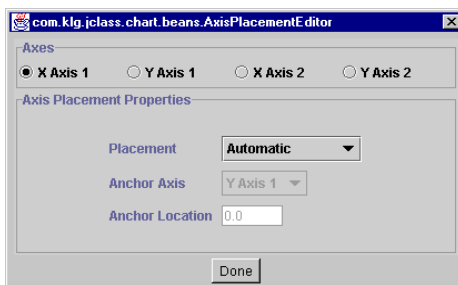
The following example anchors the origin of Y Axis 1 at 20 (default data):



Note that, by default, X Axis 1 is placed at the origin of Y Axis 1. To override this default, use the [AxisPlacement](#) editor.

### AxisPlacement

Axis placement determines the placement of an axis in relation to another. By default, this is set automatically by MultiChart, based on the given data. Sometimes, however, you may want to locate an axis in a different location.

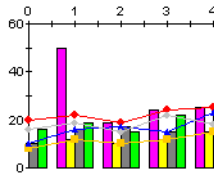


Using the **Placement** field, select the type of placement for the axis selected. Placement options include: Min, Max, Automatic, Origin, and Value\_Anchored.

The **Axis** field selects the anchor-axis that you want to place the current axis against (for example, place X Axis 1 in relation to Y Axis 2). If you select None as an Axis, MultiChart will use the default axis.

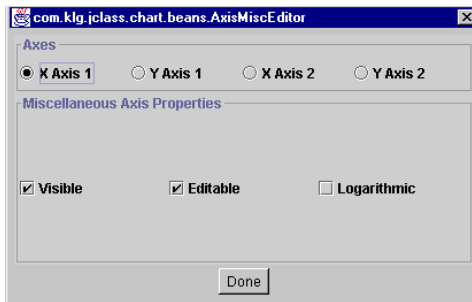
To place the axis at a specific value along another axis, select Value\_Anchored from the pull-down menu, and enter the value in the **Location** field.

The following example shows X Axis 1, with a **Placement** of Max in relation to Y Axis 1:

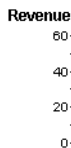


### AxisMisc

Use `AxisMisc` to show or hide any of the axes. It also allows you to make any axis logarithmic. The `Editable` property, when selected, allows zooming, editing, and translation for the selected axis. For more information on interactive events, see Section 15.3.6, [Event Controls](#).



The following example hides X Axis 1 from view by deselecting `Visible`.



### AxisPointLabels

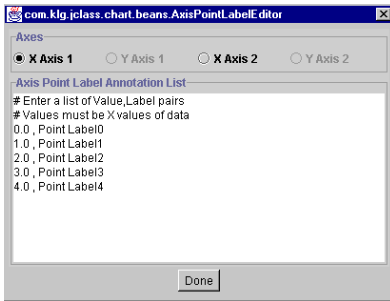
Use the `AxisPointLabels` editor to create point labels (applies to X1 and X2 axes only). Point labels label specific points of data on the X-axes.

The editor reads data from the data source associated with the selected axis and provides a list of point labels. To change the text in these labels, change the text alongside the point. Note that the format is “point value then comma then the name of the label”. For example,

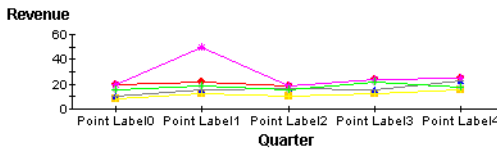
3.0, PointLabel3

In order for the labels to appear on the chart, you also have to set the annotation method to `Point_Labels` in the `AxisAnnotation` editor.

See below for an example.



The following example shows how the default data's point labels appear on X Axis 1:



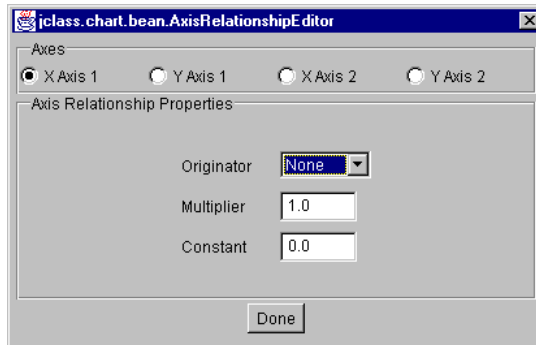
Note that if you are mapping multiple data sources against a single axis, then you will want to use value labels instead, as the `AxisPointLabels` editor only uses points from the first data source associated with the selected axis.

### AxisRelationships

The `AxisRelationship` editor allows you to create a mathematical relationship between two axes. For example, if you want to create a thermometer chart with Celsius values on the left and the Fahrenheit values on the right, you could create a Celsius axis, and then base the Fahrenheit axis values on it.

There are three properties included in this calculation: **Originator**, **Multiplier**, and **Constant**. The calculation is based on the formula:

$$\text{New Axis Value} = \text{Constant} + \text{Multiplier} \times \text{Originator}.$$



To use this editor, first click on the radio button next to the Axis that you want to alter. Next, select an axis from the **Originator** menu that your calculation will be based on, and then enter a value in the **Multiplier** field that represents the relationship. The **Constant** value is optional; its default value is 0.0.

### AxisScale

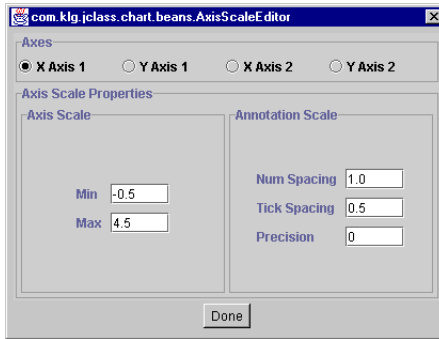
The `AxisScale` editor controls the range on each axis, the interval of the numbering, and **Tick Spacing**. It is used primarily for the Value method of axis annotation (see the [AxisAnnotation](#)). **Precision** determines the numeric precision of the axis numbering.

The effect of Precision depends on whether it is positive or negative:

- *Positive* values add that number of places after the decimal place. For example, a value of 2 displays an annotation of 10 as “10.00”.
- *Negative* values indicate the minimum number of zeros to use before the decimal place. For example, a value of -2 displays annotation in multiples of 100.

The default value of Precision is calculated from the data supplied.

The Min and Max fields determine the range of data that is displayed on the chart. There are intelligent defaults in this editor that adjust to your data and other chart settings. You can override these settings with the fields provided.

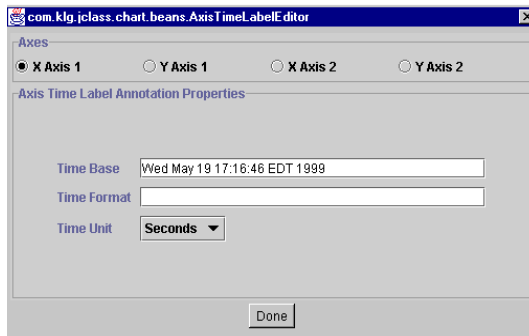


### AxisTimeLabels

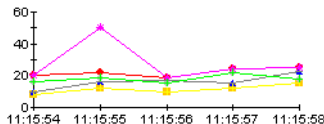
The `AxisTimeLabel` editor allows you to control how the time labels appear. When you select the annotation method with `AxisAnnotations`, you can select time labels, which represent the values on the axis as units of time.

**Time Base** determines the date and time that the labelling starts from (default is current time/date). **Time Unit** is the unit of time the labels use, such as year, month, day, minute, second, and so on. The default time unit is minutes.

**Time Format** allows you to customize the text in the time labels with a set of formatting codes. See [Axis Labelling and Annotation Methods](#), in Chapter 6, for a list of these codes.

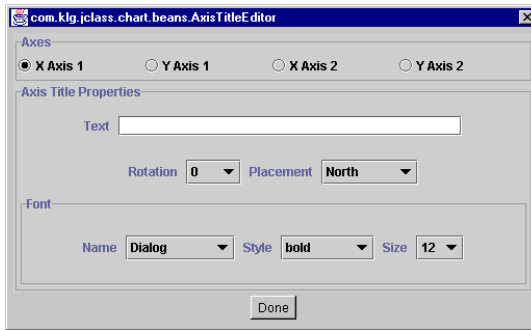


The following example uses time labelling on X Axis 1, with seconds as the time unit:

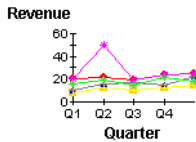


## AxisTitle

Using the `AxisTitle` editor, you can add axis titles to each axis. There are also settings for the font, point, rotation, and placement of the title.

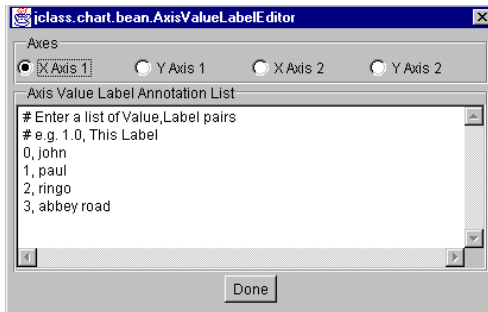


In the **Placement** field's pull-down menu are a list of compass directions for title placement. Not all options are available to X- and Y-axes. If you select a placement, and it returns to the previous selection, that placement is not available for that axis. The following image shows the effects of adding titles to X Axis 1 and Y Axis 1, and setting the font to bold, with a size of 12:



## AxisValueLabels

Use the `AxisValueLabel` editor to enter value labels for the axes. Value labels appear along the axis at specified values. You also have to set the annotation method to **Value\_Labels**, in the `AxisAnnotation` editor before the labels will display.



To add value labels, enter the value, followed by a comma and a label (see above). The following example shows how the labels in the editor above appear on X Axis 1.



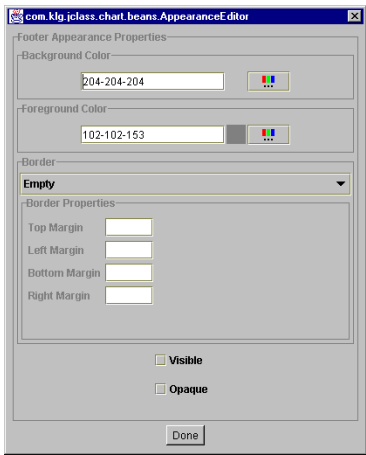
### 15.3.2 Headers, Footers, and Legends

#### FooterText

The `FooterText` editor allows you to enter text that will appear at the bottom of the chart area. You can also select a font, font style, and size for the footer.

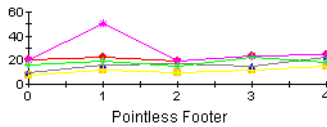


Note that the footer will not display unless you check the **Visible** box, in the [FooterAppearance](#) editor (this editor also controls footer opacity, background, and foreground).



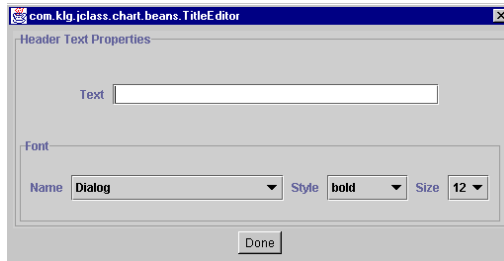


The following example shows how a ‘pointless footer’ appears on the chart area:

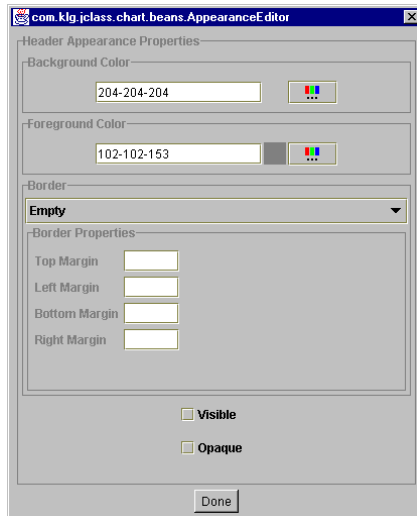


## HeaderText

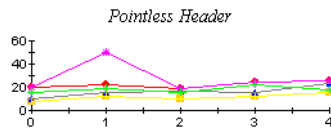
The `HeaderText` editor allows you to enter header text, that will appear at the top of the chart area. You can also select a font, font style, and size of the header.



Note that the header will not display unless you check the **Visible** box, in the [HeaderAppearance](#) editor (which also controls header opacity, background and foreground).



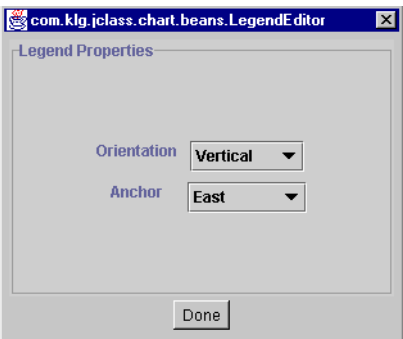
The following example shows how a ‘pointless header’ displays on the chart:



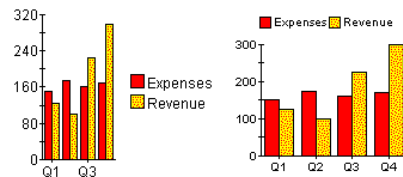
### LegendLayout

The `LegendLayout` editor controls the layout of the legends. **Orientation** determines how the legend items are placed in the legend (either vertically or horizontally). The **Anchor** property positions the entire legend on the chart, based on compass directions.

In order for the legend to display on your chart, the **Visible** checkbox in the [LegendAppearance](#) editor must be selected.



Below are two examples of legend layout:



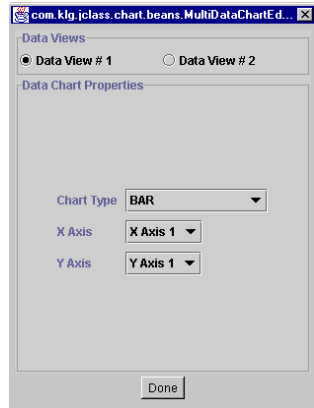
The example on the left uses the default settings with **Anchor** = East and **Orientation** = Vertical. In the example on the right, **Anchor** = North and **Orientation** = Horizontal.

### 15.3.3 Data Source and Data View Controls

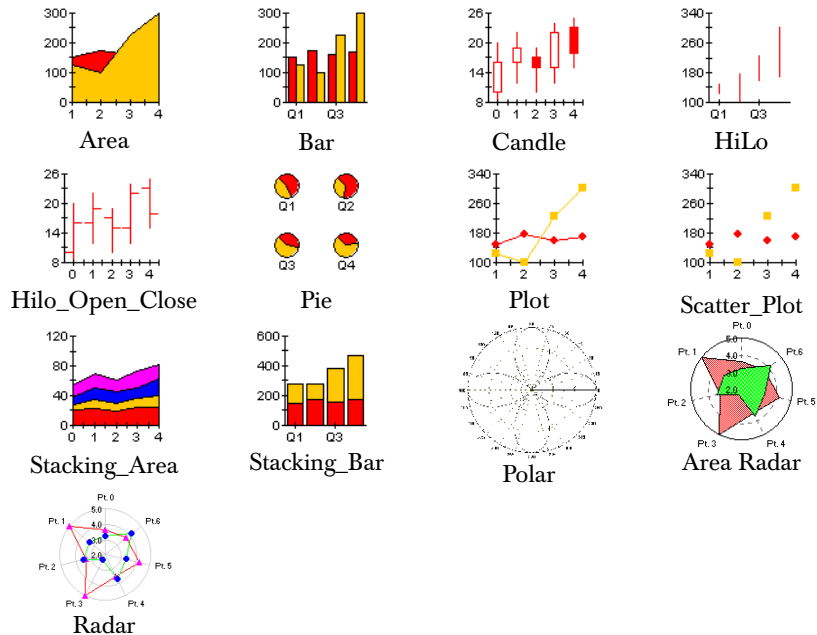
This group of editors manages the properties that control the data source, and the views on the data. `MultiChart` can load data from two different sources. Each of the data sources is assigned to a data view.

## DataChart

The DataChart editor allows you to select the chart type of each data view, and which axes each data view will be mapped against.

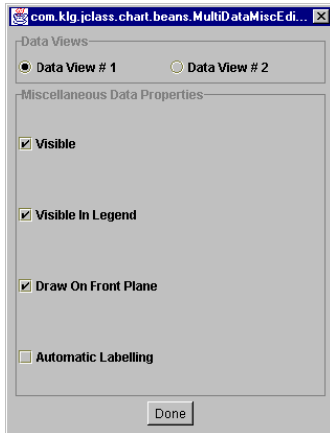


The ChartType property selects from the following chart types:



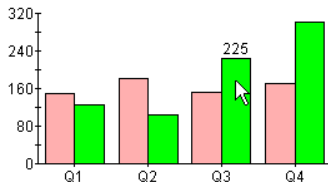
## DataMisc

The DataMisc editor controls several aspects of the data views.



With the **Visible** property, you can show or hide each data view from the display area. **Visible In Legend** will show/hide a data view from the legend (but the data will still be charted).

**Automatic Labelling** attaches a dwell label to every data point in the chart. A dwell label is an interactive label that shows the value of a point, bar or slice, when a user's mouse moves over it. In the example below, '225' appears on top of the green bar as the cursor passes over it, indicating that the value of the bar is 225.



When **Draw on Front Plane** is selected, the data view will be mapped on the front plane of a three-dimensional chart space. Applies only in cases where there are multiple data series, displayed on multiple axes, using 3D effects.

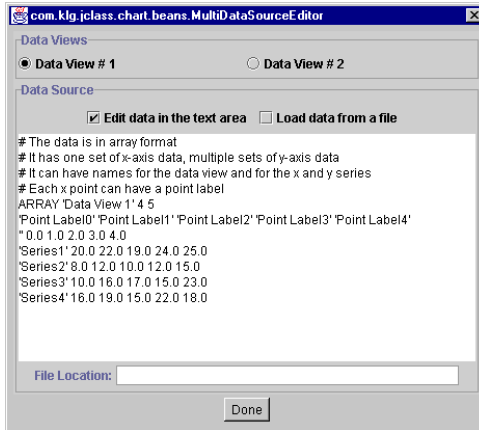
## DataSource

There are three ways of loading data with the MultiChart Bean. Two are handled by this property: from a *.dat* file, or by entering data directly into the custom editor. Both methods are managed by the DataSource editor.

The third method is to use a Swing `TableModel`-type data object as a data source, instead of using the JClass Chart built-in data source. See `SwingDataModel` below for details.

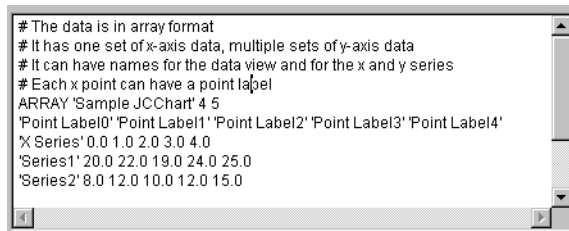
The first step is to select a data view with one of the radio buttons. Then, follow the procedure below for each data view.

To load data from a file into a data view, click **Load data from a file**, enter the name of the file in the **File Location** field, and click **Done**:



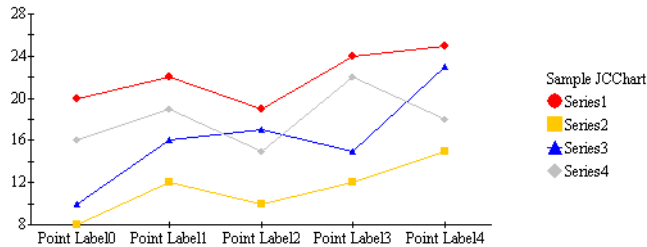
Specify the full path of the file. The file must be pre-formatted to the JClass Chart Standard (see [Text Data Formats](#), in Chapter 4). Sample data files are located in the *JCLASS\_HOME/jclass/chart/examples* directory.

You can use the data provided in the editor, as is, or you can modify it. To use existing data, just check the **Edit data in the text area** radio button, and click **Done**. Change data by deleting and inserting text in the area provided. Be careful to preserve the punctuation surrounding the original text:



The chart below shows how the default data for Data View 1 appears as a plot. Notice where the different elements are positioned. Each point on the X-axis is labelled with the names specified in the default data. The name of each series of y-values appears in the legend. The name of the data view is positioned directly above the legend.

In order for the default data to display this way, you must first set the `xAxisAnnotation` property to `Point_Labels`, and the `legendVisible` property to `true`.



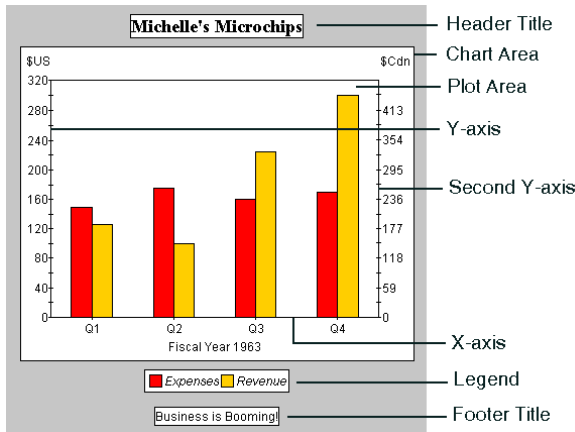
### SwingDataModel

Instead of using the chart's internal data source, you can use a `Swing TableModel`-type data object that you have already created for your application, if your IDE supports an editor for `TableModel`. This saves reformatting your data to match the format used by `JClass Chart`.

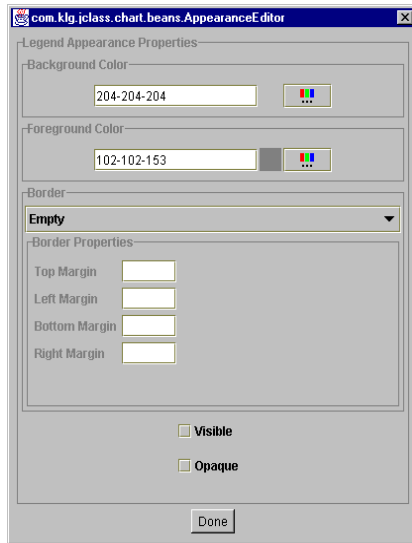
Use the `SwingDataModel1` property to specify an already-created `Swing TableModel` object to use as the data source for the first data view. Use `SwingDataModel2` to specify a `TableModel` object to use for the chart's second data view.

## 15.3.4 Appearance Controls

This group of editors allows you to control the look of specific chart subcomponents. You can control font, borders, background, and foreground for the chart, chart area, plot area, header, footer, and legend. The following diagram illustrates the different chart subcomponents.



All of the editors have the same basic functionality that apply to a specific chart subcomponent, as follows:



Small differences in each editor will be discussed below. Note that for most of the appearance editors, there are corresponding editors for controlling other properties of that chart element.

### ChartAppearance

The `ChartAppearance` editor sets the foreground/background border, and opaque values for the chart. This editor affects the areas behind all other chart elements.

### ChartAreaAppearance

The `ChartAreaAppearance` editor sets the foreground/background border, visible, and opacity values for the chart area (see diagram above).

### FooterAppearance

The `FooterAppearance` editor sets the foreground/background border, visible, and opaque values for the footer. When **Visible** is checked, the footer will be displayed in the chart. By default the footer is not showing. The `FooterAppearance` editor works in conjunction with the [FooterText](#) editor, which is used to enter the footer text.

### HeaderAppearance

The `HeaderAppearance` editor sets the foreground/background border, visible, and opaque values for the header. When **Visible** is checked, a header will be displayed in the chart. By default the header is not showing. This editor works in conjunction with the [HeaderText](#) editor.

## LegendAppearance

The `LegendAppearance` editor sets the foreground/background border, visible, and opaque values for the legend and determines if it is displayed. By default, the legend will not appear. When **Visible** is checked, a legend will be displayed in the chart.

The content of the legend comes from the information in the data source. In order to change the contents of the legend, you have to change what is in the data source. For information on how to set up legend items in the data source, see [Text Data Formats](#), in Chapter 4.

Other legend settings are found in the `LegendLayout` editor.

## PlotAreaAppearance

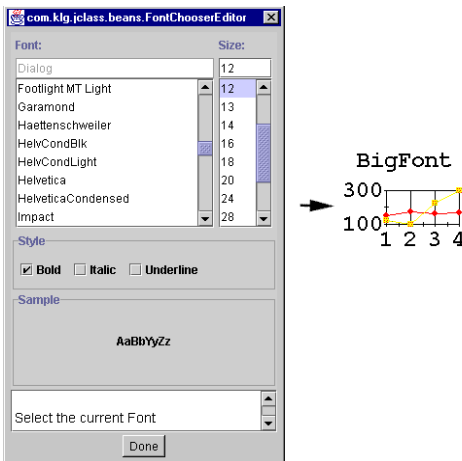
The `PlotAreaAppearance` editor sets the foreground and background for the plot area, and allows you to add an **Axis Bounding Box**. A bounding box is a graphical feature that closes off the axes, thus forming a square.



## Font

The `Font` editor sets the font defaults for your chart.

The font you choose will apply to all text on the chart simultaneously. The following example sets the font to **Courier, Bold, 24 point**:



This font editor sets up a default font for the chart (not including the header and footer). You can, however, change font for selected elements using custom editors for each

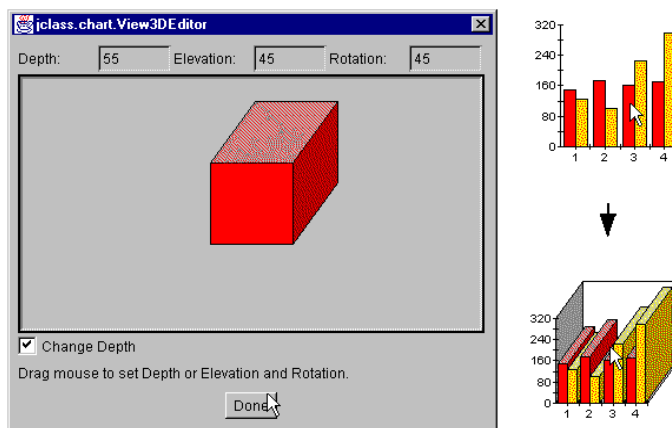


property. For example, the [HeaderText](#), [FooterText](#), and [AxisAnnotation](#) editors allow you to override the default font settings.

### 15.3.5 View3D

To add 3D effects to your chart, click the `View3D` property.

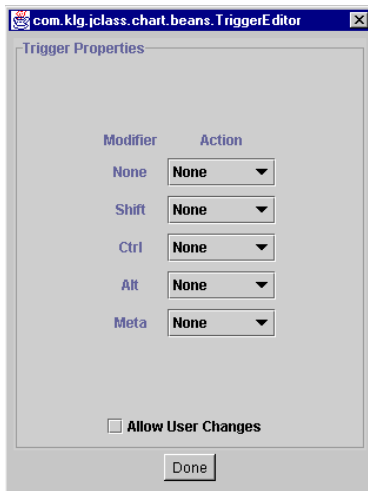
First drag the red square in the editor until it has the desired Elevation and Rotation. Then, check the **Change Depth** option box, and drag the red square until it has the Depth you want to see on your chart. The degree of depth, elevation and rotation is displayed in numbers at the top of the editor. Click **Done** to set the changes:



### 15.3.6 Event Controls

#### TriggerList

The `TriggerList` editor sets up what user events the chart will handle, either from a mouse, or mouse-keyboard combination.



**Actions** are the available event mechanisms, such as Zoom, Rotate, Depth, Customize, Pick, and Translate. By setting up these triggers, the end-user can examine data more closely or visually isolate part of the chart. The following list describes these interactions:

- **Translate** allows moving of the chart.
- **Zoom** allows zooming into or out from the chart.
- **Rotate** allows rotation (only for bar or pie charts displaying a 3D effect).
- **Depth** allows adding depth cues to the chart (only for bar or pie charts displaying a 3D effect).
- **Customize** allows the user to launch the chart Customizer. To use this feature, you must also check the **Allow User Changes** box.
- **Pick** allows you to set up pick events. The `pick` method is used to retrieve an  $x,y$  coordinate in a Chart from user input and then translate that into the data point nearest to it. This feature requires some non-bean programming. See [Using Pick and Unpick](#), in Chapter 9, for more details.

A **Modifier** is a keyboard event that can ‘modify’ a mouse click.

It is also possible in most cases for the user to reset the chart to its original display parameters. The interactions described here affect the chart displayed inside the `ChartArea`; other chart elements, like the header, are not affected.

*Part* ***III***

*Reference*  
*Appendices*



# Summary of Properties for JClass Chart Objects

[ChartDataView](#) ■ [ChartDataViewSeries](#) ■ [ChartText](#)  
[JCAnno](#) ■ [JCAreaChartFormat](#) ■ [JCAxis](#) ■ [JCAxisFormula](#) ■ [JCAxisTitle](#)  
[JCBarChartFormat](#) ■ [JCCandleChartFormat](#) ■ [JCChart](#) ■ [JCChartArea](#) ■ [JCChartLabel](#)  
[JCChartLabelManager](#) ■ [JCChartStyle](#) ■ [JCFillStyle](#) ■ [JCGrid](#) ■ [JCGridLegend](#)  
[JCHiloChartFormat](#) ■ [JCHLOCChartFormat](#) ■ [JCLegend](#) ■ [JCLineStyle](#) ■ [JCMarker](#)  
[JCMultiColLegend](#) ■ [JCPieChartFormat](#) ■ [JCPolarRadarChartFormat](#)  
[JCSymbolStyle](#) ■ [JCThreshold](#) ■ [JCValueLabel](#) ■ [PlotArea](#) ■ [SimpleChart](#)

This appendix summarizes the JClass Chart properties for all commonly-used classes in alphabetical order.

## A.1 ChartDataView

Name	Description
AutoLabel	The AutoLabel property determines if the chart automatically generates labels for each point in each series. The default is false. The labels are stored in the AutoLabelList property. They are created using the Label property of each series.
Batched	The Batched property controls whether the ChartDataView is notified immediately of data source changes, or if the changes are accumulated and sent at a later date.
BufferPlotData	The BufferPlotData property controls whether plot data is to be buffered to speed up the drawing process. This property is applicable for Plot, Scatter, Area, Hilo, HLOC, and Candle chart types only. Normally it is true. The property is ignored if the FastUpdate property is true. Plot data will be buffered for FastUpdate.

Name	Description
Changed	The <b>Changed</b> property manages whether the data view requires recalculation. If set to <code>true</code> , a recalculation may be triggered. Default value is <code>true</code> .
ChartFormat	The <b>ChartFormat</b> property represents an instance of <code>JCAreaChartFormat</code> , <code>JCBarChartFormat</code> , <code>JCCandleChartFormat</code> , <code>JCHiloChartFormat</code> , <code>JCHLOCCChartFormat</code> , or <code>JCPieChartFormat</code> , depending on the current chart type.
ChartStyle	The <b>ChartStyle</b> property contains all the <code>JCChartStyles</code> for the data series in this data view. Default value is generated.
ChartType	The <b>ChartType</b> property of the <code>ChartData</code> object specifies the type of chart used to plot the data. Valid values are: <code>JCChart.AREA</code> , <code>JCChart.AREA_RADAR</code> , <code>JCChart.BAR</code> , <code>JCChart.CANDLE</code> , <code>JCChart.HILO</code> , <code>JCChart.HILO_OPEN_CLOSE</code> , <code>JCChart.PIE</code> , <code>JCChart.PLOT</code> (default), <code>JCChart.POLAR</code> , <code>JCChart.RADAR</code> , <code>JCChart.SCATTER_PLOT</code> , <code>JCChart.STACKING_AREA</code> , and <code>JCChart.STACKING_BAR</code> .
ColorHandler	The <b>ColorHandler</b> property specifies a class used to override the default color determination. The <b>ColorHandler</b> property must implement <code>JCDrawableColorHandler</code> .
DataSource	The <b>DataSource</b> property, if non-null, is used as a source for data in the <code>ChartDataView</code> . The object must implement <code>ChartDataModel</code> .
DrawFrontPlane	The <b>DrawFrontPlane</b> property determines whether a data view that has both axes on the front plane of a 3d chart will draw on the front or back plane of that chart. If <code>true</code> , it will draw on the front plane; if <code>false</code> it will draw on the back plane. If either axis associated with the data view is on the back plane, this property will be ignored and the data view will automatically be drawn on the back plane. This property is also ignored for 3d chart types such as bar and stacking bars that automatically appear on the front plane.
DrawingOrder	The <b>DrawingOrder</b> property determines the drawing order of items. When the <b>DrawingOrder</b> property is changed, the order properties of all <code>ChartDataView</code> instances managed by a single <code>JCChart</code> object are normalized.
FastUpdate	The <b>FastUpdate</b> property controls whether column appends to the data are performed quickly, only recalculating and redrawing the newly-appended data.

Name	Description
HoleValue	The HoleValue property is a floating point number used to represent a hole in the data. Internally, ChartDataView places this value in the x- and y-arrays to represent a missing data value. Note that if the HoleValue is changed, values in the x- and y-data previously set with HoleValues will <b>not</b> change their values but will now draw.
Inverted	If the Inverted property is set to true for rectangular charts, the x-axis becomes vertical, and the y-axis becomes horizontal. Default value is false. Note: This property is ignored for circular charts.
Markers	The Markers property associates a list of markers with the data view.
Name	The Name property is used as an index for referencing particular ChartDataView objects.
NumPointLabels	The NumPointLabels property determines the number of labels in the PointLabels property. The PointLabels property is an indexed property consisting of a series of Strings representing the desired label for a particular data point.
NumSeries	The NumSeries property determines how many data series there are in a ChartDataView.
OutlineColor	The OutlineColor property determines the color with which to draw the outline around a filled chart item (e.g. bar, pie).
PickFocus	The PickFocus property specifies how distance is determined for pick operations. When set to PICK_FOCUS_XY, a pick operation will use the actual distance between the point and the drawn data. When set to values of PICK_FOCUS_X or PICK_FOCUS_Y, the distance only along the x-axis or y-axis is used.
PointLabel	Sets a particular PointLabel from the PointLabels property (see below).
PointLabels	The PointLabels property is an indexed property comprising a series of Strings representing the desired label for a particular data point.
Series	The Series property is an indexed property that contains all data series for a particular ChartDataView. The order of ChartDataViewSeries objects in the series array corresponds to the drawing order.

Name	Description
Thresholds	The <b>Thresholds</b> property associates a list of thresholds with the data view.
Visible	The <b>Visible</b> property determines whether the dataview is showing or not. Default value is true.
VisibleInLegend	The <b>VisibleInLegend</b> property determines whether or not the view name and its series will appear in the chart legend.
XAxis	The <b>XAxis</b> property determines the x-axis against which the data in <b>ChartDataView</b> is plotted.
YAxis	The <b>YAxis</b> property determines the y-axis against which the data in <b>ChartDataView</b> is plotted.

## A.2 ChartDataViewSeries

Name	Description
DrawingOrder	The <b>DrawingOrder</b> property determines the order of display of data series. When the <b>DrawingOrder</b> property is changed, <b>ChartDataView</b> will normalize the order properties of all the <b>ChartDataViewSeries</b> objects that it manages.
FirstPoint	The <b>FirstPoint</b> property controls the index of the first point displayed in the <b>ChartDataViewSeries</b> .
Included	The <b>Included</b> property determines whether a data series is included in chart calculations (like axis bounds).
Label	The <b>Label</b> property controls the text that appears next to the data series inside the legend.
LastPoint	The <b>LastPoint</b> property controls the index of the first point displayed in the <b>ChartDataViewSeries</b> .
LastPointIsDefault	The <b>LastPointIsDefault</b> property determines whether the <b>LastPoint</b> property should be calculated from the data.
Name	The <b>Name</b> property represents the name of the data series. In <b>JClass Chart</b> , data series are named, and can be retrieved by name.
Style	The <b>Style</b> property defines the rendering style for the data series.



Name	Description
Visible	The <code>Visible</code> property determines whether the data series is showing in the chart area. Note that data series that are not showing are still used in axis calculations. See the <code>Included</code> property for details on how to omit a data series from chart calculations.
VisibleInLegend	The <code>VisibleInLegend</code> property determines whether or not this series will appear in the chart legend.

### A.3 ChartText

Name	Description
Adjust	The <code>Adjust</code> property determines how text is justified (positioned) in the label. Valid values include <code>ChartText.LEFT</code> , <code>ChartText.CENTER</code> , and <code>ChartText.RIGHT</code> . The default value is <code>ChartText.LEFT</code> .
Background	The <code>Background</code> property determines the background color used to draw inside the chart region. Note that the <code>Background</code> property is inherited from the parent <code>ChartRegion</code> .
Font	The <code>Font</code> property determines what font is used to render text inside the chart region. Note that the <code>Font</code> property is inherited from the parent <code>ChartRegion</code> .
Foreground	The <code>Foreground</code> property determines the foreground color used to draw inside the chart region. Note that the <code>Foreground</code> property is inherited from the parent <code>ChartRegion</code> .
Height	The <code>Height</code> property determines the height of the <code>ChartRegion</code> . The default value is calculated.
HeightIsDefault	The <code>HeightIsDefault</code> property determines whether the height of the chart region is calculated by <code>Chart</code> ( <code>true</code> ) or taken from the <code>Height</code> property ( <code>false</code> ). The default value is <code>true</code> .
Insets	The <code>Insets</code> property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.

Name	Description
Left	The <code>Left</code> property determines the location of the left of the <code>ChartRegion</code> . The default value is calculated.
LeftIsDefault	The <code>LeftIsDefault</code> property determines whether the left position of the chart region is calculated by the chart ( <code>true</code> ) or taken from the <code>Left</code> property ( <code>false</code> ). The default value is <code>true</code> .
Name	The <code>Name</code> property specifies a <code>String</code> identifier for the <code>ChartRegion</code> object.
Rotation	The <code>Rotation</code> property controls the rotation of the label. Valid values include <code>ChartText.DEG_90</code> , <code>ChartText.DEG_180</code> , <code>ChartText.DEG_270</code> , and <code>ChartText.DEG_0</code> . The default value is <code>ChartText.DEG_0</code> .
Text	The <code>Text</code> property is a <code>String</code> property that represents the text to be displayed inside the chart label. Default value is “ ” (empty <code>String</code> ).
Top	The <code>Top</code> property determines the location of the top of the <code>ChartRegion</code> . The default value is calculated.
TopIsDefault	The <code>TopIsDefault</code> property determines whether the top position of the chart region is calculated by the chart ( <code>true</code> ) or taken from the <code>Top</code> property ( <code>false</code> ). The default value is <code>true</code> .
Visible	The <code>Visible</code> property determines whether the associated <code>ChartRegion</code> is currently visible. Default value is <code>true</code> .
Width	The <code>Width</code> property determines the width of the <code>ChartRegion</code> . The default value is calculated.
WidthIsDefault	The <code>WidthIsDefault</code> property determines whether the width of the chart region is calculated by the chart ( <code>true</code> ) or taken from the <code>Width</code> property ( <code>false</code> ). The default value is <code>true</code> .

## A.4 JCAнно

Name	Description
AnnotationList	The <code>AnnotationList</code> property represents the list of <code>JCAнно</code> objects that have been added to the axis.

Name	Description
DrawLabels	The DrawLabels property determines whether or not labels are drawn for a series of annotations.
DrawTicks	The DrawTicks property determines whether or not tick marks are drawn for a series of annotations.
GridList	The GridList property represents the list of JCGrid objects that have been added to the axis.
IncrementValue	The IncrementValue property determines the value spacing between annotations.
InnerExtent	The InnerExtent property determines the pixel distance that tick marks extend away from the plot area.
LabelColor	The LabelColor property determines the color of labels.
LabelExtent	The LabelExtent property determines the pixel distance of labels from the axis.
OuterExtent	The OuterExtent property determines the pixel distance that tick marks extend away from the plot area.
Precision	The Precision property determines the precision to which numeric labels are displayed.
StartValue	The StartValue property determines the axis value at which the drawing of annotations begin.
StopValue	The StopValue property determines the axis value where the drawing of annotations end.
TickColor	The TickColor property determines the color of tick marks.
Type	The Type property determines the tick object's type. Default_Labels defines the JCAнно object to be the set of default labels for the axis. Default_Ticks defines the JCAнно object to be the set of default ticks for the axis. Default is User_Defined.
UseAnnoTicks	The UseAnnoTicks property determines whether or not tick marks are drawn at the labels when the annotation method is VALUE_LABELS, POINT_LABELS, or TIME_LABELS. If True, tick marks are drawn at the labels. If False, tick marks are drawn according to the properties of one or more JCAнно objects.

## A.5 JCAreaChartFormat

Name	Description
100Percent	The 100Percent property determines whether a stacking area will be charted versus an axis representing a percentage between 0 and 100. Default value is false.

## A.6 JCAxis

Name	Description
AnnotationMethod	The AnnotationMethod property determines how axis annotations are generated. Valid values are JCAxis.VALUE (annotation is generated by the chart, with possible callbacks to a label generator); JCAxis.VALUE_LABELS (annotation is taken from a list of value labels provided by the user -- a value label is a label that appears at a particular axis value); JCAxis.POINT_LABELS (annotation comes from the data source's point labels that are associated with particular data points); and JCAxis.TIME_LABELS (the chart generates time/date labels based on the TimeUnit, TimeBase and TimeFormat properties). Default value is JCAxis.VALUE.
AnnotationRotation	The AnnotationRotation property specifies the rotation of each axis label. Valid values are JCAxis.ROTATE_90, JCAxis.ROTATE_180, JCAxis.ROTATE_270, JCAxis.ROTATE_NONE, or JCAxis.ROTATE_OTHER. Default value is JCAxis.ROTATE_NONE.
AnnotationRotationAngle	Specifies the angle of the annotation for the currently selected axis. The angle is always set in degrees.
AnnotationVisible	The AnnotationVisible property determines whether or not the annotation is visible.
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent ChartRegion.

Name	Description
DropOverlappingLabels	The DropOverlappingLabels property determines whether or not overlapping labels are dropped. When true, one or more of the overlapping labels are eliminated so that the remaining labels do not overlap.
Editable	The Editable property determines whether the axis can be affected by edit/translation/zooming. Default value is true.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent ChartRegion.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent ChartRegion.
Formula	The Formula property determines how an axis is related to another axis object. If set, the Formula property overrides all other axis properties. See JCAxisFormula for details.
Gap	The Gap property determines the amount of space left between adjacent axis annotations, in pixels.
GeneratedValueLabels	The GeneratedValueLabels property reveals the value label at the specified index in the list of value labels generated for this axis.
GridDefault	The GridDefault property determines whether or not grid lines are drawn at the labels.
GridSpacing	The GridSpacing property controls the spacing between gridlines relative to the axis. Default value is 0.0.
GridSpacingIsDefault	The GridSpacingIsDefault property determines whether the chart is responsible for calculating the grid spacing value. If true, the chart will calculate the grid spacing. If false, the chart will use the provided grid spacing. Default value is true.
GridStyle	The GridStyle property controls how grids are drawn. The default value is generated.
GridVisible	The GridVisible property determines whether a grid is drawn for the axis. Default value is false.

Name	Description
Height	The Height property determines the height of the ChartRegion. The default value is calculated.
HeightIsDefault	The HeightIsDefault property determines whether the height of the chart region is calculated by the chart (true) or taken from the Height property (false). Default value is true.
LabelGenerator	The LabelGenerator property holds a reference to an object that implements the JCLabelGenerator interface. This interface is used to externally generate labels if the AnnotationMethod property is set to JCAxis.VALUE. Default value is null.
Left	The Left property determines the location of the left of the ChartRegion. The default value is calculated.
LeftIsDefault	The LeftIsDefault property determines whether the left position of the chart region is calculated by the chart (true) or taken from the Left property (false). Default value is true.
Logarithmic	The Logarithmic property determines whether the axis will be logarithmic (true) or linear (false). Default value is false.
Max	The Max property controls the maximum value shown on the axis. The data max is determined by the chart. Default value is calculated.
MaxIsDefault	The MaxIsDefault property determines whether the chart is responsible for calculating the maximum axis value. If true, the chart calculates the axis max. If false, the chart uses the provided axis max. Default value is true.
Min	The Min property controls the minimum value shown on the axis. The data min is determined by the chart. Default value is calculated.
MinIsDefault	The MinIsDefault property determines whether the chart is responsible for calculating the minimum axis value. If true, the chart will calculate the axis min. If false, the chart will use the provided axis min. Default value is true.

Name	Description
Name	The <code>Name</code> property specifies a String identifier for the <code>ChartRegion</code> object. Note that the <code>Name</code> property is inherited from the parent <code>ChartRegion</code> .
NumSpacing	The <code>NumSpacing</code> property controls the interval between axis labels. The default value is calculated.
NumSpacingIsDefault	The <code>NumSpacingIsDefault</code> property determines whether the chart is responsible for calculating the numbering spacing. If <code>true</code> , the chart will calculate the spacing. If <code>false</code> , the chart will use the provided numbering spacing. Default value is <code>true</code> .
Origin	The <code>Origin</code> property controls location of the origin along the axis. The default value is calculated.
OriginIsDefault	The <code>OriginIsDefault</code> property determines whether the chart is responsible for positioning the axis origin. If <code>true</code> , the chart calculates the axis origin. If <code>false</code> , the chart uses the provided axis origin value. Default value is <code>true</code> .
OriginPlacement	The <code>OriginPlacement</code> property specifies where the origin is placed. Note that the <code>OriginPlacement</code> property is only active if the <code>Origin</code> property has not been set. Valid values include <code>AUTOMATIC</code> (places origin at minimum value), <code>ZERO</code> (places origin at zero), <code>MIN</code> (places origin at minimum value on axis) or <code>MAX</code> (places origin at maximum value on axis). Default value is <code>AUTOMATIC</code> .
OriginPlacementIsDefault	The <code>OriginPlacementIsDefault</code> property determines whether the chart is responsible for determining the location of the axis origin. If <code>true</code> , the chart calculates the origin positioning. If <code>false</code> , the chart uses the provided origin placement.

Name	Description
Placement	The Placement property determines the method used to place the axis. Valid values include JCAxis.AUTOMATIC (the chart chooses an appropriate location), JCAxis.ORIGIN (appears at the origin of another axis, specified via the PlacementAxis property), JCAxis.MIN (appears at the minimum axis value), JCAxis.MAX (appears at the maximum axis value) or JCAxis.VALUE_ANCHORED (appears at a particular value along another axis, specified via the PlacementAxis property). Default value is AUTOMATIC.
PlacementAxis	The PlacementAxis property determines the axis that controls the placement of this axis. In JCCart, it is possible to position an axis at a particular position on another axis (in conjunction with the PlacementLocation property or the Placement property). Default value is null.
PlacementIsDefault	The PlacementIsDefault property determines whether the chart is responsible for determining the location of the axis. If true, the chart calculates the axis positioning. If false, the chart uses the provided axis placement.
PlacementLocation	The PlacementLocation property is used with the PlacementAxis property to position the current axis object at a particular point on another axis. Default value is 0.0.
Precision	The Precision property controls the number of zeros that appear after the decimal place in chart-generated axis labels. The default value is calculated.
PrecisionIsDefault	The PrecisionIsDefault determines whether the chart is responsible for calculating the numbering precision. If true, the chart will calculate the precision. If false, the chart will use the provided precision. Default value is true.
Reversed	The Reversed property of JCAxis determines if the axis direction is reversed. Default value is false.



Name	Description
TickSpacing	The <code>TickSpacing</code> property controls the interval between tick lines on the axis. Note: if the <code>AnnotationMethod</code> property is set to <code>POINT_LABELS</code> , tick lines appear at point values. The default value is calculated.
TickSpacingIsDefault	The <code>TickSpacingIsDefault</code> property determines whether the chart is responsible for calculating the tick spacing. If <code>true</code> , the chart will calculate the tick spacing. If <code>false</code> , the chart will use the provided tick spacing. Default value is <code>true</code> .
TimeBase	The <code>TimeBase</code> property defines the start time for the axis. Default value is the current time.
TimeFormat	The <code>TimeFormat</code> property controls the format used to generate time labels for time labelled axes. The formats supported are the same as in Java's <code>SimpleDateFormat</code> class. Default value is calculated based on <code>TimeUnit</code> .
TimeFormatIsDefault	The <code>TimeFormatIsDefault</code> property determines whether a time label format is generated automatically, or the user value for <code>TimeFormat</code> is used. Default value is <code>true</code> .
TimeUnit	The <code>TimeUnit</code> property controls the unit of time used for labelling a time labelled axis. Valid <code>TimeUnit</code> values include <code>JCAxis.SECONDS</code> , <code>JCAxis.MINUTES</code> , <code>JCAxis.HOURS</code> , <code>JCAxis.DAYS</code> , <code>JCAxis.WEEKS</code> , <code>JCAxis.MONTHS</code> and <code>JCAxis.YEARS</code> . Default value is <code>JCAxis.SECONDS</code> .
TimeZone	The <code>TimeZone</code> property specifies the time zone for this axis. Use only for time-based labels.
Title	The <code>Title</code> property controls the appearance of the axis title.
Top	The <code>Top</code> property determines the location of the top of the <code>ChartRegion</code> . The default value is calculated.
TopIsDefault	The <code>TopIsDefault</code> property determines whether the top position of the chart region is calculated by the chart ( <code>true</code> ) or taken from the <code>Top</code> property ( <code>false</code> ). Default value is <code>true</code> .

Name	Description
ValueLabels	The <code>ValueLabels</code> property is an indexed property containing a list of all annotations for an axis. Default value is <code>null</code> , no value labels.
Vertical	The <code>Vertical</code> property determines whether the associated <code>Axis</code> is vertical. Default value is <code>false</code> .
Visible	The <code>Visible</code> property determines whether the associated <code>Axis</code> is currently visible. Default value is <code>true</code> . Note that the <code>Font</code> property is inherited from the parent <code>ChartRegion</code> .
Width	The <code>Width</code> property determines the width of the <code>ChartRegion</code> . The default value is calculated.
WidthIsDefault	The <code>WidthIsDefault</code> property determines whether the width of the chart region is calculated by the chart ( <code>true</code> ) or taken from the <code>Width</code> property ( <code>false</code> ). Default value is <code>true</code> .

## A.7 JCAxisFormula

Name	Description
Constant	The <code>Constant</code> property specifies the “c” value in the axis relationship $y2 = my + c$ .
Multiplier	The <code>Multiplier</code> property specifies the “m” value in the relationship $y2 = my + c$ .
Originator	The <code>Originator</code> property specifies an object representing the axis that is related to the current axis by the formula $y = mx + c$ . The originator is “x”.

## A.8 JCAxisTitle

Name	Description
Adjust	The <code>Adjust</code> property determines how text is justified (positioned) in the label. Valid values include <code>ChartText.LEFT</code> , <code>ChartText.CENTER</code> , and <code>ChartText.RIGHT</code> . Default value is <code>ChartText.LEFT</code> .

Name	Description
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent ChartText.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent ChartText.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent ChartText.
Height	The Height property defines the height of the chart region. The default value is calculated.
HeightIsDefault	The HeightIsDefault property determines whether the height of the chart region is calculated by the chart (true) or taken from the Height property (false).
Left	The Left property determines the location of the left of the chart region. The default value is calculated.
LeftIsDefault	The LeftIsDefault property determines whether the left position of the chart region is calculated by the chart (true) or taken from the Left property (false).
Placement	The Placement property controls where the JCAxis title is placed relative to the “opposing” axis. Valid values include JCLegend.NORTH or JCLegend.SOUTH for horizontal axes, and JCLegend.EAST, JCLegend.WEST, JCLegend.NORTHEAST, JCLegend.SOUTHEAST, JCLegend.NORTHWEST or JCLegend.SOUTHWEST for vertical axes. The default value is calculated.
PlacementIsDefault	The PlacementIsDefault property determines whether the chart is responsible for calculating a reasonable default placement for the axis title. Default value is true.
Rotation	The Rotation property controls the rotation of the label. Valid values include ChartText.DEG_90, ChartText.DEG_180, ChartText.DEG_270, and ChartText.DEG_0. Default value is ChartText.DEG_0.
Text	The Text property is a String property that represents the text to be displayed inside the chart label. Default value is “ ” (nothing).

Name	Description
Top	The <code>Top</code> property determines the location of the top of the chart region. The default value is calculated.
TopIsDefault	The <code>TopIsDefault</code> property determines whether the top position of the chart region is calculated by the chart ( <code>true</code> ) or taken from the <code>Top</code> property ( <code>false</code> ).
Visible	The <code>Visible</code> property determines whether the associated Axis is currently visible. Default value is <code>true</code> .
Width	The <code>Width</code> property defines the width of the chart region. The default value is calculated.
WidthIsDefault	The <code>WidthIsDefault</code> property determines whether the width of the chart region is calculated by the chart ( <code>true</code> ) or taken from the <code>Width</code> property ( <code>false</code> ).

## A.9 JCBBarChartFormat

Name	Description
100Percent	The <code>100Percent</code> property determines whether stacking bar charts will be charted versus an axis representing a percentage between 0 and 100. Default value is <code>false</code> .
ClusterOverlap	The <code>ClusterOverlap</code> property specifies the overlap between bars. Valid values are between -100 and 100. Default value is 0.
ClusterWidth	The <code>ClusterWidth</code> property determines the percentage of available space which will be occupied by the bars. Valid values are between 0 and 100. Default value is 80.

## A.10 JCCandleChartFormat

Name	Description
CandleOutlineStyle	The <code>CandleOutlineStyle</code> determines the candle outline style of the complex candle chart.

Name	Description
Complex	The <code>Complex</code> property determines whether candle charts use the simple or the complex display style. When <code>false</code> , the chart only uses the style referenced by <code>getHiLoStyle()</code> for the candle appearance. When set to <code>true</code> , all four styles are used. Default value is <code>false</code> .
FallingCandleStyle	The <code>FallingCandleStyle</code> determines the candle style of the falling candle style of the complex candle chart.
HiloStyle	The <code>HiloStyle</code> determines the candle style of the simple candle or the Hi-Lo line of the complex candle chart.
RisingCandleStyle	The <code>RisingCandleStyle</code> determines the rising candle style of the complex candle chart.

## A.11 JCChart

Name	Description
About	The <code>About</code> property displays contact information for Quest Software in the bean box.
AllowUserChanges	The <code>AllowUserChanges</code> property determines whether the user viewing the graph can modify graph values. Default value is <code>false</code> .
Background	The <code>Background</code> property determines the background color used to draw inside the chart region. Note that the <code>Background</code> property is inherited from the parent <code>JCComponent</code> .
Batched	The <code>Batched</code> property controls whether chart updates are accumulated. Default value is <code>false</code> .
CancelKey	The <code>CancelKey</code> property specifies the key used to perform a cancel operation.
Changed	The <code>Changed</code> property determines whether the chart requires recalculation. Default value is <code>false</code> .
ChartArea	The <code>ChartArea</code> property controls the object that controls the display of the graph. Default value is <code>null</code> .
ChartLabelManager	The <code>ChartLabelManager</code> property manages all chart labels.

Name	Description
CustomizerName	The CustomizerName property specifies the full class name of the Chart Customizer. Default is <code>com.klg.jclass.chart.customizer.ChartCustomizer</code> .
DataRow	The DataRow property is an indexed property that contains all the data to be displayed in Chart. See <code>ChartDataView</code> for details on data format. By default, one <code>ChartDataView</code> is created.
FillColorIndex	The FillColorIndex property controls the fill color index. Default value is 0.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent <code>JCCComponent</code> .
Footer	The Footer property controls the object that controls the display of the footer. Default value is a <code>JLabel</code> instance
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent <code>JCCComponent</code> .
Header	The Header property controls the object that controls the display of the header. Default value is a <code>null</code> .
LayoutHints	The LayoutHints property sets layout hints for a child component of <code>JClass Chart</code> .
Legend	The Legend property controls the object that controls the display of the legend. Default value is an instance of <code>JCGridLegend</code> .
LineColorIndex	The LineColorIndex property controls the line color index. Default value is 0.
NumData	The NumData property indicates how many <code>ChartDataView</code> objects are stored in <code>JCChart</code> . This is a read-only property. Default value is 1.
NumTriggers	The NumTriggers property indicates how many event triggers have been specified.
ResetKey	The ResetKey property specifies the key used to perform a reset operation.
SymbolColorIndex	The SymbolColorIndex property controls the symbol color index. Default value is 0.

Name	Description
SymbolShapeIndex	The SymbolShapeIndex property controls the symbol shape index. Default value is 1.
Trigger	The Trigger property is an indexed property that contains all the information necessary to map user events into Chart actions. The Trigger property is made up of a number of EventTrigger objects. Default value is empty.
WarningDialog	The WarningDialog property determines whether JClass Chart will display a warning dialogue when required.

## A.12 JCChartArea

Name	Description
AngleUnit	The AngleUnit property determines the unit of all angle values. Default value is DEGREES.
AxisBoundingBox	The AxisBoundingBox property determines whether a box is drawn around the area bound by the inner axes.
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent JCChart.
Depth	The Depth property controls the apparent depth of a graph. Default value is 0.0.
Elevation	The Elevation property controls distance from the x-axis. Default value is 0.0.
FastAction	The FastAction property determines whether chart actions will use an optimized mode in which it does not bother to display axis annotations or gridlines. Default value is false.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent JCChart.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent JCChart.
HorizActionAxis	The HorizActionAxis property determines the axis used for actions (zooming, translating) in the horizontal direction. Default value is null.

Name	Description
PlotArea	The <code>PlotArea</code> property represents the region of the <code>ChartArea</code> that is inside the axes. This property is read-only.
Rotation	The <code>Rotation</code> property controls the position of the eye relative to the y-axis. Default value is 0.0.
VertActionAxis	The <code>VertActionAxis</code> property determines the axis used for actions (zooming, translating) in the vertical direction. Default value is null.
Visible	If true, the <code>ChartRegion</code> will appear on the screen. If false, it will not appear on the screen. (Legend, header, footer and chart area are all <code>ChartRegion</code> instances.) Default value is true.
XAxis	The <code>XAxis</code> property is an indexed property that contains all the x-axes for the chart area. Default value is one x-axis.
YAxis	The <code>YAxis</code> property is an indexed property that contains all the y-axes for the chart area. Default value is one y-axis.

## A.13 JCChartLabel

Name	Description
Anchor	Specifies how the label is to be positioned relative to the specified point. Valid values are <code>JCChartLabel.NORTHEAST</code> , <code>JCChartLabel.NORTHWEST</code> , <code>JCChartLabel.NORTH</code> , <code>JCChartLabel.EAST</code> , <code>JCChartLabel.WEST</code> , <code>JCChartLabel.SOUTHEAST</code> , <code>JCChartLabel.SOUTHWEST</code> , <code>JCChartLabel.SOUTH</code> , <code>JCChartLabel.CENTER</code> , or <code>JCChartLabel.AUTO</code> .
AttachMethod	Specifies how the label is attached to the chart. Valid values are <code>JCChartLabel.ATTACH_COORD</code> (attach label to an absolute point anywhere on the chart), <code>JCChartLabel.ATTACH_DATACOORD</code> (attach label to a point in the data space on the chart area), and <code>JCChartLabel.ATTACH_DATAINDEX</code> (attach the label to a specific point/bar/slice on the chart).
Component	The Swing component used as the chart label. By default, this is a <code>JLabel</code> instance.
Coord	The coordinate in the chart's space where the label is to be attached.



Name	Description
DataCoord	The coordinate in the chart area's data space where the label is to be attached.
DataIndex	A data index representing the point/bar/slice in the chart to which the label is to be attached.
DataRow	For labels using ATTACH_DATACoord, this property specifies which ChartDataRow's axes should be used.
Offset	The Offset property specifies where the label should be positioned relative to the position the labels thinks it should be, depending on what the label's attachMethod is.
ParentManager	The ParentManager property is the ChartLabelManager instance that controls the JCCartLabel.
Text	The Text property controls the text displayed inside the label.

## A.14 JCCartLabelManager

Name	Description
AutoLabelList	The AutoLabelList property is a two-dimensional array of automatically-generated JCCartLabel instances, one for every point and series. The inner array is indexed by point; the outer array by series. Default is empty.

## A.15 JCCartStyle

Name	Description
FillColor	The FillColor property determines the color used to fill regions in chart. Default value is generated.
FillImage	The FillImage property determines the image used to paint the fill region of bar and area charts. Default value is null.
FillPattern	The FillPattern property determines the fill pattern used to fill regions in chart. Default value is JCFillStyle.SOLID.

Name	Description
FillStyle	The FillStyle property controls the appearance of filled areas in chart. See JCFillStyle for additional properties. Note that all JCCartStyle properties of the format Fill* are virtual properties that map to properties of JCFillStyle.
LineCap	The LineCap property specifies the cap style used to end a line. Valid values include BasicStroke.CAP_BUTT, BasicStroke.CAP_ROUND, and BasicStroke.CAP_SQUARE.
LineColor	The LineColor property determines the color used to draw a line. Default value is generated.
LineJoin	The LineJoin property specifies the join style used to join two lines. Valid values include BasicStroke.JOIN_MITER, BasicStroke.JOIN_BEVEL, and BasicStroke.JOIN_ROUND.
LinePattern	The LinePattern property dictates the pattern used to draw a line. Valid values include JCLineStyle.NONE, JCLineStyle.SOLID, JCLineStyle.LONG_DASH, JCLineStyle.SHORT_DASH, JCLineStyle.LSL_DASH, and JCLineStyle.DASH_DOT. Default value is JCLineStyle.SOLID.
LineStyle	The LineStyle property controls the appearance of lines in chart. See JCLineStyle for additional properties.
LineWidth	The LineWidth property controls the line width. Default value is 1.
SymbolColor	The SymbolColor property determines the color used to paint the symbol. Default value is generated.
SymbolCustomShape	The SymbolCustomShape property contains an object derived from JCShape that is used to draw points. See JCShape for details. Default value is null.
SymbolShape	The SymbolShape property determines the type of symbol that will be drawn. Valid values include JCSymbolStyle.NONE, JCSymbolStyle.DOT, JCSymbolStyle.BOX, JCSymbolStyle.TRIANGLE, JCSymbolStyle.DIAMOND, JCSymbolStyle.STAR, JCSymbolStyle.VERT_LINE, JCSymbolStyle.HORIZ_LINE, JCSymbolStyle.CROSS, JCSymbolStyle.CIRCLE, and JCSymbolStyle.SQUARE. Default value is generated.
SymbolSize	The SymbolSize property determines the size of the symbol. Default value is 6.

Name	Description
SymbolStyle	The SymbolStyle property controls the symbol that represents an individual point. See JCSymbolStyle for additional properties. Note that all JCCartStyle properties of the format Symbol* are virtual properties that map to properties of JCSymbolStyle.

## A.16 JCFillStyle

Name	Description
Background	The Background property determines the background color used when painting patterned fills.
Color	The Color property determines the color used to fill regions in chart. The default value is generated.
CustomPaint	The CustomPaint property specifies the TexturePaint object used to paint the fill region when the pattern is set to CUSTOM_PAINT.
Image	The Image property determines the image used to paint the fill region when the pattern is set to CUSTOM_FILL or CUSTOM_STACK. The default value is null.
Pattern	<p>The Pattern property determines the fill pattern used to fill regions in chart. The default value is JCFillStyle.SOLID.</p> <p>Available fill patterns are: NONE, SOLID, 25_PERCENT, 50_PERCENT, 75_PERCENT, HORIZ_STRIPE, VERT_STRIPE, 45_STRIPE, 135_STRIPE, DIAG_HATCHED, CROSS_HATCHED, CUSTOM_FILL, CUSTOM_PAINT, or, for bar charts only, CUSTOM_STACK.</p>

## A.17 JCGrid

Name	Description
GridStyle	The GridStyle property determines the style in which grid lines are drawn.
IncrementValue	The IncrementValue property determines the value spacing between grid lines.

Name	Description
Orientation	The <code>Orientation</code> property determines how legend information is laid out. Valid values include <code>JCLegend.VERTICAL</code> and <code>JCLegend.HORIZONTAL</code> . The default value is <code>JCLegend.VERTICAL</code> .
StartValue	The <code>StartValue</code> property determines the axis value at which the drawing of grid lines begins.
StopValue	The <code>StopValue</code> property determines the axis value where the drawing of grid lines ends.
SymbolSize	The <code>SymbolSize</code> property determines the size of the symbol. Default value is 6.
Visible	The <code>Visible</code> property determines whether the grid lines defined by this object are visible.

## A.18 JCGridLegend

See also: [JCLegend](#)

Name	Description
Anchor	The <code>Anchor</code> property determines the position of the legend relative to the <code>ChartArea</code> . Valid values include <code>JCLegend.NORTH</code> , <code>JCLegend.SOUTH</code> , <code>JCLegend.EAST</code> , <code>JCLegend.WEST</code> , <code>JCLegend.NORTHWEST</code> , <code>JCLegend.SOUTHWEST</code> , <code>JCLegend.NORTHEAST</code> , and <code>JCLegend.SOUTHEAST</code> . The default value is <code>JCLegend.EAST</code> .
Background	The <code>Background</code> property determines the background color used to draw inside the legend. Note that the <code>Background</code> property is inherited from the parent <code>JCChart</code> .
Font	The <code>Font</code> property determines what font is used to render text inside the legend. Note that the <code>Font</code> property is inherited from the parent <code>JCChart</code> .
Foreground	The <code>Foreground</code> property determines the foreground color used to draw inside the legend. Note that the <code>Foreground</code> property is inherited from the parent <code>JCChart</code> .
GroupGap	The <code>GroupGap</code> property determines the gap between groups of items in the chart legend (e.g. the columns/rows associated with a data view).

Name	Description
InsideItemGap	The <code>InsideItemGap</code> property determines the gap between the symbol and text portions of a legend item.
ItemGap	The <code>ItemGap</code> property determines the gap between the legend items in the same group.
MarginGap	The <code>MarginGap</code> property determines the gap between the edge of the legend and the start of the item layout.
Orientation	The <code>Orientation</code> property determines how legend information is laid out. Valid values include <code>JCLegend.VERTICAL</code> and <code>JCLegend.HORIZONTAL</code> . The default value is <code>JCLegend.VERTICAL</code> .
SymbolSize	The <code>SymbolSize</code> property determines the size of the symbol. Default value is 6.
Visible	The <code>Visible</code> property determines the gap between the legend items in the same group.

## A.19 JCHiloChartFormat

Name	Description
HiloStyle	The <code>HiloStyle</code> property specifies the <code>JCChartStyle</code> object that defines the style of the chart.

## A.20 JCHLOCChartFormat

Name	Description
HiloStyle	The <code>HiloStyle</code> property specifies the <code>JCChartStyle</code> object that defines the style of the chart.
OpenCloseFullWidth	The <code>OpenCloseFullWidth</code> property indicates whether the open and close tick indications are drawn across the full width of the Hi-Lo bar or just on one side. The default value is <code>false</code> .
ShowingClose	The <code>ShowingClose</code> property indicates whether the close tick indication is shown or not. The tick appears to the right of the Hi-Lo line. The default value is <code>true</code> .

Name	Description
ShowingOpen	The ShowingOpen property indicates whether the open tick indication is shown or not. The tick appears to the left of the Hi-Lo line. The default value is true.
TickSize	The TickSize property specifies the tick size for open and close ticks.

## A.21 JCLegend

Name	Description
Anchor	The Anchor property determines the position of the legend relative to the ChartArea. Valid values include JCLegend.NORTH, JCLegend.SOUTH, JCLegend.EAST, JCLegend.WEST, JCLegend.NORTHWEST, JCLegend.SOUTHWEST, JCLegend.NORTHEAST, and JCLegend.SOUTHEAST. The default value is JCLegend.EAST.
Background	The Background property determines the background color used to draw inside the legend. Note that the Background property is inherited from the parent JCCart.
Border	The Border property sets the border of a component. Note that the Border property is inherited from JComponent.
Font	The Font property determines what font is used to render text inside the legend. Note that the Font property is inherited from the parent JCCart.
Foreground	The Foreground property determines the foreground color used to draw inside the legend. Note that the Foreground property is inherited from the parent JCCart.
ItemTextAlignment	The ItemTextAlignment property determines the alignment for the text in a column. Valid values are: SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.RIGHT, SwingConstants.LEADING, and SwingConstants.TRAILING. Default is SwingConstants.LEADING. By default, this property applies to all columns, but you can set it on individual columns by specifying the column number.
ItemTextToolTipEnabled	The ItemTextToolTipEnabled property determines whether or not tooltips are displayed when the mouse hovers over a legend item.

Name	Description
MaxItemTextWidth	The <code>MaxItemTextWidth</code> property specifies the maximum width of the column in pixels. If the column text exceeds this width, the text is truncated. By default, this property applies to all columns, but you can set it on individual columns by specifying the column number.
Opaque	The <code>Opaque</code> property determines the background color. If the component is completely opaque, the background will be filled with the background color. Otherwise, the background is transparent, and whatever is underneath will show through. Note, that the <code>Opaque</code> property is inherited from <code>JComponent</code> .
Orientation	The <code>Orientation</code> property determines how legend information is laid out. Valid values include <code>JCLegend.VERTICAL</code> and <code>JCLegend.HORIZONTAL</code> . The default value is <code>JCLegend.VERTICAL</code> .
TruncateMode	The <code>TruncateMode</code> property determines how text is truncated when the length of the text exceeds the maximum width of the column. Valid values are: <code>JCLegend.TRUNCATE_LEFT</code> , <code>JCLegend.TRUNCATE_RIGHT</code> , <code>JCLegend.TRUNCATE_MIDDLE</code> , <code>JCLegend.TRUNCATE_END</code> , <code>JCLegend.TRUNCATE_LEADING</code> , and <code>JCLegend.TRUNCATE_TRAILING</code> . Default is <code>TRUNCATE_TRAILING</code> . By default, this property applies to all columns, but you can set it on individual columns by specifying the column number.
UseEllipsisWhenTruncating	The <code>UseEllipsisWhenTruncating</code> property determines whether or not an ellipsis is used to indicate truncated legend text.
Visible	The <code>Visible</code> property determines whether the legend is currently visible. Default value is <code>false</code> .

## A.22 JCLineStyle

Name	Description
Cap	The <code>Cap</code> property specifies the cap style used to end a line. Valid values include <code>BasicStroke.CAP_BUTT</code> , <code>BasicStroke.CAP_ROUND</code> , and <code>BasicStroke.CAP_SQUARE</code> .
Color	The <code>Color</code> property determines the color used to draw a line. The default value is generated.

Name	Description
Join	The <code>Join</code> property specifies the join style used to join two lines. Valid values include <code>BasicStroke.JOIN_MITER</code> , <code>BasicStroke.JOIN_BEVEL</code> , and <code>BasicStroke.JOIN_ROUND</code> .
Pattern	The <code>Pattern</code> property dictates the pattern used to draw a line. Valid values include <code>JCLineStyle.NONE</code> , <code>JCLineStyle.SOLID</code> , <code>JCLineStyle.LONG_DASH</code> , <code>JCLineStyle.SHORT_DASH</code> , <code>JCLineStyle.LSL_DASH</code> , and <code>JCLineStyle.DASH_DOT</code> . The default value is <code>JCLineStyle.SOLID</code> .
Width	The <code>Width</code> property controls the line width. The default value is 1.

## A.23 JCMarker

Name	Description
<code>AssociatedWithYAxis</code>	The <code>AssociatedWithYAxis</code> property specifies whether the marker is associated with the y-axis ( <code>true</code> ) or the x-axis ( <code>false</code> ). Default is <code>true</code> .
<code>ChartLabel</code>	The <code>ChartLabel</code> property is of type <code>JCChartLabel</code> . Setting this property allows a chart label to be attached to the marker (the default is <code>null</code> which means no chart label is attached to the marker). The default attachment point is on the marker, halfway between the start point and the end point.
<code>DrawnBeforeData</code>	The <code>DrawnBeforeData</code> property specifies whether the marker is drawn before the data ( <code>true</code> ) or after the data ( <code>false</code> ). Default is <code>false</code> .
<code>EndPoint</code>	The <code>EndPoint</code> property specifies the value on the non-associated axis at which to end the marker line. The default is the maximum value on the axis.
<code>IncludedInDataBounds</code>	The <code>IncludedInDataBounds</code> property determines whether or not the marker's value is included when calculating the data minimum and data maximum for the data view.
<code>Label</code>	The <code>Label</code> property specifies the name of the marker and the label used in the legend (if <code>VisibleInLegend</code> is <code>true</code> ).
<code>LineStyle</code>	The <code>LineStyle</code> property determines the width, color, and type of the marker line via a <code>JCLineStyle</code> object. By default, the <code>LineStyle</code> is a solid black line of width 1 (one). Setting the <code>LineStyle</code> property to <code>null</code> means that no line is drawn.



Name	Description
StartPoint	The <code>StartPoint</code> property specifies the value on the non-associated axis at which to start the marker line. The default is the minimum value on the axis.
Value	The <code>Value</code> property specifies the value on the associated axis at which to draw the marker line.
VisibleInLegend	The <code>VisibleInLegend</code> property determines whether the marker label is displayed in the legend ( <code>true</code> ) or not ( <code>false</code> ). The default is <code>false</code> .

## A.24 JCMultiColLegend

See also: [JCLegend](#)

Name	Description
Anchor	The <code>Anchor</code> property determines the position of the legend relative to the <code>ChartArea</code> . Valid values include <code>JCLegend.NORTH</code> , <code>JCLegend.SOUTH</code> , <code>JCLegend.EAST</code> , <code>JCLegend.WEST</code> , <code>JCLegend.NORTHWEST</code> , <code>JCLegend.SOUTHWEST</code> , <code>JCLegend.NORTHEAST</code> and <code>JCLegend.SOUTHEAST</code> . The default value is <code>JCLegend.EAST</code> .
Background	The <code>Background</code> property determines the background color used to draw inside the legend. Note that the <code>Background</code> property is inherited from the parent <code>ChartRegion</code> .
Border	The <code>Border</code> property sets the border of a component. Note that the <code>Border</code> property is inherited from <code>JComponent</code> .
Font	The <code>Font</code> property determines what font is used to render text inside the legend. Note that the <code>Font</code> property is inherited from the parent <code>JCChart</code> .
Foreground	The <code>Foreground</code> property determines the foreground color used to draw inside the legend. Note that the <code>Foreground</code> property is inherited from the parent <code>JCChart</code> .
GroupGap	The <code>GroupGap</code> property determines the gap between groups of items in the chart legend (e.g. the columns/rows associated with a data view).
InsideItemGap	The <code>InsideItemGap</code> property determines the gap between the symbol and text portions of a legend item.

Name	Description
ItemGap	The ItemGap property determines the gap between the legend items in the same group.
MarginGap	The MarginGap property determines the gap between the edge of the legend and the start of the item layout.
NumColumns	The NumColumns property determines the number of columns in this legend. If the number of columns is set to zero (the default), then the NumColumns will be adjusted automatically.
NumRows	The NumRows property determines the number of rows in this legend. If the number of rows is set to zero (the default), the number of rows will be adjusted automatically.
Orientation	The Orientation property determines how legend information is laid out. Valid values include JCLegend.VERTICAL and JCLegend.HORIZONTAL. The default value is JCLegend.VERTICAL.
SymbolSize	The SymbolSize property determines the size of the symbol. Default value is 6.

## A.25 JCPieChartFormat

Name	Description
ExplodeList	The ExplodeList property specifies a list of exploded pie slices in the pie charts. Default value is an empty list.
ExplodeOffset	The ExplodeOffset property specifies the distance a slice is exploded from the center of a pie chart. Default value is 10.
MinSlices	The MinSlices property represents the minimum number of pie slices that the chart will try to display before grouping slices into the other slice. Default value is 5.
OtherLabel	The OtherLabel property represents the label used on the “other” pie slice. As with other point labels, the “other” label is a ChartText instance. Default value is “ ” (empty String).
OtherStyle	The OtherStyle property specifies the style used to render the “other” pie slice.

Name	Description
SortOrder	The SortOrder property determines the order in which pie slices will be displayed. Note that the other slice is always last in any ordering. Valid values include JCPieChartFormat.ASCENDING_ORDER, JCPieChartFormat.DECENDING_ORDER, and JCPieChartFormat.DATA_ORDER. Default value is JCPieChartFormat.DATA_ORDER.
StartAngle	The position in the pie chart where the first pie slice is drawn. A value of zero degrees represents a horizontal line from the center of the pie to the right-hand side of the pie chart; a value of 90 degrees represents a vertical line from the center of the pie to the top-most point of the pie chart; a value of 180 degrees represents a horizontal line from the center of the pie to the left-hand side of the pie chart; and so on. Slices are drawn clockwise from the specified angle. Values must lie in the range from zero degrees to 360 degrees. The default value is 135 degrees.
ThresholdMethod	The ThresholdMethod property determines how the ThresholdValue property is used. If the method is SLICE_CUTOFF, the ThresholdValue is used as a cutoff to determine what items are lumped into the other slice. If the method is PIE_PERCENTILE, items are groups into the other slice until it represents “ThresholdValue” percent of the pie. Default value is SLICE_CUTOFF.
ThresholdValue	The ThresholdValue property is a percentage value between 0.0 and 100.0. How this value is used depends on the ThresholdMethod property. Default value is 10.0.

## A.26 JCPolarRadarChartFormat

Name	Description
HalfRange	The HalfRange property determines whether the x-axis for polar charts consists of two half-ranges or one full range from 0 to 360 degrees.

Name	Description
OriginBase	The <code>OriginBase</code> property determines the angle of the theta axis origin in polar, radar, and area radar charts. Angles are based on zero degrees pointing east (the normal rectangular x-axis direction) with positive angles going counter-clockwise. The angle units are assumed to be the current value of the chart area's <code>angleUnit</code> property.
RadarCircularGrid	The <code>YAxisGridCircular</code> property determines whether gridlines are circular or “webbed” for radar and area radar charts.
YAxisAngle	The <code>YAxisAngle</code> property determines the angle of the y-axis in polar, radar, and area radar charts. Angles are relative to the current origin base. The angle units are assumed to be the current value of the chart area's <code>angleUnit</code> property.

## A.27 JCSymbolStyle

Name	Description
Color	The <code>Color</code> property determines the color used to paint the symbol. The default value is generated.
CustomShape	The <code>CustomShape</code> property contains an object derived from <code>JCShape</code> that is used to draw points. See <code>JCShape</code> for details. The default value is <code>null</code> .
Shape	The <code>Shape</code> property determines the shape of symbol that will be drawn. Valid values include <code>JCSymbolStyle.NONE</code> , <code>JCSymbolStyle.DOT</code> , <code>JCSymbolStyle.BOX</code> , <code>JCSymbolStyle.TRIANGLE</code> , <code>JCSymbolStyle.DIAMOND</code> , <code>JCSymbolStyle.STAR</code> , <code>JCSymbolStyle.VERT_LINE</code> , <code>JCSymbolStyle.HORIZ_LINE</code> , <code>JCSymbolStyle.CROSS</code> , <code>JCSymbolStyle.CIRCLE</code> and <code>JCSymbolStyle.SQUARE</code> . The default value is <code>JCSymbolStyle.DOT</code> .
Size	The <code>Size</code> property determines the size of the symbol. The default value is 6.

## A.28 JCThreshold

Name	Description
AssociatedWithYAxis	The <code>AssociatedWithYAxis</code> property specifies whether the threshold is associated with the y-axis ( <code>true</code> ) or the x-axis ( <code>false</code> ). Default is <code>true</code> .
EndLineStyle	The <code>EndLineStyle</code> property determines the width, color, and type of a line on the end value of the threshold via a <code>JCLineStyle</code> object. By default, this property is null, which means no line is drawn on the end boundary.
EndValue	The <code>EndValue</code> property specifies the value on the associated axis at which to end the threshold.
IncludedInDataBounds	The <code>IncludedInDataBounds</code> property determines whether or not the threshold's <code>StartValue</code> and <code>EndValue</code> are included when calculating the data minimum and data maximum for the data view.
Label	The <code>Label</code> property specifies the name of the threshold and the label used in the legend (if <code>VisibleInLegend</code> is <code>true</code> ).
FillStyle	The <code>FillStyle</code> property determines the color and fill pattern of the threshold via a <code>JCFillStyle</code> object. By default, the <code>FillStyle</code> is a solid color determined by the chart to be different from the background color (it is best to set your own color). Setting this property to null means the threshold area is unfilled.
StartLineStyle	The <code>StartLineStyle</code> property determines the width, color, and type of a line on the start value of the threshold via a <code>JCLineStyle</code> object. By default, this property is null, which means no line is drawn on the start boundary.
StartValue	The <code>StartValue</code> property specifies the value on the associated axis at which to start the threshold.
VisibleInLegend	The <code>VisibleInLegend</code> property determines whether the threshold label is displayed in the legend ( <code>true</code> ) or not ( <code>false</code> ). The default is <code>false</code> .

## A.29 JCValueLabel

Name	Description
ChartText	The ChartText property controls the ChartText associated with this Value label. The default value is a ChartText instance.
Text	The Text property specifies the text displayed inside the label. The default value is “ ” (empty String).
Value	The Value property controls the position of a label in data space along a particular axis. The default value is 0.0.

## A.30 PlotArea

Name	Description
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background is inherited from the parent ChartRegion.
Bottom	The Bottom property determines the location of the bottom of the PlotArea.
BottomIsDefault	The BottomIsDefault property determines whether the Bottom of the chart region is calculated by the chart (true) or taken from the Bottom property (false).
Foreground	The Foreground property determines the color used to draw the axis bounding box controlled by JCChartArea. Note that the Foreground property is inherited from the parent ChartRegion.
Left	The Left property determines the location of the left of the PlotArea
LeftIsDefault	The LeftIsDefault property determines whether the left position of the chart region is calculated by the chart (true) or taken from the Left property (false).
Right	The Right property determines the Right of the PlotArea.
RightIsDefault	The RightIsDefault property determines whether the Right of the chart region is calculated by the chart (true) or taken from the Right property (false).
Top	The Top property determines the location of the top of the PlotArea.

Name	Description
TopIsDefault	The <b>TopIsDefault</b> property determines whether the top position of the chart region is calculated by the chart ( <b>true</b> ) or taken from the <b>Top</b> property ( <b>false</b> ).

## A.31 SimpleChart

Name	Description
AxisOrientation	The <b>AxisOrientation</b> property determines if the x- and y-axes are inverted and reversed.
Background	The <b>Background</b> property determines the background color used to draw inside the chart region. Note that the <b>Background</b> property is inherited from the parent <b>JCComponent</b> .
ChartType	The <b>ChartType</b> property determines the chart type of the first set of data in the chart.
Data	The <b>Data</b> property controls the file or URL used for the first set of data in chart.
Font	The <b>Font</b> property determines what font is used to render text inside the chart region. Note that the <b>Font</b> property is inherited from the parent <b>JCComponent</b> .
FooterFont	The <b>FooterFont</b> property determines what font is used to render text inside the footer region.
FooterText	The <b>FooterText</b> property holds the text that is displayed in the footer. The default value is “ ” (empty String).
Foreground	The <b>Foreground</b> property determines the foreground color used to draw inside the chart region. Note that the <b>Foreground</b> property is inherited from the parent <b>JCComponent</b> .
HeaderFont	The <b>HeaderFont</b> property determines what font is used to render text inside the header region.
HeaderText	The <b>HeaderText</b> property holds the text that is displayed in the header. The default value is “ ” (empty String).

Name	Description
LegendAnchor	The LegendAnchor property determines the position of the legend relative to the ChartArea. Valid values include NORTH, SOUTH, EAST, WEST, NORTHWEST, SOUTHWEST, NORTHEAST, and SOUTHEAST. The default value is EAST.
LegendOrientation	The LegendOrientation property determines how legend information is laid out. Valid values include VERTICAL and HORIZONTAL. The default value is VERTICAL.
LegendVisible	The LegendVisible property determines whether the legend is currently visible. Default value is false.
SwingDataModel	Sets the chart's data source to use a specified Swing TableModel object, instead of using the Data property.
View3D	The View3D property combines the values of the Depth, Elevation, and Rotation properties defined in JCCartArea. Depth controls the apparent depth of a graph. Elevation controls the distance above the x-axis for the 3D effect. Rotation controls the position of the eye relative to the y-axis for the 3D effect. The default value s "0.0,0.0,0.0".
XAxisAnnotationMethod	The XAxisAnnotationMethod property determines how axis annotations are generated. Valid values include VALUE (annotation is generated by the chart, with possible callbacks to a label generator), VALUE_LABELS (annotation is taken from a list of value labels provided by the user – a value label is a label that appears at a particular axis value), POINT_LABELS (annotation comes from the data source's point labels that are associated with particular data points), and TIME_LABELS (the chart generates time/date labels based on the TimeUnit, TimeBase and TimeFormat properties). The default value is VALUE.
XAxisGridVisible	The XAxisGridVisible property determines whether a grid is drawn for the axis. The default value is false.
XAxisLogarithmic	The XAxisLogarithmic property determines whether the first x-axis will be logarithmic (true) or linear (false). The default value is false.



Name	Description
XAxisMinMax	The <code>XAxisMinMax</code> controls both the <code>XAxisMin</code> and <code>XAxisMax</code> properties. The <code>XAxisMin</code> property controls the minimum value shown on the axis. If a null String is used, the chart will calculate the axis minimum. The data minimum is determined by the chart. The default value is calculated. The <code>XAxisMax</code> property controls the maximum value shown on the axis. If a null String is used, the chart will calculate the axis maximum. The data maximum is determined by the chart. The default value is calculated.
XAxisNumSpacing	The <code>XAxisNumSpacing</code> property controls the interval between axis labels. If a null String is used, the chart will calculate the interval. The default value is calculated.
XAxisTitleText	The <code>XAxisTitleText</code> property specifies the text that will appear as the x-axis title. The default value is “ ” (empty String).
XAxisVisible	The <code>XAxisVisible</code> property determines whether the first x-axis is currently visible. Default value is <code>true</code> .
YAxisAnnotationMethod	The <code>YAxisAnnotationMethod</code> property determines how axis annotations are generated. Valid values include <code>VALUE</code> (annotation is generated by the chart, with possible callbacks to a label generator), <code>VALUE_LABELS</code> (annotation is taken from a list of value labels provided by the user – a value label is a label that appears at a particular axis value), <code>POINT_LABELS</code> (annotation comes from the data source's point labels that are associated with particular data points), and <code>TIME_LABELS</code> (the chart generates time/date labels based on the <code>TimeUnit</code> , <code>TimeBase</code> and <code>TimeFormat</code> properties). The default value is <code>VALUE</code> .
YAxisGridVisible	The <code>YAxisGridVisible</code> property determines whether a grid is drawn for the axis.
YAxisLogarithmic	The <code>YAxisLogarithmic</code> property determines whether the first y-axis will be logarithmic ( <code>true</code> ) or linear ( <code>false</code> ). The default value is <code>false</code> .

Name	Description
YAxisMinMax	The YAxisMinMax controls both the YAxisMin and YAxisMax properties. The YAxisMin property controls the minimum value shown on the axis. If a null String is used, Chart will calculate the axis min. The data min is determined by Chart. The default value is calculated. The YAxisMax property controls the maximum value shown on the axis. If a null String is used, Chart will calculate the axis max. The data max is determined by Chart. The default value is calculated.
YAxisNumSpacing	The YAxisNumSpacing property controls the interval between axis labels. If a null String is used, Chart will calculate the interval. The default value is calculated.
YAxisTitleText	The YAxisTitleText property specifies the text that will appear as the y-axis title. The default value is “ ” (empty String).
YAxisVisible	The YAxisVisible property determines whether the first y-axis is currently visible. Default value is true.

## HTML Syntax

[ChartDataView Properties](#) ■ [ChartDataViewSeries Properties](#) ■ [Header and Footer Properties](#)  
[JCAreaChartFormat Properties](#) ■ [JCAAnnoProperties](#)  
[JCAxis X-Axes and Y-Axes Properties](#) ■ [JCBarChartFormat Properties](#) ■ [JCCandleChartFormat Properties](#)  
[JCChart Properties](#) ■ [JCChartArea Properties](#) ■ [JCChartLabel Properties](#) ■ [JCDataIndex Properties](#)  
[JCGrid Properties](#) ■ [JCHiLoChartFormat Properties](#) ■ [JCHLOCChartFormat Properties](#)  
[JCLegend Properties](#) ■ [JCMarker Properties](#) ■ [JCMultiColLegend Properties](#)  
[JCPieChartFormat Properties](#) ■ [JCPolarRadarChartFormat Properties](#) ■ [JCThreshold Properties](#)

This appendix lists the syntax of JClass Chart property parameters when specified in an HTML file. For example, the following HTML code sets the x-axis annotation method property:

```
<PARAM NAME="xaxis.annotationMethod" VALUE="POINT_LABELS">
```

Some value types are listed as *enum*. If you are unfamiliar with the enumerations available for a chart property, you can look up the property's class in the *API documentation* and then search on the property name. Enumerations are usually located with the `set` method for the property.

## B.1 ChartDataView Properties

Chart Property	HTML Syntax	Value Type
Auto Label	<i>data.autoLabel</i> <sup>a</sup>	boolean
Buffer Plot Data	<i>data.bufferPlotData</i>	boolean
Character Set	<i>data.fileCharset</i>	String
Chart Type	<i>data.chartType</i>	enum
Data	<i>data</i>	AppletDataSource
Data File	<i>dataFile</i> , <i>data1File</i> , or <i>data2File</i>	URLDataSource, FileDataSource
Data Name	<i>dataName</i> <i>n</i>	String <sup>b</sup>
Draw Front Plane	<i>data.drawFrontPlane</i>	boolean
Fast Update	<i>data.fastUpdate</i>	boolean
File Access	<i>data.fileAccess</i>	String
File Type	<i>data.fileType</i>	XML or Text
Hole Value	<i>data.holeValue</i>	double
Inverted	<i>data.inverted</i>	boolean
Name	<i>data.name</i>	String
Outline Color	<i>data.line.color</i>	Color
Outline Cap	<i>data.line.cap</i>	enum
Outline Join	<i>data.line.join</i>	enum
Outline Pattern	<i>data.line.pattern</i>	enum
Outline Width	<i>data.line.width</i>	int
Point Labels	<i>data.pointLabels</i>	String
Visible	<i>data.visible</i>	boolean
Visible In Legend	<i>data.visibleInLegend</i>	boolean
X Axis	<i>data.xaxis</i>	X axis name
Y Axis	<i>data.yaxis</i>	Y axis name

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *data<sub>n</sub>*.

b. *n* is the data view number; not needed for first data view.

## B.2 ChartDataViewSeries Properties

Chart Property	HTML Syntax	Value Type
Fill Background	<i>data.seriesn.fill.background</i> <sup>a</sup>	enum
Fill Color	<i>data.seriesn.fill.color</i>	Color
Fill Color Index	<i>data.seriesn.fill.colorIndex</i>	int
Fill Image	<i>data.seriesn.fill.image</i>	Image
Fill Image Name	<i>data.seriesn.fill.image.fileName</i>	String
Fill Image Access Type	<i>data.seriesn.fill.image.fileAccess</i>	String
Fill Pattern	<i>data.seriesn.fill.pattern</i>	enum
First Point	<i>data.seriesn.firstPoint</i>	int
Included	<i>data.seriesn.included</i>	boolean
Label	<i>data.seriesn.label</i>	String
Last Point	<i>data.seriesn.lastPoint</i>	int
Line Color	<i>data.seriesn.line.color</i>	Color
Line Color Index	<i>data.seriesn.line.colorIndex</i>	int
Line Cap	<i>data.seriesn.line.cap</i>	enum
Line Join	<i>data.seriesn.line.join</i>	enum
Line Pattern	<i>data.seriesn.line.pattern</i>	enum
Line Width	<i>data.seriesn.line.width</i>	int
Name	<i>data.seriesn.name</i>	String
Symbol Color	<i>data.seriesn.symbol.color</i>	Color
Symbol Color Index	<i>data.seriesn.symbol.colorIndex</i>	int
Symbol Shape	<i>data.seriesn.symbol.shape</i>	enum
Symbol Shape Index	<i>data.seriesn.symbol.symbolIndex</i>	int
Symbol Size	<i>data.seriesn.symbol.size</i>	int
Visible	<i>data.seriesn.visible</i>	boolean
Visible In Legend	<i>data.seriesn.visibleInLegend</i>	boolean

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.3 Header and Footer Properties

Chart Property	HTML Syntax	Value Type
Background	header.background footer.background	Color
Border	header.border footer.border	String <sup>a</sup>
Font	header.font footer.font	Font
Foreground	header.foreground footer.foreground	Color
Height	header.height footer.height	int
Opaque	header.opaque footer.opaque	boolean
Text	header.orientation footer.orientation	String
Visible	header.visible footer.visible	boolean
Width	header.width footer.width	int
X	header.x footer.x	int
Y	header.y footer.y	int

a. String of format `bordertype|param1|param2|...`

## B.4 JCAreaChartFormat Properties

Chart Property	HTML Syntax	Value Type
100 Percent	<i>data</i> .Area.100Percent <sup>a</sup>	boolean

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.5 JCAAnnoProperties

Chart Property	HTML Syntax	Value Type
Draw Labels	<i>[xy]axis.annon.drawLabels</i> <sup>a</sup>	boolean
Draw Ticks	<i>[xy]axis.annon.drawTicks</i>	boolean
Increment Value	<i>[xy]axis.annon.incrementValue</i>	double
Inner Extent	<i>[xy]axis.annon.innerExtent</i>	int
Label Color	<i>[xy]axis.annon.labelColor</i>	Color
Label Extent	<i>[xy]axis.annon.labelExtent</i>	int
Outer Extent	<i>[xy]axis.annon.outerExtent</i>	int
Precision	<i>[xy]axis.annon.precision</i>	int
Start Value	<i>[xy]axis.annon.startValue</i>	double
Stop Value	<i>[xy]axis.annon.stopValue</i>	double
Tick Color	<i>[xy]axis.annon.tickColor</i>	Color
Type	<i>[xy]axis.annon.tickType</i>	enum

a. *xaxis* and *yaxis* are the names of the first axes, generated when chart properties are saved to an HTML file; additional axes are named *xaxis1*, *xaxis2*, ... *xaxisn* and *yaxis1*, *yaxis2*, ... *yaxisn*.

## B.6 JCAxis X-Axes and Y-Axes Properties

Chart Property	HTML Syntax	Value Type
Annotation Method	<i>[xy]axis.annotationMethod</i> <sup>a</sup>	enum
Annotation Rotation	<i>[xy]axis.annotationRotation</i>	enum
Annotation Rotation Angle	<i>[xy]axis.annotationRotationAngle</i>	int
Annotation Visible	<i>[xy]axis.annotationVisible</i>	boolean
Drop Overlapping Labels	<i>[xy]axis.dropOverlappingLabels</i>	boolean
Editable	<i>[xy]axis.editable</i>	boolean
Font	<i>[xy]axis.font</i>	Font
Foreground	<i>[xy]axis.foreground</i>	Color

Chart Property	HTML Syntax	Value Type
Formula Constant	<i>[xy]axis</i> .formula.constant	double
Formula Multiplier	<i>[xy]axis</i> .formula.multiplier	double
Formula Originator	<i>[xy]axis</i> .formula.originator	Axis Name <sup>b</sup>
Gap	<i>[xy]axis</i> .gap	int
Grid Color	<i>[xy]axis</i> .grid.color	Color
Grid Default	<i>[xy]axis</i> .grid.default	boolean
Grid Visible	<i>[xy]axis</i> .grid.visible	boolean
Grid Spacing	<i>[xy]axis</i> .grid.spacing	double
Logarithmic	<i>[xy]axis</i> .logarithmic	boolean
Max	<i>[xy]axis</i> .max	double
Min	<i>[xy]axis</i> .min	double
Name	<i>[xy]axis</i> .name	String
Num Spacing	<i>[xy]axis</i> .numSpacing	double
Origin	<i>[xy]axis</i> .origin	double
Origin Placement	<i>[xy]axis</i> .originPlacement	enum
Placement	<i>[xy]axis</i> .placement	enum
Placement Axis	<i>[xy]axis</i> .placementAxis	Axis Name <sup>b</sup>
Placement Location	<i>[xy]axis</i> .placementLocation	double
Precision	<i>[xy]axis</i> .precision	int
Reversed	<i>[xy]axis</i> .reversed	boolean
Tick Spacing	<i>[xy]axis</i> .tickSpacing	double
Time Base	<i>[xy]axis</i> .timeBase	Date
Time Format	<i>[xy]axis</i> .timeFormat	String
Time Unit	<i>[xy]axis</i> .timeUnit	enum
Time Zone	<i>xy]axis</i> .timeZone	java.util.TimeZone
Title Adjust	<i>[xy]axis</i> .title.adjust	enum
Title Background	<i>[xy]axis</i> .title.background	Color
Title Font	<i>[xy]axis</i> .title.font	Font



Chart Property	HTML Syntax	Value Type
Title Foreground	<i>[xy]axis.title.foreground</i>	Color
Title Placement	<i>[xy]axis.title.placement</i>	enum
Title Rotation	<i>[xy]axis.title.rotation</i>	0, 90, 180, 270
Title Text	<i>[xy]axis.title.text</i>	String
Title Visible	<i>[xy]axis.title.visible</i>	boolean
Type	<i>[xy]axis.type</i>	enum
Use Anno Ticks	<i>[xy]axis.useAnnoTicks</i>	boolean
Use Default Grid	<i>[xy]axis.useDefaultGrid</i>	boolean
Use Default Labels	<i>[xy]axis.useDefaultLabels</i>	boolean
Use Default Ticks	<i>[xy]axis.useDefaultTicks</i>	boolean
Value Labels	<i>[xy]axis.valueLabels</i>	String[] (values separated by “;”)
Vertical	<i>[xy]axis.vertical</i>	boolean
Visible	<i>[xy]axis.visible</i>	boolean

a. *xaxis* and *yaxis* are the names of the first axes, generated when chart properties are saved to an HTML file; additional axes are named *xaxis1*, *xaxis2*, ... *xaxisn* and *yaxis1*, *yaxis2*, ... *yaxisn*.

b. For example, *xaxis1*.

## B.7 JCBBarChartFormat Properties

Chart Property	HTML Syntax	Value Type
100 Percent	<i>data.Bar.100Percent</i> <sup>a</sup>	boolean
Cluster Overlap	<i>data.Bar.clusterOverlap</i>	int
Cluster Width	<i>data.Bar.clusterWidth</i>	int

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.8 JCCandleChartFormat Properties

Chart Property	HTML Syntax	Value Type
Complex	<i>data</i> .Candle.Complex <sup>a</sup>	boolean

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.9 JCChart Properties

Chart Property	HTML Syntax	Value Type
Allow User Changes	allowUserChanges	boolean
Antialiasing	antiAliasing	enum
Background	background	Color
Batched	batched	boolean
Border	border	String <sup>a</sup>
Cancel Key	cancelKey	int
Customize Trigger	customizeTrigger	enum <sup>b</sup>
Depth Trigger	depthTrigger	enum <sup>b</sup>
Edit Trigger	editTrigger	enum <sup>b</sup>
Font	font	Font
Foreground	foreground	Color
Height	height	int
Label Name	label <i>n</i>	String <sup>c</sup>
Name	name	String
Opaque	opaque	boolean
Parameter File	paramFile	File <sup>d</sup>
Pick Trigger	pickTrigger	enum <sup>b</sup>
Reset Key	resetKey	int
Rotate Trigger	rotateTrigger	enum <sup>b</sup>
Translate Trigger	translateTrigger	enum <sup>b</sup>

Chart Property	HTML Syntax	Value Type
Width	width	int
Zoom Trigger	zoomTrigger	enum <sup>b</sup>

a. String of format `bordertype|param1|param2|...`

b. Valid values for any Trigger property are NONE, CTRL, SHIFT, ALT, or META (right-mouse-click).

c. `labeln` is the number of Chart Labels when chart properties are saved to HTML.

d. File from which to load additional properties.

## B.10 JCChartArea Properties

Chart Property	HTML Syntax	Value Type
Angle Unit	<code>chartArea.angleUnit</code>	enum
Axis Bounding Box	<code>chartArea.axisBoundingBox</code>	boolean
Background	<code>chartArea.background</code>	Color
Border	<code>chartArea.border</code>	String <sup>a</sup>
Depth	<code>chartArea.depth</code>	int
Elevation	<code>chartArea.elevation</code>	int
Fast Action	<code>chartArea.fastAction</code>	boolean
Font	<code>chartArea.font</code>	Font
Foreground	<code>chartArea.foreground</code>	Color
Height	<code>chartArea.height</code>	int
Horiz Action Axis	<code>chartArea.horizActionAxis</code>	Axis Name <sup>b</sup>
Insets	<code>chartArea.insets</code>	Insets
Opaque	<code>chartArea.opaque</code>	boolean
Plot Area Background	<code>chartArea.plotArea.background</code>	Color
Plot Area Bottom	<code>chartArea.plotArea.bottom</code>	int
Plot Area Foreground	<code>chartArea.plotArea.foreground</code>	Color
Plot Area Left	<code>chartArea.plotArea.left</code>	int
Plot Area Right	<code>chartArea.plotArea.right</code>	int
Plot Area Top	<code>chartArea.plotArea.top</code>	int

Chart Property	HTML Syntax	Value Type
Rotation	<code>chartArea.rotation</code>	int
Vert Action Axis	<code>chartArea.vertActionAxis</code>	Axis Name <sup>b</sup>
Visible	<code>chartArea.visible</code>	boolean
Width	<code>chartArea.width</code>	int
X	<code>chartArea.x</code>	int
Y	<code>chartArea.y</code>	int

a. String of format `bordertype|param1|param2|...`

b. For example, `xaxis1`.

## B.11 JCChartLabel Properties

Chart Property	HTML Syntax	Value Type
Anchor	<code>label<math>n</math>.anchor<sup>a</sup></code>	enum
Attach Method	<code>label<math>n</math>.attachMethod</code>	enum
Background	<code>label<math>n</math>.background</code>	Color
Connected	<code>label<math>n</math>.connected</code>	boolean
Coord	<code>label<math>n</math>.coord</code>	Point
Data Attach X	<code>label<math>n</math>.dataAttachX</code>	int
Data Attach Y	<code>label<math>n</math>.dataAttachY</code>	int
Data Index	<code>label<math>n</math>.dataIndex</code>	DataIndex Name, for example, <code>indexName</code>
Data View	<code>label<math>n</math>.dataView</code>	ChartDataView
Dwell Label	<code>label<math>n</math>.dwellLabel</code>	boolean
Font	<code>label<math>n</math>.font</code>	Font
Foreground	<code>label<math>n</math>.foreground</code>	Color
Label Name	<code>labelName<math>n</math></code>	String
Last Label Index	<code>lastLabelIndex</code>	int <sup>b</sup>
Offset	<code>label<math>n</math>.offset</code>	Font
Text	<code>label<math>n</math>.text</code>	String

Chart Property	HTML Syntax	Value Type
Visible	<code>label<math>n</math>.visible</code>	boolean

a. *label1* is the name of the first Chart Label, generated when chart properties are saved to an HTML file; additional labels are named *label2*, *label3*, ... *label $n$* .

b. The index of the last label. Used as the upper boundary on labels and data indices during load. Only needs to be explicitly specified if *n* is greater than 99.

## B.12 JCDataIndex Properties

Chart Property	HTML Syntax	Value Type
Data View	<code>index<math>n</math>.dataView<sup>a</sup></code>	ChartDataView
Distance	<code>index<math>n</math>.distance</code>	int
Index Name	<code>indexName<math>n</math></code>	String
Point	<code>index<math>n</math>.point</code>	Font
Series Index	<code>index<math>n</math>.seriesIndex</code>	int

a. *n* is the index number.

## B.13 JCGrid Properties

Chart Property	HTML Syntax	Value Type
Grid Line Cap	<code>[<math>xy</math>]axis.grid<math>n</math>.cap<sup>a</sup></code>	enum
Grid Line Color	<code>[<math>xy</math>]axis.grid<math>n</math>.color</code>	Color
Grid Line Join	<code>[<math>xy</math>]axis.grid<math>n</math>.join</code>	enum
Grid Line Pattern	<code>[<math>xy</math>]axis.grid<math>n</math>.pattern</code>	enum
Grid Line Width	<code>[<math>xy</math>]axis.grid<math>n</math>.width</code>	int
Increment Value	<code>[<math>xy</math>]axis.grid<math>n</math>.incrementValue</code>	double
Start Value	<code>[<math>xy</math>]axis.grid<math>n</math>.startValue</code>	double
Stop Value	<code>[<math>xy</math>]axis.grid<math>n</math>.stopValue</code>	double

a. *xaxis* and *yaxis* are the names of the first axes, generated when chart properties are saved to an HTML file; additional axes are named *xaxis1*, *xaxis2*, ... *xaxis $n$*  and *yaxis1*, *yaxis2*, ... *yaxis $n$* .

## B.14 JChiLoChartFormat Properties

Chart Property	HTML Syntax	Value Type
Line Color	<i>data</i> .Hilo.series <i>n</i> .line.color <sup>a</sup>	Color
Line Width	<i>data</i> .Hilo.series <i>n</i> .line.width	int

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.15 JCHLOCChartFormat Properties

Chart Property	HTML Syntax	Value Type
Line Color	<i>data</i> .HLOC.series <i>n</i> .hilo.line.color <sup>a</sup>	Color
Line Width	<i>data</i> .HLOC.series <i>n</i> .hilo.line.width	int
Open Close Full Width	<i>data</i> .HLOC.openCloseFullWidth	boolean
Showing Close	<i>data</i> .HLOC.showingClose	boolean
Showing Open	<i>data</i> .HLOC.showingOpen	boolean
Tick Size	<i>data</i> .HLOC.series <i>n</i> .tickSize	int

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.16 JCLegend Properties

Chart Property	HTML Syntax	Value Type
Anchor	legend.anchor	enum
Background	legend.background	Color
Border	legend.border	String <sup>a</sup>
Column	legend.column	int
Font	legend.font	Font
Foreground	legend.foreground	Color

Chart Property	HTML Syntax	Value Type
Height	<code>legend.height</code>	int
Item Text Alignment	<code>legend.itemTextAlignment</code>	enum
Item Text Tool Tip Enabled	<code>legend.itemTextToolTipEnabled</code>	boolean
Max Item Text Width	<code>legend.maxItemTextWidth</code>	int
Opaque	<code>legend.opaque</code>	boolean
Orientation	<code>legend.orientation</code>	enum
Truncate Mode	<code>legend.truncateMode</code>	enum
Type	<code>legend.type</code>	enum
Use Ellipsis When Truncating	<code>legend.useEllipsisWhenTruncating</code>	boolean
Visible	<code>legend.visible</code>	boolean
Width	<code>legend.width</code>	int
X	<code>legend.x</code>	int
Y	<code>legend.y</code>	int

a. String of format `bordertype|param1|param2|...`

## B.17 JCMarkeer Properties

Chart Property	HTML Syntax	Value Type
Associated With Y-Axis	<code>data.markern.associatedWithYAxis<sup>a</sup></code>	boolean
Chart Label Anchor	<code>data.markern.chartLabel.anchor</code>	enum
Chart Label Attach Method	<code>data.markern.chartLabel.attachMethod</code>	enum
Chart Label Background	<code>data.markern.chartLabel.background</code>	Color
Chart Label Border	<code>data.markern.chartLabel.border</code>	String
Chart Label Connected	<code>data.markern.chartLabel.connected</code>	boolean
Chart Label Data Attach X	<code>data.markern.chartLabel.dataAttachX</code>	int

Chart Property	HTML Syntax	Value Type
Chart Label Data Attach Y	<i>data.markern</i> .chartLabel.dataAttachY	int
Chart Label Data View	<i>data.markern</i> .chartLabel.dataView	ChartDataView
Chart Label Font	<i>data.markern</i> .chartLabel.font	Font
Chart Label Foreground	<i>data.markern</i> .chartLabel.foreground	Color
Chart Label Offset	<i>data.markern</i> .chartLabel.offset	Font
Chart Label Opaque	<i>data.markern</i> .chartLabel.opaque	boolean
Chart Label Text	<i>data.markern</i> .chartLabel.text	String
Chart Label Visible	<i>data.markern</i> .chartLabel.visible	boolean
Drawn Before Data	<i>data.markern</i> .drawnBeforeData	boolean
End Point	<i>data.markern</i> .endPoint	double
Has Chart Label	<i>data.markern</i> .hasChartLabel	boolean
Included in Data Bounds	<i>data.markern</i> .includedInDataBounds	boolean
Label	<i>data.markern</i> .label	String
Line Color	<i>data.markern</i> .line.color	Color
Line Pattern	<i>data.markern</i> .line.pattern	enum
Line Width	<i>data.markern</i> .line.width	int
Start Point	<i>data.markern</i> .startPoint	double
Value	<i>data.markern</i> .value	double
Visible in Legend	<i>data.markern</i> .visibleInLegend	boolean

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.18 JCMultiColLegend Properties

Chart Property	HTML Syntax	Value Type
Number of Columns	legend.numCols	int
Number of Rows	legend.numRows	int



## B.19 JCPieChartFormat Properties

Chart Property	HTML Syntax	Value Type
Explode Offset	<i>data</i> .Pie.explodeOffset <sup>a</sup>	int
Min Slices	<i>data</i> .Pie.minSlices	int
Other Fill Background	<i>data</i> .Pie.other.fill.background	enum
Other Fill Color	<i>data</i> .Pie.other.fill.color	Color
Other Fill Color Index	<i>data</i> .Pie.other.fill.colorIndex	int
Other Fill Image	<i>data</i> .Pie.other.fill.image	Image
Other Fill Image File Name	<i>data</i> .Pie.other.fill.image.fileName	String
Other Fill Image File Access	<i>data</i> .Pie.other.fill.image.fileAccess	String
Other Fill Pattern	<i>data</i> .Pie.other.fill.pattern	enum
Other Label	<i>data</i> .Pie.other.label	String
Sort Order	<i>data</i> .Pie.sortOrder	ASCENDING, DESCENDING
Start Angle	<i>data</i> .Pie.startAngle	double
Threshold Method	<i>data</i> .Pie.thresholdMethod	enum
Threshold Value	<i>data</i> .Pie.thresholdValue	int

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.20 JCPolarRadarChartFormat Properties

Chart Property	HTML Syntax	Value Type
HalfRange	<i>data</i> .PolarRadar.halfRange <sup>a</sup>	boolean
OriginBase	<i>data</i> .PolarRadar.originBase	double
RadarCircularGrid	<i>data</i> .PolarRadar.radarCircularGrid	boolean
YAxisAngle	<i>data</i> .PolarRadar.yAxisAngle	double

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *datan*.

## B.21 JCThreshold Properties

Chart Property	HTML Syntax	Value Type
Associated With Y-Axis	<i>data.threshold</i> <i>n</i> .associatedWithYAxis <sup>a</sup>	boolean
End Line Color	<i>data.threshold</i> <i>n</i> .endLine.color	Color
End Line Pattern	<i>data.threshold</i> <i>n</i> .endLine.pattern	enum
End Line Width	<i>data.threshold</i> <i>n</i> .endLine.width	int
End Value	<i>data.threshold</i> <i>n</i> .endValue	double
Fill Color	<i>data.threshold</i> <i>n</i> .fill.color	Color
Fill Image	<i>data.threshold</i> <i>n</i> .fill.image	Image
Fill Image Name	<i>data.threshold</i> <i>n</i> .fill.image.fileName	String
Fill Image Access Type	<i>data.threshold</i> <i>n</i> .fill.image.fileAccess	String
Fill Pattern	<i>data.threshold</i> <i>n</i> .fill.pattern	enum
Has End Line Style	<i>data.threshold</i> <i>n</i> .hasEndLineStyle	boolean
Has Start Line Style	<i>data.threshold</i> <i>n</i> .hasStartLineStyle	boolean
Included in Data Bounds	<i>data.marker</i> <i>n</i> .includedInDataBounds	boolean
Label	<i>data.threshold</i> <i>n</i> .label	String
Start Line Color	<i>data.threshold</i> <i>n</i> .startLine.color	Color
Start Line Pattern	<i>data.threshold</i> <i>n</i> .startLine.pattern	enum
Start Line Width	<i>data.threshold</i> <i>n</i> .startLine.width	int
Start Value	<i>data.threshold</i> <i>n</i> .startValue	double
Visible in Legend	<i>data.threshold</i> <i>n</i> .visibleInLegend	boolean

a. *data* is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named *data1*, *data2*, ... *data**n*.



## XML DTD

*Chart.dtd* ■ *JCChartData.dtd*

This appendix describes the DTDs available with JClass Chart. For more information, see Chapter 12, [Loading and Saving Charts Using XML](#).

JClass Chart has two DTDs: *Chart.dtd* and *JCChartData.dtd*. In general, the elements, sub-elements, and attributes coincide with the objects, sub-objects, and properties within the chart component.

You should find that the values and types expected for each attribute are, for the most part, straightforward and easy to understand. However, there are a few types that require further explanation:

- **Font:** You specify Font types in the format “name-style-size”, where style is one of plain, bold, italic, or bolditalic. For example, the following syntax specifies a plain, 10 point Helvetica font: Helvetica-plain-10
- **Color:** You specify Color types in one of three ways: as a hexadecimal (#RRGGBB), as an RGB value (RRR-GGG-BBB), or as a color enum (such as black or white).
- **JCAxis:** JCAxis values correspond to the name of an axis. You specify JCAxis values using the value of the name attribute from the desired <axis> tag in the XML file.
- **ChartDataView:** ChartDataView values correspond to the name of a data view. You specify ChartDataView values using the value of the name attribute from the desired <chart-data-view> element in the XML file.

The next two sections describe the contents of the DTDs.

# C.1 Chart.dtd

When using the *Chart.dtd*, the values of any properties that remain unspecified in the XML file remain unchanged when the XML file is applied to the chart. Properties are specified as strings in the XML file, and are converted to the appropriate type by the chart's XML handler.

The following is a list of each of the *Chart.dtd* elements, as well as their descriptions.

## C.1.1 chart

**Purpose:**

The main chart element. The properties and sub-elements by and large coincide with the properties and sub-objects of the corresponding object within the chart component.

**Equivalent in  
JClass Chart:** JCChart

- Sub-Elements:**
- chart-area
  - chart-data-view
  - chart-label
  - component
  - event-trigger
  - external-java-code
  - footer
  - header
  - key
  - legend
  - locale

**Attributes:**

Name	Definition	Possible Values or Type
allowUserChanges	Determines whether or not the user viewing the graph can modify graph values.	boolean
antiAliasing	Determines if anti-aliasing is enabled. Default is Default.	■ Default ■ On ■ Off
height	Determines the height of the chart. If not specified, the height remains unchanged.	int
name	Specifies a string identifier for the chart.	String

Name	Definition	Possible Values or Type
width	Determines the width of the chart. If not specified, the width remains unchanged.	int

### C.1.2 anno

**Purpose:**

Controls the chart annotation marks.

**Equivalent in JClass Chart:** JCAнно

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
drawLabels	Determines if labels are drawn.	boolean
drawTicks	Determines if tick marks are drawn.	boolean
incrementValue	Specifies the increment between ticks or labels	double
innerExtent	Specifies the pixel extent of tick marks into the plot area.	int
labelColor	Specifies the color of labels.	Color
labelExtent	Specifies the pixel distance of labels from the axis.	int
outerExtent	Specifies the pixel extent of tick marks away from the plot area.	int
precision	Specifies the precision to which numeric labels are displayed.	int
startValue	Specifies the start value of a tick or label.	double
stopValue	Specifies the end value of a tick or label.	double
tickColor	Specifies the color of the tick marks.	Color

Name	Definition	Possible Values or Type
type	Determines the tick object's type. Default_Labels defines the anno object to be the set of default labels for the axis. Default_Ticks defines the anno object to be the set of default ticks for the axis. Default is User_Defined.	<ul style="list-style-type: none"> <li>■ Default_Labels</li> <li>■ Default_Ticks</li> <li>■ User_Defined</li> </ul>

### C.1.3 area-format

**Purpose:**

Properties specific to area charts.

**Equivalent in** JCAreaChartFormat

**JClass Chart:**

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
percent100 <sup>1</sup>	Determines whether a stacking area will be charted versus an axis representing a percentage between 0 and 100.	boolean

<sup>1</sup>Equivalent property is 100percent.

### C.1.4 axis

**Purpose:**

Has attributes that deal with drawing of x-axes and y-axes. Note that the `placementAxis` name needs to match an axis of the opposite type (i.e. for an x-axis the `placementAxis` needs to match a y-axis name). If no match occurs, the default “xaxis” or “yaxis” is assumed.

**Equivalent in  
JClass Chart:**

JCAxis

**Sub-Elements:**

- anno
- grid
- axis-formula
- axis-title
- chart-interior-region
- line-style
- value-label

**Attributes:**

Name	Definition	Possible Values or Type
annotationMethod	Determines the type of axis annotation.	■ Value ■ Value_Labels ■ Point_Labels ■ Time_Labels
annotationRotation	Determines the rotation of each axis label.	■ None ■ Rotate_90 ■ Rotate_180 ■ Rotate_270 ■ Rotate_Other
annotationRotationAngle	Specifies the angle of the annotation for the currently selected axis when the <code>annotationRotation</code> property is set to <code>Rotate_Other</code> . The angle is always set in degrees.	int
annotationVisible	Determines whether or not the annotation is visible.	boolean
dropOverlappingLabels	Determines whether or not overlapping labels are dropped. When true, one or more of the overlapping labels are eliminated so that the remaining labels do not overlap.	boolean

Name	Definition	Possible Values or Type
editable	Determines whether or not the axis can be affected by editing, translating, or zooming.	boolean
gap	Determines the amount of space left between adjacent axis annotations, in pixels.	int
gridDefault	Determines whether or not grid lines are drawn at the labels.	boolean
gridSpacing	Controls the spacing between gridlines relative to the axis.	double
gridVisible	Determines whether or not a grid is drawn for the axis.	boolean
logarithmic	Determines whether or not the axis will be logarithmic.	boolean
max	Controls the maximum value shown on the axis.	double
min	Controls the minimum value shown on the axis.	double
name	Specifies a string identifier.	String
numSpacing	Controls the interval between axis labels.	double
origin	Controls the location of the origin along the axis.	double
originPlacement	Determines where the origin is placed. Note that it is only active if origin is not set.	<ul style="list-style-type: none"> <li>■ Automatic</li> <li>■ Zero</li> <li>■ Min</li> <li>■ Max</li> </ul>
placement	Determines the method used to place the axis with respect to the placementAxis.	<ul style="list-style-type: none"> <li>■ Automatic</li> <li>■ Max</li> <li>■ Min</li> <li>■ Origin</li> <li>■ Value_Anchored</li> </ul>



Name	Definition	Possible Values or Type
placementAxis	Determines the axis that controls the placement of this axis.	JCAxis
placementLocation	Positions the current axis object at a particular point on another axis. Used in conjunction with placementAxis.	double
precision	Controls the number of zeros that appear after the decimal place in chart-generated axis labels.	int
reversed	Determines if the axis direction is reversed (true) or not (false).	boolean
tickSpacing	Controls the interval between tick lines on the axis.	double
timeBase	<p>Defines the start time for the axis.</p> <p>To use the timeBase attribute in the XML file, one must specify the attribute's value to be a data String (for example, Feb 21, 2003 10:11:07 AM EST). The com.klg.jclass.util.JCTypeConverter.toDate() method converts the String into a java.util.Date object, using a standard java.text.SimpleDateFormat object to parse it. The object is then set on the axis using the JCAxis.setTimeBase() method.</p>	java.util.Date
timeFormat	<p>Controls the format used to generate time labels for time labelled axes.</p> <p>To use the timeFormat attribute, one must specify the attribute's value to be a String set directly on the axis using the JCAxis.setTimeFormat() method. For more information, see <a href="#">Time Format</a>, in Chapter 6.</p>	String

Name	Definition	Possible Values or Type
timeUnit	<p>Controls the unit of time used for labelling a time labelled axis.</p> <p>To use the <code>timeUnit</code> attribute in the XML file, one must specify the attribute's value to be an enum value. The String is converted into an enum value, then set on the axis using the <code>JCAxis.setTimeUnit()</code> method.</p>	<ul style="list-style-type: none"> <li>■ Seconds</li> <li>■ Minutes</li> <li>■ Hours</li> <li>■ Days</li> <li>■ Weeks</li> <li>■ Months</li> <li>■ Years</li> </ul>
timeZone	Specifies the time zone for this axis. Use only for time-based labels.	<code>java.util. TimeZone</code>
type	Determines which axis is being defined. Default is <code>XAxis</code> .	<ul style="list-style-type: none"> <li>■ <code>XAxis</code></li> <li>■ <code>YAxis</code></li> </ul>
useAnnoTicks	<p>Determines if the ticks specified by <code>JCAAnno</code> objects are drawn. This requires that a non-default <code>tickSpacing</code> is used, and that the <code>annotationMethod</code> is one of the following:</p> <ul style="list-style-type: none"> <li>■ <code>JCAxis.VALUE_LABEL</code></li> <li>■ <code>JCAxis.POINT_LABEL</code></li> <li>■ <code>JCAxis.TIME_LABEL</code></li> </ul>	boolean
useDefaultGrid	Determines if the default <code>JCGrid</code> is automatically added to the axis.	boolean
useDefaultLabels	Determines if the default <code>JCAAnno</code> label object is automatically added to the axis.	boolean
useDefaultTicks	Determines if the default <code>JCAAnno</code> tick objects is automatically added to the axis.	boolean
vertical	Determines whether or not the axis is vertical.	boolean

### C.1.5 axis-formula

**Purpose:**

Defines a relationship between two axes. The originator attribute needs to match an existing axis name of the same axis type. If no match occurs, the default “xaxis” or “yaxis” is used.

**Equivalent in** JCAxisFormula  
**JClass Chart:**

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
constant	Specifies the “c” value in the relationship $y2 = m * y + c.$	double
multiplier	Specifies the “m” value in the relationship $y2 = m * y + c.$	double
originator	Specifies an object representing the axis that is related to the current axis by the formula $y2 = m * y + c.$ The originator is “y”.	JCAxis

### C.1.6 axis-title

**Purpose:**

The title for an axis.

**Equivalent in** JCAxisTitle  
**JClass Chart:**

**Sub-Elements:** chart-interior-region

**Attributes:**

Name	Definition	Possible Values or Type
adjust	Determines how text is justified (positioned) in the label.	■ Left ■ Center ■ Right

Name	Definition	Possible Values or Type
placement	Controls where the axis title is placed relative to the “opposing” axis.	<ul style="list-style-type: none"> <li>■ East</li> <li>■ North</li> <li>■ Northeast</li> <li>■ Northwest</li> <li>■ South</li> <li>■ Southeast</li> <li>■ Southwest</li> <li>■ West</li> </ul>
rotation	Controls the rotation of the label.	<ul style="list-style-type: none"> <li>■ None</li> <li>■ Rotate_90</li> <li>■ Rotate_180</li> <li>■ Rotate_270</li> </ul>
text	A string that represents the text to be displayed inside the chart label.	String

### C.1.7 bar-format

**Purpose:**

Properties specific to bar and stacking bar charts.

**Equivalent in JClass Chart:** JCBarchartFormat

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
clusterOverlap	Specifies the overlap between bars.	int
clusterWidth	Determines the percentage of available space which will be occupied by the bars.	int
percent100 <sup>1</sup>	Determines whether a stacking bar will be charted versus an axis representing a percentage between 0 and 100.	boolean

<sup>1</sup>Equivalent property is 100percent.

### C.1.8 bevel-border

**Purpose:**

Bevel or SoftBevel borders.

**Equivalent in JClass Chart:** javax.swing.border.BevelBorder or javax.swing.border.SoftBevelBorder

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
highlightColor	The color to use for the bevel highlight.	Color
shadowColor	The color to use for the bevel shadow.	Color
soft	If true, this element represents a Bevel Border; otherwise, a SoftBevel Border.	boolean
type	The bevel type.	■ Raised ■ Lowered

### C.1.9 candle-format

**Purpose:**

Properties specific to candle charts.

**Equivalent in JClass Chart:** JCCandleChartFormat

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
complex	Determines whether candle charts use the simple or complex display style.	boolean

C.1.10 chart-area

Purpose:

This is the component within the chart in which the actual chart is drawn. Note that the `horizActionAxis` and `vertActionAxis` properties must match an axis name within the axis list or they will default to the primary x-axis or y-axis.

Equivalent in JClass Chart: JCCChartArea

- Sub-Elements:
- component
  - layout-hints
  - plot-area
  - axis

Attributes:

Name	Definition	Possible Values or Type
angleUnit	Determines the unit of all angle values.	■ Degrees ■ Grads ■ Radians
axisBoundingBox	Determines whether or not a box is drawn around the area bound by the inner axes.	boolean
depth	Controls the apparent depth of a graph.	int
elevation	Controls distance from the x-axes.	int
fastAction	Determines whether chart actions will use an optimized mode in which it does not bother to display axis annotations or gridlines.	boolean
horizActionAxis	Determines the axis used for actions (zooming, translating) in the horizontal direction.	JCAxis
rotation	Controls the position of the eye relative to the y-axis.	int
vertActionAxis	Determines the axis used for actions (zooming, translating) in the vertical direction.	JCAxis

### C.1.11 chart-data-file

**Purpose:**

Data file. If the data is in a file, use this element to reference it.

**Equivalent in JClass Chart:**

BaseDataSource or other class that implements the ChartDataModel.

**Sub-Elements:**

none

**Attributes:**

Name	Definition	Possible Values or Type
fileAccess	Determines how to interpret the fileName. For more information on the values, see <a href="#">Overview of the LoadProperties Class</a> , in Chapter 10. Default is Default.	<ul style="list-style-type: none"><li>■ Default</li><li>■ Absolute</li><li>■ Resolving_Class</li><li>■ Url</li><li>■ Relative_Url</li><li>■ Servlet</li></ul>
fileCharset	The character set to use when reading data.	String
fileName	The name of the data file.	String
fileType	Determines whether the data is written in the standard chart text format or as XML. Default is Xml.	<ul style="list-style-type: none"><li>■ Text</li><li>■ Xml</li></ul>

C.1.12 chart-data-view

**Purpose:**  
Contains a representation of chartable data and attributes used to draw this data, like per series drawing information, chart type, and various chart type specific properties.

**Equivalent in JClass Chart:** ChartDataView

- Sub-Elements:**
- area-format
  - bar-format
  - candle-format
  - chart-data
  - JCChartData (only if the data is specified using the old data XML format)
  - chart-data-file
  - chart-data-view-series
  - hi-lo-open-close-format
  - line-style
  - marker
  - pie-format
  - polar-radar-format
  - point-label
  - threshold

**Attributes:**

Name	Definition	Possible Values or Type
autoLabel	Determines if the chart automatically generates labels for each point in each series.	boolean
bufferPlotData	Controls whether plot data is to be buffered to speed up the drawing process. Only applicable for plot, scatter plot, area, Hi-Lo, Hi-Lo-Open-Close, and candle chart types.	boolean



Name	Definition	Possible Values or Type
chartType	Specifies the type of chart used to plot the data. Default is Plot.	<ul style="list-style-type: none"> <li>■ Area</li> <li>■ Area_Radar</li> <li>■ Stacking_Area</li> <li>■ Bar</li> <li>■ Stacking_Bar</li> <li>■ Plot</li> <li>■ Scatter_Plot</li> <li>■ Pie</li> <li>■ Hi_Lo</li> <li>■ Hi_Lo_Open_Close</li> <li>■ Candle</li> <li>■ Polar</li> <li>■ Radar</li> </ul>
drawFrontPlane	Determines whether a data view that has both axes on the front plane of a 3d chart will draw on the front or back place of that chart. If true, it will draw on the front plane; if false, it will draw on the back plane. If either axis associated with the data view is on the back plane, this property will be ignored and the data view will automatically be drawn on the back plane. This property is also ignored for 3d chart types such as bar and stacking bars that automatically appear on the front plane.	boolean
fastUpdate	Controls whether column appends to the data are performed quickly, only recalculating and redrawing the newly-appended data.	boolean
holeValue	A floating point number that represents a hole in the data.	double
inverted	If set to true, the x-axis becomes vertical and the y-axis becomes horizontal.	boolean
name	Used as an index for referencing particular chart-data-view objects.	String
visible	Determines whether or not the dataview is showing.	boolean
visibleInLegend	Determines whether or not the view name and its series will appear in the chart legend.	boolean

Name	Definition	Possible Values or Type
xaxis	Determines the x-axis against which the data is plotted.	JCAxis
yaxis	Determines the y-axis against which the data is plotted.	JCAxis

### C.1.13 chart-data-view-series

**Purpose:**

Information on how and what to draw for a specific data series.

**Equivalent in JClass Chart:**

ChartDataViewSeries

**Sub-Elements:**

- chart-style
- hole-style

**Attributes:**

Name	Definition	Possible Values or Type
firstPoint	Controls the index of the first point displayed in the chart-data-view-series.	int
included	Determines whether a data series is included in chart calculations.	boolean
label	Controls the text that appears next to the data series inside the legend.  <b>Note:</b> The <data-series-label> fulfills the same purpose as setting the <chart-data-view-series> tag's label attribute. If both are used, the <data-series-label> tag is ignored.	String
lastPoint	Controls the index of the first point displayed in the chart-data-view-series.	int
name	Specifies the name of this data series.	String

Name	Definition	Possible Values or Type
visible	Determines whether or not the data series is showing in the chart area. Note that data series that are not showing are still used in axis calculations. See <code>included</code> for details on how to omit a data series from chart calculations.	boolean
visibleInLegend	Determines whether or not this series will appear in the chart legend.	boolean

### C.1.14 **chart-interior-region**

**Purpose:**

A rectangular region for drawing within chart that is not a component.

**Equivalent in JClass Chart:**

`ChartInteriorRegion`

**Sub-Elements:**

`insets`

**Attributes:**

Name	Definition	Possible Values or Type
background	Determines the background color of the drawing region.	Color
font	Determines the font used to render text inside the drawing region.	Font
foreground	Determines the foreground color used to draw inside the drawing region.	Color
groupingUsed	Determines whether or not grouping will be used in formatting numbers.	boolean
height	Defines the height of the drawing region.	int
left	Determines the location of the left of the drawing region.	int
numberLocalization	Determines whether or not numbers are localized.	boolean

Name	Definition	Possible Values or Type
top	Determines the location of the top of the drawing region.	int
visible	Determines whether or not the chart interior region is visible.	boolean
width	Defines the width of the drawable region.	int

### C.1.15 chart-label

**Purpose:**

A floating label that is attached somewhere to the chart. Note that if the `dataView` property is missing or does not match the `name` property of an existing `ChartDataView` element, it is assumed that the chart label is associated with the first (or primary) `ChartDataView`.

**Equivalent in JClass Chart:** `JCChartLabel`

- Sub-Elements:**
- `label`
  - `offset`
  - `coord`
  - `data-coord`
  - `data-index`

**Attributes:**

Page	Definition	Possible Values or Type
anchor	Specifies how the label is to be positioned relative to its attach point.	<ul style="list-style-type: none"> <li>■ North</li> <li>■ South</li> <li>■ East</li> <li>■ West</li> <li>■ Northeast</li> <li>■ Northwest</li> <li>■ Southeast</li> <li>■ Southwest</li> <li>■ Center</li> <li>■ Auto</li> </ul>
attachMethod	Specifies how the label is attached to the chart.	<ul style="list-style-type: none"> <li>■ None</li> <li>■ Coord</li> <li>■ Data_Coord</li> <li>■ Data_Index</li> </ul>

Page	Definition	Possible Values or Type
connected	Determines whether or not there is a line connecting the label to its attach point.	boolean
dataView	Specifies which data view should be used for the chart label's Data_Coord attachment. The dataView for the Data_Index attachment is specified on the data-index tag.	ChartDataView
dwellLabel	When set to true, the label is only displayed when the cursor is over the point/bar/slice that the label is attached to. When set to false (the default), the label is always displayed.	boolean

### C.1.16 chart-style

#### Purpose:

Drawing specific properties of a series. Depending on the chart type, different aspects of the JCLineStyle, JCFillStyle and JCSymbolStyle are used.

**Equivalent in JClass Chart:** JCChartStyle

**Sub-Elements:**

- line-style
- fill-style
- symbol-style

**Attributes:** none

**C.1.17    component**

**Purpose:**  
Element which contains component properties. This element is a sub-element of those elements which represent components.

**Equivalent in JClass Chart:**                    javax.swing.JComponent

**Sub-Elements:**                    (empty-border|bevel-border|etched-border|  
line-border|matte-border|titled-border|  
compound-border)

**Attributes:**

Name	Definition	Possible Values or Type
background	Determines the background color of the component region.	Color
font	Determines the font used to render text inside the component region.	Font
foreground	Determines the foreground color used to draw inside the component region.	Color
opaque	An opaque component paints every pixel within its rectangular bounds. A non-opaque component paints only a subset of its pixels or none at all, allowing the pixels underneath to “show through”.	boolean
visible	Determines whether or not the legend is currently visible.	boolean

### C.1.18 compound-border

**Purpose:**

Specifies an outside border and inside border. Note that these sub-borders can also be of the type `compound-border`.

**Equivalent in JClass Chart:** `javax.swing.border.CompoundBorder`

**Sub-Elements:**

- (empty-border|bevel-border|etched-border|line-border|matte-border|titled-border|compound-border)
- (empty-border|bevel-border|etched-border|line-border|matte-border|titled-border|compound-border)

**Attributes:** none

### C.1.19 coord

**Purpose:**

Given a `chart-label` whose `attachMethod` attribute is `Coord`, this is the chart coordinate (in pixels) to which the label is attached.

**Equivalent in JClass Chart:** `java.awt.Point`

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
x	The X coordinate.	int
y	The Y coordinate.	int

C.1.20 data-coord

**Purpose:**  
Given a chart-label whose attachMethod attribute is DataCoord, this is the coordinate in data space to which the label is attached.

**Equivalent in JClass Chart:** JCDataCoord

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
x	The x-value.	double
y	The y-value.	double

C.1.21 data-index

**Purpose:**  
Given a chart-label whose attachMethod attribute is DataIndex, gives the data index that the label is attached to.

**Equivalent in JClass Chart:** JCDataIndex

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
dataView	The name of the data view that is represented by this data point.	String
point	The point index within the series.	int
series	The index of the series within the dataview. If this attribute represents the “other” slice of a pie chart, its value is Other_Slice.	int or Other_Slice



### C.1.22 empty-border

**Purpose:**  
Empty borders.

**Equivalent in JClass Chart:** javax.swing.border.EmptyBorder

**Sub-Elements:** insets

**Attributes:** none

### C.1.23 end-line-style

**Purpose:**  
A line style for the ending value boundary line for thresholds.

**Equivalent in JClass Chart:** JLineStyle

**Sub-Elements:** line-style

**Attributes:** none

### C.1.24 etched-border

**Purpose:**  
Etched borders.

**Equivalent in JClass Chart:** javax.swing.border.EtchedBorder

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
highlightColor	The color to use for the etched highlight.	Color
shadowColor	The color to use to the etched shadow.	Color
type	The type of etch.	■ Raised ■ Lowered

**C.1.25 event-trigger**

**Purpose:**  
Event triggers are used to assign predefined chart actions such as rotate the chart or popup the customized to mouse events.

**Equivalent in JClass Chart:** EventTrigger

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
modifier	The modifier key that needs to be pressed along with the mouse click for this event trigger. Default is None.	■ None ■ Ctrl ■ Shift ■ Alt ■ Meta
trigger	The type of action that gets triggered by this event trigger. Default is Customize.	■ Rotate ■ Zoom ■ Translate ■ Edit ■ Pick ■ Pick_Series ■ Depth ■ Customize

**C.1.26 explode-list**

**Purpose:**  
A list of (series, point) pairs that represent which slices on a pie have been exploded.

**Equivalent in JClass Chart:** none

**Sub-Elements:** series-point

**Attributes:** none

### C.1.27 external-java-code

**Purpose:**

Specifies a Java class that will be created and called between the creation of a chart via XML and the return of control to the calling code. This can be used for encapsulating chart settings that cannot be set with XML (for example, populating the chart with data from a database query).

The Java class must contain an empty constructor, as well as implement the `com.klg.jclass.util.property.xml.ExternalCodeHandler` interface. The `ExternalCodeHandler` interface specifies a method named `handle()` that is called by the JClass Chart XML parser. The contents of the body of the tag will be passed to the `handle()` method in the `userData` parameter. The value of the `UserObject` property in the current `LoadProperties` class will be passed to the `handle()` method when it is called. For more information on `LoadProperties`, see Section 12.3, [Creating a Chart Using XML](#).

**Note:** When a chart that was created via XML is later saved to XML, the contents of this tag are not written out.

**Equivalent in JClass Chart:** n/a

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
class	Fully qualified name of the class that will be created and called by the JClass Chart parser.	String

### C.1.28 fill-style

**Purpose:**

Fill specific drawing properties.

**Equivalent in JClass Chart:** JCFillStyle

**Sub-Elements:** image-file

Attributes:

Name	Definition	Possible Values or Type
background	Determines the background color used when painting patterned fills.	Color
color	Determines the color used to fill regions in chart.	Color
image	<i>Deprecated.</i> See the image-file element. Determines the image used to paint the fill region. Either a file name or a URL.	<i>Deprecated.</i> String or java.net.URL
pattern	Determines the fill pattern used to fill regions in chart. Default is Solid.	<ul style="list-style-type: none"><li>■ None</li><li>■ Solid</li><li>■ Per_25</li><li>■ Per_50</li><li>■ Per_75</li><li>■ Horiz_Stripe</li><li>■ Vert_Stripe</li><li>■ Stripe_45</li><li>■ Stripe_135</li><li>■ Diag_Hatched</li><li>■ Cross_Hatched</li><li>■ Custom_Fill</li><li>■ Custom_Stacked</li></ul>

C.1.29 footer

**Purpose:**  
The component used as the footer for the chart.

**Equivalent in JClass Chart:** javax.swing.JLabel

**Sub-Elements:**

- component
- layout-hints

**Attributes:**

Name	Definition	Possible Values or Type
horizontalAlignment	Determines the horizontal alignment. Default is Leading.	■ Left ■ Center ■ Right ■ Leading ■ Trailing
text	A string property that represents the text to be displayed in the footer.	String
verticalAlignment	Determines the vertical alignment. Default is Center.	■ Top ■ Center ■ Bottom

**C.1.30 grid****Purpose:**

Specifies the properties that are specific to the chart's grids.

**Equivalent in** JCGrid  
**JClass Chart:**

**Sub-Elements:** line-style

**Attributes:**

Name	Definition	Possible Values or Type
incrementValue	Specifies the increment between grid lines.	double
startValue	Specifies the start value for grid lines.	double
stopValue	Specifies the end value for grid lines.	double

**C.1.31 header**

**Purpose:**  
The component used as the header for the chart.

**Equivalent in JClass Chart:** `javax.swing.JLabel`

**Sub-Elements:**

- `component`
- `layout-hints`

**Attributes:**

Name	Definition	Possible Values or Type
<code>horizontalAlignment</code>	Determines the horizontal alignment. Default is <code>Leading</code> .	<ul style="list-style-type: none"><li>■ <code>Left</code></li><li>■ <code>Center</code></li><li>■ <code>Right</code></li><li>■ <code>Leading</code></li><li>■ <code>Trailing</code></li></ul>
<code>text</code>	A string property that represents the text to be displayed in the footer.	String
<code>verticalAlignment</code>	Determines the vertical alignment. Default is <code>Center</code> .	<ul style="list-style-type: none"><li>■ <code>Top</code></li><li>■ <code>Center</code></li><li>■ <code>Bottom</code></li></ul>

**C.1.32 hi-lo-open-close-format**

**Purpose:**  
Properties specific to Hi-Lo-Open-Close charts.

**Equivalent in JClass Chart:** `JCHLOCCartFormat`

**Sub-Elements:** `none`

**Attributes:**

Name	Definition	Possible Values or Type
<code>openCloseFullWidth</code>	Indicated whether the open and close tick indications are drawn across the full width of the Hi-Lo bar or just on one side.	boolean

Name	Definition	Possible Values or Type
showingClose	Indicates whether or not the close tick indication is shown. The tick appears to the right of the Hi-Lo line.	boolean
showingOpen	Indicates whether or not the open tick indication is shown. The tick appears to the left of the Hi-Lo line.	boolean

### C.1.33 hole-style

**Purpose:**

Chart style used to draw hole values.

**Equivalent in JClass Chart:** JCChartStyle

**Sub-Elements:**

- line-style
- fill-style
- symbol-style

**Attributes:** none

### C.1.34 image-file

**Purpose:**

Specifies an image from an external source.

**Equivalent in JClass Chart:** JCFillStyle.setImage(String filename)

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
fileName	The name of the image file.	String

Name	Definition	Possible Values or Type
fileAccess	Determines how to interpret the fileName. For more information, see <a href="#">Overview of the LoadProperties Class</a> , in Chapter 10. Default is Default.	<ul style="list-style-type: none"> <li>■ Default</li> <li>■ Absolute</li> <li>■ Resolving_Class</li> <li>■ Url</li> <li>■ Relative_Url</li> <li>■ Servlet</li> </ul>

**C.1.35 insets**

**Purpose:**  
A representation of the borders of a container.

**Equivalent in JClass Chart:**                java.awt.Insets

**Sub-Elements:**                none

**Attributes:**

Name	Definition	Possible Values or Type
bottom	The bottom margin.	int
left	The left margin.	int
right	The right margin.	int
top	The top margin.	int



### C.1.36 key

**Purpose:**

Key elements allow a key to be bound to reset or cancel actions.

**Equivalent in  
JClass Chart:** none

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
keyValue	The key that will trigger the action.	int
type	The type of action.	■ ResetKey ■ CancelKey

### C.1.37 label

**Purpose:**

The label component of a chart label.

**Equivalent in  
JClass Chart:** javax.swing.JLabel

**Sub-Elements:** component

**Attributes:**

Name	Definition	Possible Values or Type
horizontalAlignment	Determines the horizontal alignment for the label text. Default is <code>Leading</code> .	■ Left ■ Center ■ Right ■ Leading ■ Trailing
text	A string property that represents the text to be displayed in the chart label.	String
verticalAlignment	Determines the vertical alignment for the label text. Default is <code>Center</code> .	■ Top ■ Center ■ Bottom

**C.1.38    layout-hints**

**Purpose:**  
A rectangle that specifies where and at what size subcomponents of chart such as the header, footer, legend, and chart area are drawn. The chart calculates default layout rectangles for each subcomponent. Supplying layout hints allows the user to override some or all of the default values. If some attributes are not specified, they will be set to their default values.

**Equivalent in**                      `java.awt.Rectangle`  
**JClass Chart:**

**Sub-Elements:**              none

**Attributes:**

Name	Definition	Possible Values or Type
height	Determines the height of the subcomponent.	int
width	Determines the width of the subcomponent.	int
x	Determines the X position of the subcomponent.	int
y	Determines the Y position of the subcomponent.	int

**C.1.39    legend**

**Purpose:**  
This is the component within the chart in which the legend is drawn.

**Equivalent in**                      `com.klg.jclass.util.legend.JCLegend`  
**JClass Chart:**

**Sub-Elements:**

- component
- layout-hints
- legend-column
- multi-col

**Attributes:**

Name	Definition	Possible Values or Type
anchor	Determines the position of the legend relative to the chart.	■ North ■ South ■ East ■ West ■ Northeast ■ Northwest ■ Southeast ■ Southwest
itemTextToolTip Enabled	Determines whether or not tooltips are displayed when the mouse hovers over a legend item. This is useful when the legend text has been truncated.	boolean
orientation	Determines how legend information is laid out.	■ Horizontal ■ Vertical
type	Determines the legend type.	■ Grid ■ MultiCol
useEllipsisWhen Truncating	Determines whether or not an ellipsis is used to indicate truncated legend text.	boolean

**C.1.40 legend-column****Purpose:**

Defines column attributes for the legend defined by the `legend` tag.

**Equivalent in  
JClass Chart:**

`com.klg.jclass.util.legend.LegendColumn`

**Sub-Elements:**

none

**Attributes:**

Name	Definition	Possible Values or Type
column	Specifies a column within the legend to which the other attributes in this element are applied. If omitted, the other <code>legend-column</code> attributes apply to all columns in the legend.	int

Name	Definition	Possible Values or Type
<code>itemTextAlignment</code>	Determines the alignment for the text in a column. Default is <code>Leading</code> .	<ul style="list-style-type: none"> <li>■ <code>Left</code></li> <li>■ <code>Center</code></li> <li>■ <code>Right</code></li> <li>■ <code>Leading</code></li> <li>■ <code>Trailing</code></li> </ul>
<code>maxItemTextWidth</code>	Specifies the maximum width of the column in pixels. If the column text exceeds this width, the text is truncated.	<code>int</code>
<code>truncateMode</code>	Determines how text is truncated when the length of the text exceeds the maximum width of the column. Default is <code>Trailing</code> .	<ul style="list-style-type: none"> <li>■ <code>Left</code></li> <li>■ <code>Right</code></li> <li>■ <code>Middle</code></li> <li>■ <code>End</code></li> <li>■ <code>Leading</code></li> <li>■ <code>Trailing</code></li> </ul>

### C.1.41 **line-border**

**Purpose:**

Line borders.

**Equivalent in JClass Chart:**

`javax.swing.border.LineBorder`

**Sub-Elements:**

none

**Attributes:**

Name	Definition	Possible Values or Type
<code>color</code>	Determines the color of the border.	<code>Color</code>
<code>roundedCorners</code>	Determines if the border corners will be straight or rounded.	<code>boolean</code>
<code>thickness</code>	Determines how thick the border will be	<code>int</code>

## C.1.42 line-style

**Purpose:**

Line specific drawing properties.

**Equivalent in** JLineStyle

**JClass Chart:**

**Sub-Elements:** none

**Attributes:**

Name	Definition	Possible Values or Type
cap	Specifies the cap style used to end a line.	■ Butt ■ Round ■ Squared
color	Determines the color used to draw a line.	Color
join	Specifies the join style used to join two lines.	■ Miter ■ Bevel ■ Round
pattern	Dictates the pattern used to draw a line. Default is Solid.	■ None ■ Solid ■ Long_Dash ■ Short_Dash ■ LSL_Dash ■ Dash_Dot
width	Controls the line width.	int

**C.1.43    locale**

**Purpose:**  
Sets the locale for date, time, and number formatting. Does not affect the choice of the resource bundle used. For more information, see Section 12.6, [Internationalizing Your XML-based Chart](#).

**Equivalent in JClass Chart:**                `javax.util.Locale`

**Sub-Elements:**                none

**Attributes:**

Name	Definition	Possible Values or Type
country	Determines the country that is associated with the chart.	String
language	Determines the language code that is associated with the chart.	String
variant	Determines the variant code for the chart.	String

**C.1.44    marker**

**Purpose:**  
Marker properties.

**Equivalent in JClass Chart:**                `JCMarker`

**Sub-Elements:**                ■ `line-style`  
                                      ■ `chart-label`

**Attributes:**

Name	Definition	Possible Values or Type
associatedWithYAxis	Determines the axis from which the marker radiates. When <code>true</code> , the marker radiates from the y-axis.	boolean
drawnBeforeData	Determines whether the marker is drawn before the data is added or after.	boolean

Name	Definition	Possible Values or Type
endPoint	Specifies the value on the non-associated axis at which to end the marker line. The default is the maximum value on the axis.	double
includedInDataBounds	Determines whether or not the marker's value is included when calculating the data minimum and data maximum for the data view.	boolean
label	Specifies the name of the marker and the label used in the legend (if <code>VisibleInLegend</code> is true)	String
startPoint	Specifies the value on the non-associated axis at which to start the marker line. The default is the minimum value on the axis.	double
value	Specifies the value on the associated axis at which to draw the marker line.	double
visibleInLegend	Determines whether or not the marker label appears in the legend.	boolean

### C.1.45 **matte-border**

**Purpose:**  
Matte borders.

**Equivalent in JClass Chart:** `javax.swing.border.MatteBorder`

**Sub-Elements:** `insets`

**Attributes:**

Name	Definition	Possible Values or Type
color	The color of the border.	Color

**C.1.46    multi-col**

**Purpose:**  
Attributes to use when the value of the <legend> element’s type attribute is MultiCol.

**Equivalent in JClass Chart:**                    com.klg.jclass.util.legend.JCMultiColLegend

**Sub-Elements:**                    none

**Attributes:**

Name	Definition	Possible Values or Type
numColumns	The number of columns in the legend.	int
numRows	The number of rows in the legend.	int

**C.1.47    offset**

**Purpose:**  
The offset from where the label is attached to the chart to where the label is drawn.

**Equivalent in JClass Chart:**                    java.awt.Point

**Sub-Elements:**                    none

**Attributes:**

Name	Definition	Possible Values or Type
x	The X offset.	int
y	The Y offset.	int



**C.1.48    pie-format**

**Purpose:**  
Properties specific to pie charts.

**Equivalent in  
JClass Chart:**                      JCPieChartFormat

**Sub-Elements:**                    ■ fill-style  
   ■ explode-list

**Attributes:**

Name	Definition	Possible Values or Type
explodeOffset	Specifies the distance a slice is exploded from the center of a pie chart.	int
minSlices	Represents the minimum number of pie slices that the chart will try to display before grouping slices into the other slice.	int
otherLabel	Represents text string used on the “other” pie slice.	String
sortOrder	Determines the order in which pie slices will be displayed. Note that the other slice is always last in any ordering.	■ Data_Order ■ Ascending ■ Descending
startAngle	Determines the position in the pie chart where the first pie slice is drawn. A value of zero degrees represents a horizontal line from the center of the pie to the right-hand side of the pie chart; a value of 90 degrees represents a vertical line from the center of the pie to the top-most point of the pie chart; a value of 180 degrees represents a horizontal line from the center of the pie to the left-hand side of the pie chart; and so on. Slices are drawn clockwise from the specified angle. Values must lie in the range from zero degrees to 360 degrees.	double

Name	Definition	Possible Values or Type
thresholdMethod	Determines how thresholdValue is used. If Slice_Cutoff, thresholdValue is used as a cutoff to determine what items are lumped into the other slice. If the method is Percentile, items are grouped into the other slice until it represents a thresholdValue percent of the pie.	<ul style="list-style-type: none"> <li>■ Slice_Cutoff</li> <li>■ Percentile</li> </ul>
thresholdValue	This is a percentage value (between 0.0 and 100.0). How this value is used depends on the thresholdMethod (see above).	double

### C.1.49 plot-area

**Purpose:**

The rectangle within the chart area into which the data is drawn.

**Equivalent in JClass Chart:** `PlotArea`

**Sub-Elements:** `none`

**Attributes:**

Name	Definition	Possible Values or Type
background	Determines the background color used to draw inside the plot area.	Color
bottom	Determines the location of the bottom of the PlotArea.	int
foreground	Determines the color used to draw the axis bounding box.	Color
left	Determines the location of the left of the PlotArea.	int
right	Determines the location of the right of the PlotArea	int
top	Determines the location of the top of the PlotArea.	int

**C.1.50    point-label**

**Purpose:**  
Labels specified on a per point basis for the data. This specification overrides any point labels specified in the data.

**Equivalent in JClass Chart:**            java.util.String

**Sub-Elements:**                none

**Attributes:**                  #PCDATA (the label)

**C.1.51    polar-radar-format**

**Purpose:**  
Properties specific to radar and polar charts.

**Equivalent in JClass Chart:**            JCPolarRadarChartFormat

**Sub-Elements:**                none

**Attributes:**

Name	Definition	Possible Values or Type
halfRange	Determines whether the x-axis for polar charts consists of two half-ranges or one full range from 0 to 360 degrees.	boolean
originBase	Determines the angle of the theta axis origin in polar, radar, and area radar charts. Angles are based on zero degrees pointing east (the normal rectangular x-axis direction) with positive angles going counter-clockwise.	double
radarCircularGrid	Determines whether gridlines are circular or webbed for radar and area radar charts.	boolean
yAxisAngle	Determines the angle of the y-axis in polar, radar, and area radar charts. Angles are relative to the current origin base.	double

**C.1.52      series-point**

**Purpose:**  
Represents a series and point for use within other elements.

**Equivalent in JClass Chart:**                  none

**Sub-Elements:**                          none

**Attributes:**

Name	Definition	Possible Values or Type
series	The series index.	int, All, or Other_Slice
point	The point index.	int or All

**C.1.53      start-line-style**

**Purpose:**  
A line style for the starting value boundary line for thresholds.

**Equivalent in JClass Chart:**                  JCLineStyle

**Sub-Elements:**                          line-style

**Attributes:**                                  none

### C.1.54 symbol-style

**Purpose:**

Symbol specific drawing properties.

**Equivalent in  
JClass Chart:**

JCSymbolStyle

**Sub-Elements:**

none

**Attributes:**

Name	Definition	Possible Values or Type
color	Determines the color used to paint the symbol.	Color
shape	Determines the shape or symbol that will be drawn. Default is None.	<ul style="list-style-type: none"><li>■ None</li><li>■ Dot</li><li>■ Box</li><li>■ Triangle</li><li>■ Diamond</li><li>■ Star</li><li>■ Vert_Line</li><li>■ Horiz_Line</li><li>■ Cross</li><li>■ Circle</li><li>■ Square</li></ul>
size	Determines the size of the symbol.	int

**C.1.55     threshold**

**Purpose:**  
Threshold properties.

**Equivalent in  
JClass Chart:**                      JCThreshold

**Sub-Elements:**                    ■ fill-style  
   ■ start-line-style  
   ■ end-line-style

**Attributes:**

Name	Definition	Possible Values or Type
associatedWithYAxis	Determines the axis from which the threshold radiates. When <code>true</code> , the threshold radiates from the y-axis.	boolean
endValue	Sets the ending value of the threshold.	int
includedInDataBounds	Determines whether or not the threshold's <code>startValue</code> and <code>endValue</code> are included when calculating the data minimum and data maximum for the data view.	boolean
label	Specifies the name of the threshold and the label used in the legend (if <code>VisibleInLegend</code> is <code>true</code> ).	String
startValue	Sets the starting value of the threshold.	int
visibleInLegend	Determines whether or not the threshold label appears in the legend.	boolean

**C.1.56     titled-border**

**Purpose:**  
A titled border.

**Equivalent in  
JClass Chart:**                      `javax.swing.border.TitledBorder`

**Sub-Elements:**                    (empty-border|bevel-border|etched-border|  
   line-border|matte-border)

**Attributes:**

Name	Definition	Possible Values or Type
color	The border's color.	Color
font	The font to use for the title.	Font
title	The string for the title.	String
titleJustification	The title justification. Default is Default.	■ Default ■ Left ■ Center ■ Right ■ Leading ■ Trailing
titlePosition	The position or placement of the title. Default is Default.	■ Default ■ Above_Top ■ Top ■ Below_Top ■ Above_Bottom ■ Bottom ■ Below_Bottom

**C.1.57 value-label****Purpose:**

A value and label to be placed on an axis.

**Equivalent in** JCV alueLabel  
**JClass Chart:**

**Sub-Elements:** #PCDATA (the label)

**Attributes:**

Name	Definition	Possible Values or Type
value	Controls the position of a label in data space along a particular axis.	double

## C.2 JCChartData.dtd

Data can be read in by series or by point. When read in by series, a list of <data-point-label> elements is followed by a list of <data-series> elements. For more information, please refer to [Specifying Data by Series](#), in Chapter 4. When read in by point, a list of <data-series-label> elements is followed by a list of <data-point> elements. For more information, please refer to [Specifying Data by Point](#), in Chapter 4. The following is a list of each of the JCChartData.dtd elements, as well as their descriptions.

### C.2.1 chart-data

**Purpose:**  
The main element for chart data.

**Equivalent in JClass Chart:** JCChartData

**Sub-Elements:**

- data-point
- data-point-label
- data-series
- data-series-label

**Attributes:**

Name	Definition	Possible Values or Type
hole	The hole value for the data	double
name	The name of the data.	String

### C.2.2 data-point

**Purpose:**  
Specify all the series data for a given point.

**Equivalent in JClass Chart:** none

**Sub-Elements:**

- data-point-label
- x-data
- y-data

**Attributes:** none



### C.2.3 data-point-label

**Purpose:**

Provide a label for a given point.

**Equivalent in  
JClass Chart:** `java.util.String`

**Sub-Elements:** `#PCDATA` (the point label)

**Attributes:** `none`

### C.2.4 data-series

**Purpose:**

Specify all the point data for a given series.

**Equivalent in  
JClass Chart:** `none`

**Sub-Elements:**

- `data-series-label`
- `x-data`
- `y-data`

**Attributes:**

Name	Definition	Possible Values or Type
<code>seriesImageMapURL</code>	Specifies information pertaining to series image maps.	String
<code>seriesImageMapExtra</code>	Specifies supplemental information pertaining to series image maps.	String
<code>legendImageMapURL</code>	Specifies information pertaining to legend image maps.	String
<code>legendImageMapExtra</code>	Specifies supplemental information pertaining to legend image maps.	String

### C.2.5 data-series-label

**Purpose:**  
Provide a label for a given series. Normally seen in the Legend.

**Equivalent in  
JClass Chart:**            java.util.string

**Sub-Elements:**            #PCDATA (the series label)

**Attributes:**               none

**Note:** The <data-series-label> fulfills the same purpose as setting the <chart-data-view-series> tag's label attribute. If both are used, the <data-series-label> tag is ignored.

### C.2.6 x-data

**Purpose:**  
Provide a single x-data value.

**Equivalent in  
JClass Chart:**            double

**Sub-Elements:**            #PCDATA (the x-value)

**Attributes:**               none

**Attributes:**

Name	Definition	Possible Values or Type
clusterImageMapURL	Specifies information pertaining to cluster image maps.	String
clusterImageMapExtra	Specifies information pertaining to cluster image maps.	String

**C.2.7    y-data**

**Purpose:**  
Provide a single y-data value.

**Equivalent in  
JClass Chart:**                      double

**Sub-Elements:**                      #PCDATA (the y-value)

**Attributes:**

Name	Definition	Possible Values or Type
pointImageMapURL	Specifies image map information for each point in the series.	String
pointImageMapExtra	Specifies supplemental image map information for each point in the series.	String



# Appendix D

## Distributing Applets and Applications

The size of the archive and its related download time are important factors to consider when deploying your applet or application.

When you create an applet or an application using third-party classes such as JClass components, your deployment archive will contain many unused class files unless you customize your JAR. Optimally, the deployment JAR should contain only your classes and the third-party classes you actually use. For example, the *jcchart.jar*, which you used to develop your applet or application, contains classes and packages that are only useful during the development process and that are not referenced by your application. These classes include the Property Editors and BeanInfo classes. JClass JarMaster helps you create a deployment JAR that contains only the class files required to run your application.

### Using JClass JarMaster to Customize the Deployment Archive

JClass JarMaster is a robust utility that allows you to customize and reduce the size of the deployment archive quickly and easily. Using JClass JarMaster you can select the classes you know must belong in your JAR, and JarMaster will automatically search for all of the direct and indirect dependencies (supporting classes).

When you optimize the size of the deployment JAR with JClass JarMaster, you save yourself the time and trouble of building a JAR manually and determining the necessity of each class or package. Your deployment JAR will take less time to load and will use less space on your server as a direct result of excluding all of the classes that are never used by your applet or application.

For more information about using JarMaster to create and edit JARs, please consult its online documentation.

JClass JarMaster is installed automatically as part of the install process for JClass DesktopViews. For more details please refer to [Quest Software's Web site](#).



# Glossary

## A

### **Abstract Class**

Any class that cannot be instantiated because it contains at least one abstract method or is declared as abstract, is an Abstract Class. It is possible to extend an abstract class, or make it concrete, by implementing the abstract method.

See [Abstract Method](#).

### **Abstract Method**

An Abstract Method is simply a method which does not have a body, and therefore cannot have an implementation. Abstract Methods are only signature definitions.

### **Abstract Windowing Toolkit (AWT)**

The Abstract Windowing Toolkit is a series of graphical user interface components. These components are implemented using their native-platform versions, and provide a common subset of functionality.

### **Annotation**

An Annotation is a commentary that has been added to a chart. Annotations can be used to describe chart elements, making the chart more useful to an end-user.

### **Applet**

An Applet is a Java program invoked from an HTML page and run in a Java-enabled browser or applet viewer.

### **Area Chart**

An Area Chart is a type of chart that draws each data series as a connected points of data and fills in the area below the points. Each series is layered over the preceding series.

### **Area Radar Chart**

An Area Radar Chart is a type of chart that draws each data series as connected points of data and fills the area inside the points. The points are the same as they would be for a radar chart.

**Argument**

The data item specified in a method call is an Argument. Also referred to as a parameter.

See [Method](#).

**Array Data**

Array-formatted data shares a single series of x-data among one or more series of y-data. Array format is the recommended standard, because it works well with all of the chart types.

**Array**

An Array is a collection of objects of the same type. Each object has its own position, that is specified by an integer.

**Attribute**

An Attribute is directly associated with the entity of the instance or instances for which it exists, and is simply a named value or relationship to the entity.

**Axis Bounds**

Axis Bounds define which area of a generated chart will be displayed.

The area of a chart within the confines of the Axis Bounds will be displayed; the area of a chart outside the Axis Bounds will not.

**Axis Orientation**

Axis Orientation determines how the axes are positioned on the chart, with regard to their orientation. An axis can be either horizontal or vertical; generally, the x-axis horizontal is and the y-axis is vertical.

**B****Bar Chart**

A Bar Chart is a type of chart that generates a rectangle bounding a bar at a data point.

**Base Class**

A Base Class is a basic set of properties and methods from which one can extend in order to create more specialized classes.



**BeanBox**

Sun's BeanBox is a tool intended to be used as a text container and reference base.

**C****Candle Chart**

A Candle Chart is a type of chart that generates a rectangle bounding the Candle shape at a data point extending outward on all four sides.

**Chart Area**

The Chart Area contains most of the chart's actual properties because it is responsible for charting the data; it is also where the data is displayed.

**Chart Data Source**

The Chart Data Source takes real-world data and puts it into a form that JClass Chart can use. The data it produces will be used to generate a chart.

**Chart Label**

A Chart Label is a text String that is placed on the visible area of the chart. Its purpose is to clarify the displayed chart. A Chart Label can be either static or interactive.

**Chart Style**

A Chart Style defines the visual attributes of how data appears in the chart. This includes the lines and points in plots and financial charts, the color of each bar in bar charts, the slice colors in pie charts, and the color of each filled area in area charts.

**Chartable Data Source**

See [Chart Data Source](#).

**Class**

A Class is a collection of data and its methods.

Object implementations are defined in Classes, as well as the interface it implements and its superclass (which is Object by default). A Class also includes instance and class variables and methods.

Classes are arranged hierarchically so as to allow classes to inherit from their superclasses.

**Classpath**

A Classpath points to files, archives, and directories so that the JVM and other Java programs can find them. It is an environment variable that must be set for JClass products to function properly.

**Cluster**

Cluster, which is used in Bar Charts, refers to a series of bars which represent one unit of data.

For example, when displaying company earnings, you might chart three bars, where each bar represents the earnings of a different year. The three bars together form a cluster.

Clusters can be useful tools when comparing different sets of data.

**Cluster Overlap**

Cluster Overlap is the amount of space that the individual bars in a cluster overlap. Bars in different clusters do not overlap one another.

**Cluster Width**

The Cluster width is the total space used by the width of each cluster. It does not specify the width of the individual bars, but the width of the entire cluster.

**Component**

A Component is simply a software unit (for example, an applet). It exists at the application-level and is supported by a container. Components are configurable at deployment time.

**Constructor**

Constructors are instance methods that have the same name as their class, and are used to do any initialization required for new objects.

**Container**

A Container surrounds a component to provide it with security, deployment, runtime services, and component-specific services.

**CSS****Cascading Style Sheets**

CSS allow styles to be defined, and later used, in an HTML file. These styles can be stored in-line in the HTML file or in a separate CSS file.

## D

### **Data Hole**

A Data Hole is a specific location in a chart where no data is drawn. Data Holes are user defined, and the area in the chart will be left blank.

### **Data Loading**

Data Loading is the process by which JClass Chart receives the data from the database to use in the chart.

### **Data Model**

A Data Model is the result of physically organizing data in a logical fashion. It is used as a template or interface through which a data source is constructed. In the case of a database, a Data Model is useful because it contains information pertaining to the contents of a database, including how the database information is used and how items in the database relate to one another.

### **Data Series**

A Data Series is a series of columns that contain numeric data.

### **Data Source**

Data Source refers to the database from which JClass Chart has gathered the data used to draw the chart.

### **Data View**

A Data View is a collection of series objects, one for each series of data points, used to store the visual display style of each series.

### **DBMS**

#### **DataBase Management System**

DBMS is the software (or set of software) that controls the major functions of a database (for example, organization, storage, retrieval of data, and security).

### **Delimiter**

A Delimiter series as an indication of the beginning and the end of a block or data.

### **Depth**

In a 3D chart, Depth defines the dimension downward, backward, or inward of the chart.

**Document Type Definition (DTD)**

In JClass Chart, the DTD is a file, included with the installation, that defines the tags and attributes used to specify the appearance of a chart when using XML.

DTD tags are built into the JClass Chart API.

**Dwell Label**

A Dwell Label is an interactive label, available with Flash encoding, that appears when a cursor remains over a chart and disappears when the cursor moves.

Normally, the cursor would remain over a point, bar, or slice, and the label would display data referring to that point, bar, or slice.

**E****Elevation**

In a 3D chart, Elevation refers to the height to which the 3D elements of the chart are lifted above a point of reference.

**Event**

An Event is a significant occurrence in a program. For example, chart actions are Events that can take place in JClass Chart.

**Event Trigger**

An Event Trigger is a mapping of a mouse operation and/or a key press to a chart action.

**Exception**

When running a program, an Exception is something that will stop the program's normal execution. Generally, an error will be produced.

**Explode**

Explode is an action that can take place in a pie chart, if exploding pie slices are enabled.

Explode refers to a pie slice that detaches from the rest of the pie when a user clicks it.

**Extensible Markup Language (XML)**

Extensible Markup Language is a standard information document exchange format. XML is a simplified version of SGML, which creates very structured

documents that is intended for use with web documents. XML allows for the creation of customized tags; its structure is defined in a DTD file.

See [Document Type Definition \(DTD\)](#).

### **Extensible Style Sheet Language (XSL)**

Extensible Style Sheet Language is a W3C standard for defining stylesheets for XML. XSL uses the XML language.

### **Extensible Style Sheet Language Transformation (XSLT)**

Extensible Style Sheet Language Transformation is the language used to alter and manipulate documents, transforming XML documents.

## **F**

### **Footer**

The Footer is the text at the bottom of the chart.

### **Foreground**

The Foreground of a chart is the part that appears the closest to the user. If there are layers to the chart, the foreground is the top layer.

## **G**

### **Gap**

Gap refers to the space between different axis annotations in a chart.

### **General Data**

General-formatted data specifies a series of x-data for every series of y-data.

General Format may not display data properly in Stacking Bar, Stacking Area, Pie, and Bar charts.

### **GIF**

#### **Graphic Interchange Format**

GIF is a loss-less compression image format. GIF files use 256 colors, and are therefore most commonly used for drawings and icons.

### **Gridlines**

Gridlines are patterns of regularly spaced horizontal, vertical, and/or circular lines that identify different X and Y points on the chart. Gridlines can be set to visible or invisible.

## **H**

### **Half-Range**

When using a circular grid, a Half-Range is an axis that is displayed in two ranges: one from -180 to 0 degrees, and a second from 0 to 180 degrees.

### **Header**

The Header is the text at the top of the chart.

### **Hex Values (color)**

Hex Values are a way to define colors. Hex values are presented in the format #RRGGBB, #RRRRGGGGBBBB, or #N.

### **Hi-Lo Chart**

The Hi-Lo Chart is a type of chart that generates a chart with a line at a data point extending outwards on all four sides

### **Hi-Lo-Open-Close Chart**

The Hi-Lo-Open-Close Chart is a type of chart that generates a chart with a rectangle bounding the Hi-Lo-Open-Close shape at a data point extending outwards on all four sides.

### **Hole Value**

See [Data Hole](#).

## **I**

### **IDE**

#### **Integrated Development Environment**

An IDE is a graphical software development interface. JClass components can be used inside many IDEs; see the [JClass DesktopViews Installation Guide](#) for details.

**Inheritance**

The variables and methods defined in a class are passed down, or inherited, by their subclasses. This concept is known as Inheritance.

**Inheritance Hierarchy**

An Inheritance Hierarchy is the diagram that represents, in a hierarchical fashion, classes and subclasses, starting with a root class.

**Instantiate**

Instantiating refers to the production of a particular object from its class template or definition.

**Intelligent Defaults**

Intelligent Defaults refers to the MultiChart Bean's set of dynamic default settings. In the custom property editors, the default settings will automatically adjust.

**Interface**

An Interface is what is seen and used or manipulated on the computer screen by a user (e.g. IBM's Visual Age).

**Internationalization**

Internationalization is the process of making software that is ready for adaptation to various languages and regions without engineering changes. JClass products are internationalized if you purchase the source code.

**Interpreter**

The Interpreter is used to invoke a Java program. It will decode and execute each statement, and runs until the end of the program.

**J****JAR****Java Archive**

A JAR is a compressed file which includes all classes necessary for a Java applet to work.

**Java Application**

A Java Application is a standalone Java program run by the Java virtual machine.

**JavaBeans**

Java Beans are logical, portable, platform-independent, and reusable components that are written and used by Java.

See [Component](#).

**JDK****Java Development Kit**

A JDK is a development environment, created by Sun, which allows a user to create Java code.

**JPEG****Joint Photographic Expert Group**

JPEG is an image format with a standardized image compression mechanism (note that it has lossy compression). It is most commonly used for pictures and other complex images.

**JVM****Java Virtual Machine**

A JVM is a virtual machine that is used to safely execute byte code in Java class files on a microprocessor.

**L****Legend**

The Legend is a reference in the JClass Chart chart which explains the significance of any differences that exist on the plotted chart (for example, bar color or different lines styles).

**Linear Axis**

A Linear Axis is an axis where every tick representing a number has the same distance between it and the number preceding or following it in the axis. In other words, the distance between ticks is a constant.

**Logarithmic Axis**

A Logarithmic Axis is an axis that uses the values of  $N/N_0$ , rather than linear values, to aid in the graphing of logarithmic expressions. In other words, the distance between ticks is based on a logarithmic scale.



## M

### Method

A Method is a function which is defined in a class and, unless otherwise specified, is not static. When a class is instantiated, its Methods will run.

### Model View Controller (MVC)

MVC is a logical way of separating the different elements of interactive software. The internal workings are referred to as the Model, the visual representation portrayed to the viewer is the View, and the ways in which the user changes the View or provides input is the Controller.

### Mouse Event

A Mouse Event is an event that has been implemented through the use of a mouse. See [Event](#).

## O

### Object

An Object is used to contain the variable data and method definitions that are needed to instantiate a class, and is the building block or object-oriented programming.

### Object Collection

An Object Collection is a container that is used to hold and group objects together. Once the objects have been placed in an Object Container, they can be accessed through one variable.

### Origin Base

The Origin Base is the start angle of a circular chart.

### “Other” Slice

In a pie chart, the “Other Slice” refers to the grouping of unimportant values into one slice, rather than have them represented separately. The use of the “Other Slice” often makes a pie chart more effective.

## P

### **Package**

A Package is a group of classes or interfaces, and is declared with the package keyword. Often, packages are groups of classes or interfaces that have been grouped together to provide a specific type of functionality.

### **Parameter**

A Parameter is a data item specified in a method call. Also referred to as an argument.

### **Parser**

A Parser determines the syntactic structure of a sentence or String.

### **PDF**

#### **Portable Document Format**

PDF allows a document to be represented in a form that is independent from the original software, hardware, and operating system that was used to create it. Adobe System's Acrobat uses this file format, and allows PDF files to contain text, graphics, and images. PDF files are known to be versatile and well suited for distribution.

### **Pick**

Pick is a mechanism in which allows users to determine the position, in pixels, of data points displayed on a chart.

### **Pie Chart**

A Pie Chart is a type of chart that generates a circular graph that is divided into sections, where each section represents a portion of the entire graph. Pie charts are most often used to represent percentage data which, when added, is equal to 100%.

### **Plot Area**

The Plot Area is the specific area of the chart where the chart data is plotted.

### **Plot Chart**

A Plot Chart is a type of chart that generates a chart with lines and symbols.

### **PNG**

PNG is an image format that uses loss-less compression. PNG uses 24 bits of color (including transparent colors).

**Point Data**

Point Data is a data type that is comprised of one or more series of points.

**Point Label**

A Point Label is an annotation method used to label specific points of data on the x-axis. It is used with array data only.

**Polar Chart**

A Polar Chart is a type of chart that generates a chart with a series of connected points of data on a polar coordinate system.

**Procedure**

A Procedure is the instructions used to perform a specific task.

**Property**

Properties are the named method attributes of a class that can affect its appearance or behavior.

**Property Parameters**

Property Parameters are programming options referring to parameters.

See [Parameter](#).

**Property Sheet**

A Property Sheet allows for changes to be made to the properties of the component under consideration.

## R

**R Value**

The R value is the distance of a point from the y-axis origin in a polar chart.

**Radar Chart**

A Radar Chart is a type of chart that draws a chart with each data series as connected points along radar “sticks” spaced equally apart.

**Rendered Chart**

The Rendered Chart is the ensemble of all components of a chart, generated by JClass Chart.

**Rendering**

Rendering is the act of converting high-level object-based description into graphical elements for display (in this case, a chart).

**RGB Value (color)**

RGB Values are a way to define colors using red, green, and blue values. RGB values are presented in the format #RRGGBB.

**RMI****Remote Method Invocation**

RMI allows one computer running a Java program to access methods and objects that are running in a Java program on another machine.

**Rotation**

Rotation refers to a transformation that you can perform on a 3D chart. Specifically, it refers to the movement of the chart's coordinate system, providing the chart with new axes with a specified angular displacement from its original position. The origin will remain fixed despite the rotation.

**S****Scatter Plot Chart**

A Scatter Plot Chart is a type of chart that plots series of points.

**Series Label**

A Series Label is an annotation method used to label a series of points of data in the legend.

**Stacking Area Chart**

A Stacking Area Chart is a type of chart that draws each data series as connected points of data and fills in the area below the points. Each y-series is placed on top of the last one to show the area relationships between each series and the total.

**Stacking Bar Chart**

A Stacking Bar Chart generates a rectangle bounding the segment of a stacking bar that represents a data point.

**Start Angle**

In a Pie Chart, the start angle is the position in the pie chart where the first pie slice is drawn.

**Stock Data Source**

A Stock Data Source is a data source that is included with your JClass Chart installation.

**String**

A String is a sequence of alphanumeric, punctuation, and white spaces.

**Subclass**

A Subclass is a class that is derived from another class. There might be one or more classes between them. A Subclass inherits the variables and methods contained in the superclass.

See [Class](#).

**Subcomponent**

A Subcomponent is a component that is derived from another component.

See [Component](#).

**Swing**

Swing is a set of classes which extend the Java AWT package and is used for creating a graphical user interface.

## T

**Theta**

Theta is the amount of rotation from the x-axis origin in a polar chart. The angle from the x axis origin is measured counterclockwise.

**Tick**

A Tick object is a collection of uniformly spaced marks and labels, used to show the scale values.

**Time Label**

A Time Label is an annotation method used to label specific points of data on the X axis according to time.

**Top-Level Object**

A Top-Level-Object is a Java Object class on which other classes base themselves.

**Transposed Data**

Transposed Data refers to data where the meaning of the data series and points is switched.

**V****Value Labels**

A Value Label is an annotation method used to appear along an axis at user-specified values.

**Z****Zooming**

Zooming refers to a transformation that you can perform on a chart. Specifically, it refers to selecting an area of a chart to expand. In other words, Zooming is seeing a close-up of one area of a chart.

# Index

- 100 percent axis 51
- 3D effect 164
- 3D effects 239, 273
  - Depth 240
  - Elevation 239
  - Rotation 239
  - View3DEditor 239

## A

- abstract class, definition 383
- abstract method, definition 383
- abstract windowing toolkit (AWT), definition 383
- action
  - axes
    - specifying 192
    - calling directly 191
    - programming 190
    - removing mappings 191
- advanced chart programming 187
- anchor
  - property 154
- anno tag 333
- annotation
  - definition 383
  - methods 121, 122
  - values 122
- API 16
- applet
  - loading data 69
- applet, definition 383
- area chart 23
- area charts
  - definition 383
  - special properties 48
- area radar chart 25
  - axis 28
  - FastUpdate 190
- area radar charts 64
  - annotation method 122
  - AreaRadarChartDraw class 65
  - background 64
  - data array 74
  - data format 65
  - definition 383
  - mapping 192
  - min value 130
  - picking 193
  - point labels 124
- area-format tag 334
- AreaRadarChartDraw class 65
- argument, definition 384
- array data
  - definition 384
  - format 73
  - layout 26
- array, definition 384
- ATTACH\_COORD 152
- ATTACH\_DATACoord 152
- ATTACH\_DATAINDEX 152
- AttachMethod property 152
- attribute, definition 384
- AutoLabel property
  - adding labels 152
  - generating labels for Flash output 153
- automatic labelling 268
- axis
  - adding second 137
  - annotation
    - overview 121
    - PointLabels 124
    - selecting a method 122
    - TimeLabels 125
    - ValueLabels 124
    - Values 122
  - area radar chart 28
  - bounding box 272
  - bounds 132
  - direction 130
  - gridlines 135
  - labelling 121
    - value annotation 122
  - labels, custom 128
  - logarithmic 133
  - min and max 132
  - multiplier property and second y-axis 137
  - orientation
    - definition 384
    - inverting 131
  - origins 132
  - polar chart 28
  - positioning 129
  - rotating annotation 133

- rotating title 133
- second y-axis, adding 137
- title 133
- axis bound, definition 384
- axis tag 335
- axis-formula tag 339
- AxisGrid 256
- AxisMisc 259
- AxisOrigin 257
- AxisPlacement 258
- AxisPointLabels 259
- AxisScale 261
- AxisTimeLabels 262
- AxisTitle 263
- axis-title tag 339

## B

- background color 162
- bar
  - cluster overlap 48, 49
  - cluster width 49
- bar chart 22
- bar charts
  - 3D effect 164
  - definition 384
  - image filled 201
  - labelling with PointLabels 124
  - origin placement 132
  - special properties 49
- Bar3d and 3d Effect 44
- bar-format tag 340
- base class, definition 384
- basic chart information 21
- batching chart updates 189
- Bean 245
  - data loading methods 241
  - MultiChart 233, 253
  - overview 245
  - properties 245
    - setting at design-time 245
  - reference 233
  - SimpleChart 233
  - tutorial 245
- BeanBox 246
  - definition 385
- bevel-border tag 341
- borders
  - using 161
- bounds, data
  - including marker values 177
  - including threshold values 184

## C

- calling
  - an action 191
  - methods 29
- candle chart 24
- candle charts
  - ChartStyle properties used 52
  - definition 385
  - simple and complex display 52
- candle-format tag 341
- chart
  - areas 21
  - basics 21
  - elements
    - location 163
    - positioning 163
    - size 163
  - labels *See* chart label.
  - orientation 130
  - outputting 187
  - setting type 22
  - special properties 47
  - styles 155
    - area charts 155
    - bar charts 155
    - customizing 156
    - customizing all 157
    - fill style 156
    - line style 156
    - pie charts 155
    - plot and financial charts 155
    - symbol style 157
  - terminology 21
  - type 47
  - user interaction 190
- chart area
  - definition 385
- chart customizer
  - editing properties 32
  - viewing properties 32
- chart data source, definition 385
- chart label
  - definition 385
  - internationalizing, in XML-based charts 229
  - marker, setting 175
  - overview 151
  - See also* labels.
- chart style, definition 385
- chart tag 332
- chart type
  - area 23
  - area radar 25
  - bar 22
  - candle 24
  - hi-lo 24



- hi-lo-open-close 24
- PIE 238
- pie 23
- plot 22
- polar 24
- radar 25
- scatter plot 22
- stacking area 23
- stacking bar 23
- chart.dtd 332
  - anno 333
  - area-format 334
  - axis 335
  - axis-formula 339
  - axis-title 339
  - bar-format 340
  - bevel-border 341
  - candle-format 341
  - chart 332
  - chart-area 342
  - chart-data-file 343
  - chart-data-view 344
  - chart-data-view-series 346
  - chart-interior-region 347
  - chart-label 348
  - chart-style 349
  - component 350
  - compound-border 351
  - coord 351
  - data-coord 352
  - data-index 352
  - empty-border 353
  - end-line-style 353
  - etched-border 353
  - event-trigger 354
  - explode-list 354
  - external-java-code 355
  - fill-style 355
  - footer 356
  - grid 357
  - header 358
  - hi-lo-open-close-format 358
  - hole-style 359
  - image-file 359
  - insets 360
  - key 361
  - label 361
  - layout-hints 362
  - legend 362
  - legend-column 363
  - line-border 364
  - line-style 365
  - locale 366
  - marker 366
  - matte-border 367
  - multi-col 368

- offset 368
- pie-format 369
- plot-area 370
- point-label 371
- polar-radar-format 371
- series-point 372
- start-line-style 372
- symbol-style 373
- threshold 374
- titled-border 374
- value-label 375
- chartable data source, definition 385
- ChartArea
  - positioning 163
- chart-area tag 342
- chart-data tag 376
- ChartDataEvent 82
- chart-data-file tag 343
- ChartDataListener 82
- ChartDataManageable 83
- ChartDataManager 83
- ChartDataModel 31, 78
- ChartDataSupport 83
- ChartDataView 152
  - ChartType property 22
  - containment hierarchy 31
  - converting coordinates 199
  - HTML property syntax 316
  - Inverted property 48, 50
  - IsInverted property 48, 50, 131
  - PointLabels 124
  - programming ChartStyles 155
  - property summary 277
- chart-data-view tag 344
- ChartDataViewSeries 31
  - HTML property syntax 317
  - property summary 280
- chart-data-view-series tag 346
- chart-interior-region tag 347
- chart-label tag 348
- ChartLabels property 151
- chart-style tag 349
- ChartStyles
  - customizing 155
  - use in financial chart types 52
- ChartText
  - property summary 281
- ChartType property 22
  - ChartDataView 22
- choosing chart type 22
- class, definition 385
- classpath, definition 386
- cluster
  - definition 386
  - overlap
    - bar charts 49

- definition 386
- width
  - bar chart 49
  - definition 386
- collections of objects 28
- colors
  - background 162
  - default 162
  - foreground 162
  - setting 161
  - transparency 163
- component tag 350
- component, definition 386
- compound-border tag 351
- constructor, definition 386
- container 246
  - definition 386
- coord tag 351
- coordinates, converting 199
  - CoordToDataCoord 200
  - CoordToDataIndex 200
  - CoordToDataSeries 200
  - DataCoordToCoord 200
  - DataIndexToCoord 200
  - map 192, 199
  - unmap 192, 199
- CoordToDataCoord 200
- CoordToDataIndex 200
- CoordToDataSeries 200
- crosshair, creating 171
- CSS, definition 386
- CUSTOM\_FILL 299
- CUSTOM\_PAINT 299
- CUSTOM\_STACK 299
- customizer
  - editing properties 32
  - viewing properties 32
- CustomPaint 299

## D

- data
  - array
    - area radar chart 74
    - layout 26
    - polar chart 74
    - radar chart 74
  - formats 68, 73
    - area radar charts 65
    - array 73
    - array layout 76
    - comments 76
    - examples 75
    - explanation 75
    - general 73
    - general layout 77
    - point labels 76
    - polar charts 60
    - polar charts, array 60
    - polar charts, general 60
    - pre-formatted 73
    - radar charts 63
    - series labels 77
    - transposed data 77
  - layout 26
  - loading from XML source 70
    - interpreter 71
    - labels 73
    - specifying by point 72
    - specifying data by series 71
  - min and max 132
- data bounds 132
  - including marker values 177
  - including threshold values 184
- data hole
  - definition 387
  - logarithmic axes 133
- data loading, definition 387
- data model
  - definition 387
  - targeted *See* targeted data model.
  - underlying 67
- data series, definition 387
- data set
  - custom, creating 112
- data source
  - BaseDataSource 68
- data formats 73
  - array layout 76
  - comments 76
  - examples 75
  - explanation 75
  - general layout 77
  - point labels 76
  - series labels 77
  - transposed data 77
- data views 67
- definition 387
- formatted file 68
- hierarchy 84
- JCChartSwingDataSource 68
- JCDefaultDataSource 68
- JCFileDataSource 68
- JCInputStreamDataSource 68
- JCStringDataSource 68
- JCURLDataSource 68
- JDBCDataSource 68
- loading data from a file 68
- loading from a Swing TableModel 70
- loading from a URL 68
- loading from an XML source 70

- interpreter 71
  - labels 73
  - specifying by point 72
  - specifying data by series 71
- making your own 77
  - hole values 81
  - labelling 79
  - simplest 77
- pre-built 68
- support classes 82
- updating 82
- data values, highlighting 167
- data view 67, 267
  - definition 387
- database management system, definition 387
- DataChart 267
- data-coord tag 352
- DataCoordToCoord 200
- data-index tag 352
- DataIndexToCoord 200
- data-point tag 376
- data-point-label tag 377
- data-series tag 377
- data-series-label tag 378
- DataSource 31, 268
- DataView, multiple axes 137
- Date methods 127
- dateToValue method 127
- DBMS, definition 387
- DefaultDataModel 89
- delimiter, definition 387
- depth
  - definition 387
- document type definition, definition 388
- draw on front plane 268
- drawLegendItem() 146
- drawLegendItemSymbol() 146
- DTD
  - chart.dtd 332
    - anno 333
    - area-format 334
    - axis 335
    - axis-formula 339
    - axis-title 339
    - bar-format 340
    - bevel-border 341
    - candle-format 341
    - chart 332
    - chart-area 342
    - chart-data-file 343
    - chart-data-view 344
    - chart-data-view-series 346
    - chart-interior-region 347
    - chart-label 348
    - chart-style 349
    - component 350
    - compound-border 351
    - coord 351
    - data-coord 352
    - data-index 352
    - empty-border 353
    - end-line-style 353
    - etched-border 353
    - event-trigger 354
    - explode-list 354
    - external-java-code 355
    - fill-style 355
    - footer 356
    - grid 357
    - header 358
    - hi-lo-open-close-format 358
    - hole-style 359
    - image-file 359
    - insets 360
    - key 361
    - label 361
    - layout-hints 362
    - legend 362
    - legend-column 363
    - line-border 364
    - line-style 365
    - locale 366
    - marker 366
    - matte-border 367
    - multi-col 368
    - offset 368
    - pie-format 369
    - plot-area 370
    - point-label 371
    - polar-radar-format 371
    - series-point 372
    - start-line-style 372
    - symbol-style 373
    - threshold 374
    - titled-border 374
    - value-label 375
  - chart-data.dtd
    - chart-data 376
    - data-point 376
    - data-point-label 377
    - data-series 377
    - data-series-label 378
    - x-data 378
    - y-data 379
  - JCChartData.dtd 376
  - primer 220
- DTD, definition 388
- dwell labels 268
  - automatically generated 153
  - definition 388
  - individual 153

## E

- EditableChartDataModel 80
- elements
  - location 163
  - positioning 163
  - size 163
- elevation
  - definition 388
- empty-border tag 353
- encode
  - chart as image 187
  - method 187
- Encode example 188
- Encode method 187
- Encode method code example 188
- end-line-style tag 353
- end-user interaction 44
- EPS file format 187, 188
- error bar charts 52
- etched-border tag 353
- event trigger 190
  - programming 191
- event trigger, definition 388
- event, definition 388
- event-trigger tag 354
- exception, definition 388
- explode, definition 388
- ExplodeList property 56
- explode-list tag 354
- ExplodeOffset property 56
- exploding pie slices 56
- extensible markup language, definition 388
- extensible style sheet language transformation, definition 389
- extensible style sheet language, definition 389
- external-java-code tag 355

## F

- FastAction 189
- FastUpdate 189
- FileDataSource 68
  - tutorial 38
- fill style, setting threshold 182
- fill-style tag 355
- financial charts
  - ChartStyle properties used 52
  - ChartStyles 52
- Flash
  - interactive labels 153
- fonts
  - changing 161
  - choosing 161
- footer 39

- footer tag 356
- footers
  - definition 389
  - HTML property syntax 318
  - positioning 163
  - titles 139
- foreground
  - color 162
  - definition 389
- formatted file 68
- full-range x-axis, polar charts 61

## G

- gap 256
- gap, definition 389
- general data
  - definition 389
  - format 73
  - layout 26
- get object properties 27
  - Java code 27
- getOutlineColor 146
- GIF 163
- GIF file format 187, 188
- GIF, definition 389
- grid tag 357
- gridlines 135, 237, 256
  - definition 390
  - grid appearance 136
  - grid spacing 136

## H

- half-range
  - definition 390
  - x-axis, polar charts 61
- HalfRange property 61
- header 39
- header tag 358
- headers
  - definition 390
  - HTML property syntax 318
  - positioning 163
  - titles 139
- HeaderText 265
- highlighting data values 167
- hi-lo chart 24
- hi-lo charts
  - ChartStyle properties used 52
  - definition 390
- hi-lo-open-close chart 24
- hi-lo-open-close charts 52
  - definition 390

- hi-lo-open-close tag 358
- hole value
  - definition 390
  - specifying 81
- hole values 81
- hole-style tag 359
- HoleValueChartDataModel 81
- HTML property syntax
  - ChartDataView 316
  - ChartDataViewSeries 317
  - footer 318
  - header 318
  - JCAreaChartFormat 318
  - JCAxis 319
  - JCBarChartFormat 321
  - JCCandleChartFormat 322
  - JCChart 322
  - JCChartArea 323
  - JCChartLabel 324
  - JCDataIndex 325
  - JCHiLoChartFormat 326
  - JCHLOCCChartFormat 326
  - JCLegend 326
  - JCPieChartFormat 329
  - JCPolarRadarChartFormat 329

## I

- IDE
  - definition 390
  - setting properties 28
- Image 201
- image formats 187
- image-file tag 359
- images
  - bar charts, image filled 201
  - transparent 163
- inheritance hierarchy 29
- inheritance hierarchy, definition 391
- inheritance, definition 391
- insets tag 360
- instantiate, definition 391
- intelligent defaults 254
- intelligent defaults, definition 391
- interacting with the chart 190
- interactive labels 153
- interactively
  - setting properties 27
- interface, definition 391
- internationalization 33
  - definition 391
  - XML-based charts 229
- interpreter, definition 391
- introduction to JClass Chart 13
- Inverted property

- properties
  - Inverted 48, 50
- inverting
  - a chart 131
- inverting X- and Y-axis 43
- IsComplex property 52
- IsConnected 155
- IsInverted property 48, 50, 131
- IsOpenCloseFullWidth
  - using for error bar charts 52
- IsOpenCloseFullWidth property 52
- IsShowingClose property 52
- IsShowingOpen property 52

## J

- JAR
  - definition 391
- Java application, definition 391
- JavaBeans 245
  - overview 245
- JavaBeans, definition 392
- JCAнно
  - property summary 282
- JCAppletDataSource
  - pre-built DataSource 68
- JCAreaChartFormat
  - HTML property syntax 318
  - property summary 284
- JCAxis
  - AnnotationRotation property 134
  - containment hierarchy 31
  - HTML property syntax 319
  - IsLogarithmic property 133
  - IsReversed property 131
  - Min and Max properties 132
  - property summary 284
  - second y-axis 137
  - JCAxis.POINT\_LABELS 122
  - JCAxis.TIME\_LABELS 122
  - JCAxis.VALUE 122
  - JCAxis.VALUE\_LABELS 122
  - JCAxisFormula
    - property summary 290
  - JCAxisTitle 134
    - property summary 290
    - Rotation property 134
    - Text property 134
- JCBarChartFormat
  - HTML property syntax 321
  - property summary 292
- JCCandleChartFormat 52
  - HTML property syntax 322
  - property summary 292
- JCChart

- HTML property syntax 322
- object hierarchy 30
- property summary 293
- JCChartApplet 212
- JCChartArea 30
  - 3D effect properties 164
  - HTML property syntax 233
  - property summary 295
- JCChartData.dtd 376
  - chart-data 376
  - data-point 376
  - data-point-label 377
  - data-series 377
  - data-series-label 378
  - x-data 378
  - y-data 379
- JCChartLabel 31, 151
  - HTML property syntax 324
  - property summary 296
- JCChartLabelManager 31
  - property summary 297
- JCChartLegendManager 147, 148
- JCChartStyle 31, 155
  - property summary 297
- JCDataIndex 200
  - HTML property syntax 325
  - returned by pick() method 199
- JCEditableDataSource
  - pre-built DataSource 68
- JCEncodeComponent class 187
- JCFillStyle 156
  - property summary 299
- JCGrid
  - property summary 299
- JCGridLegend 140
  - property summary 300
- JCHiLoChartFormat 52
  - HTML property syntax 326
- JCHiloChartFormat
  - property summary 301
- JCHLOCCChartFormat 52
  - HTML property syntax 326
  - property summary 301
- JCLabelGenerator interface 128
- JClass Chart
  - feature overview 13
  - overview 13
- JClass Chart Beans 233, 234
- JClass JarMaster 381
- JCLegend 30, 143, 144
  - HTML property syntax 326
  - property summary 302
  - Toolkit 143
- JCLegendItem 144, 145, 148
- JCLegendPopulator 143, 144, 145, 147
- JCLegendRenderer 143, 144, 146, 147

- JCLineStyle 156
  - property summary 303
- JCMarker
  - property summary 304
  - using 167
- JCMultiColLegend
  - property summary 305
- JCMultiColumnLegend 140
- JComponent 30
- JCPieChartFormat 54, 56
  - HTML property syntax 329
  - property summary 306
- JCPolarRadarChartFormat
  - HTML property syntax 329
  - property summary 307
- JCSymbolStyle 157
  - property summary 308
- JCThreshold
  - property summary 309
  - using 177
- JCURLDataSource
  - parameter options 69
- JCValueLabel
  - property summary 310
- JDBC
  - adding result set data to a chart 87
  - result set data set implementations 90
  - result set, review 86
- JDK
  - definition 392
- JPEG
  - definition 392
- JPEG file format 187, 188
- JVM, definition 392

## K

- key tag 361

## L

- label 39
  - demos 151
- label tag 361
- LabelledChartDataModel 79
- labelling points 41
- labels
  - adding
    - connecting lines 154
    - text 154
    - to a chart 152
  - attaching
    - to a data item 152
    - to chart area coordinates 152

- to plot area coordinates 152
- attachment method 152
- automatically generated dwell labels 153
- axis 121
  - custom axis 128
- Flash support 153
- formatting text 154
- implementations 151
- individual dwell labels 153
- interactive 151, 153
- positioning labels 154
- static 151
  - XML data loading 73
- labels, chart 151
- layout-hints tag 362
- learning JClass Chart 45
- legend
  - definition 392
- legend tag 362
- legend-column tag 363
- LegendLayout 266
- legends
  - custom legends 140
    - layout 144
    - population 145
    - rendering 146
  - customizing 140
  - JCLegend Toolkit 143
  - marker labels, adding 175
  - multiple-column 140
  - orientation 141
  - positioning 163
  - single-column 140
  - text 141
  - threshold labels, adding 184
  - using 139
- license 16
- licensing 16
- line style
  - setting marker 173
- linear axis, definition 392
- line-border tag 364
- lines, adding marker 167
- lines, adding threshold boundary 183
- line-style tag 365
- listener
  - adding JClass Chart 84
- loading data 26
  - applet 69
- loading data from a file 38
- LoadProperties
  - constructing 207
- locale tag 366
- logarithmic
  - axes 133
    - specifying 133

- x-axis 133
- logarithmic axis, definition 392
- logical series
  - style 52

## M

- map 199
  - coordinates 192, 199
- marker tag 366
- markers
  - chart labels, attaching 175
  - crosshair, creating 171
  - drawing before or after data 174
  - including values in data bounds calculations 177
  - labels in legend 175
  - line style, setting 173
  - overview 167
  - start and end points, setting 172
  - troubleshooting missing 177
  - x-axis, creating 168
  - y-axis, creating 169
- matte-border tag 367
- method, definition 393
- methods
  - calling 29
- model view controller, definition 393
- modifying data 80
- mouse event, definition 393
- MultiChart 253
  - 3D Depth 273
  - 3D effects 273
  - 3D Elevation 273
  - 3D planes 268
  - 3D Rotation 273
  - actions
    - customize 274
    - depth 274
    - pick 274
    - rotation 274
    - translate 274
    - zoom 274
  - adding footer text 264
  - adding header text 265
  - appearance controls 270
  - automatic dwell labels 268
  - axis
    - annotation 255
    - bounding 272
    - controls 255
    - number precision 261
    - numbering 261
    - origin 257
    - placement 258
    - precision 260

- range 260, 261
- tick marks 260
- title 263
- AxisMisc 259
- AxisPointLabels 259
- AxisRelationships 260
- AxisScale 261
- background 270
- bounding box 272
- chart areas 270
- chart types 267
- ChartAppearance 271
- ChartAreaAppearance 271
- controlling 3D planes 268
- data view 268, 269
- DataMisc 268
- draw on front plane 268
- Editable 259
- events 273
- Font 272
- foreground 270
- gap 256
- get started 254
- gridlines 256
- hiding an axis 259
- intelligent defaults 254
- label rotation 256
- legend layout 266
- loading data from a file 269
- multiple axes 253
- multiple data views 254
- point labels 259
- property reference 255
- selecting axes for a data view 267
- setting multiple axes properties 254
- tick spacing 261
- time base 262
- time format 262
- time labels 262
- time unit 262
- TriggerList 273
- value labels 263
- MultiChart Bean 233
- MultiChart showing a data view 268
- multi-col tag 368
- multiple axes
  - MultiChart 253
  - setting properties 254
- multiple data views
  - MultiChart 254
- multiple x-axes 74
- MVC, definition 393

## O

- object collection, definition 393
- object collections 28
- object containment hierarchy 30
- object properties
  - get 27
  - get with Java code 27
  - set 27
  - set with Java code 27
- object, definition 393
- offset tag 368
- origin
  - coordinates 132
  - customizing 132
  - placement 132
    - bar chart 132
  - setting in polar charts 59
- origin base, definition 393
- Origin property 132
- OriginBase property 59
- OriginPlacement property 132
- other slice 54
  - definition 393
  - labels 55
  - style 55
- OutputProperties
  - setting on a background image 210
- outputting charts 187
- outputting charts to images 187

## P

- package, definition 394
- parameter, definition 394
- parameters
  - reference 315
- parser, definition 394
- PCL file format 188
- PDF file format 187, 188
- PDF, definition 394
- pick 192, 199
  - focus 193
- pick, definition 394
- pie chart 23
  - definition 394
- pie charts
  - 3D effect 164
  - exploding pie slices 56
  - labelling with PointLabels 124
  - logarithmic axes 133
  - other slice 54
    - labels 55
    - style 55
  - pie ordering 55



- special properties 53
- start angle 56
- thresholding 53
- use with unpick method 194
- pie-format tag 369
- Placement property 129
- plot area, definition 394
- plot chart type 22
- plot chart, definition 394
- plot1.java demo program 35
- plot2.java demo program 39
- PlotArea
  - property summary 310
- plot-area tag 370
- PNG 163
- PNG file format 187, 188
- PNG, definition 394
- point data, definition 395
- point labels, definition 395
- point-label tag 371
- PointLabels
  - axis annotation 124
  - ChartDataView 124
  - use with bar charts 124
  - use with pie charts 124
  - use with stacking bar charts 124
- polar chart 24
  - axis 28
  - definition 395
  - FastUpdate 190
- polar charts 58
  - angles 59
  - array data format 60
  - axis direction 131
  - background 58
  - data array 74
  - data format 60
  - default annotation 122
  - fill style 156
  - full range x-axis 61
  - general data format 60
  - half-range 61
  - half-range x-axis 61
  - mapping 192
  - max value 130
  - min value 130
  - negative values 62
  - OriginBase property 59
  - picking 193
  - point labels 124
  - PolarChartDraw class 61
  - r value 58
  - setting origin 59
  - theta 58
  - x- and y-values 59
- PolarChartDraw 61
- polar-radar-format tag 371
- positioning chart elements 163
  - location 163
  - size 163
- pre-formatted data 73
- procedure, definition 395
- programming
  - advanced 187
- programming actions 190
- programming basics
  - collections 28
- properties
  - 100Percent 49, 51
  - access in IDE 28
  - Anchor 141
  - AnnotationMethod 27, 41, 122, 126, 245
  - AnnotationRotation 134
  - AssociatedWithYAxis 168, 169, 178, 179
  - AxisAnnotation 255
  - axisOrientation 237
  - Background 162
  - background 239
  - Bean property overview 245
  - ChartLabel 175
  - ChartType 22, 42
  - chartType 238
  - ClusterOverlap 50
  - ClusterWidth 50
  - Color 55, 156, 157, 161
  - Constant 260
  - CustomShape 157
  - DataView 141
  - Depth 164
  - DrawnBeforeData 174, 175
  - Editable 259
  - editing and viewing in the customizer 32
  - Elevation 164
  - EndLineStyle 183
  - EndPoint 172
  - ExplodeList 56
  - ExplodeOffset 56
  - FastAction 189
  - FastUpdate 189
  - FillStyle 156, 182
  - Font 161
  - font 238
  - footerText 240
  - Foreground 162
  - foreground 239
  - GridSpacing 136
  - GridStyle 136
  - HalfRange 61
  - headerText 240
  - HorizActionAxis 192
  - IncludedInDataBounds 177, 184
  - IsBatched 189

- IsComplex 52
- IsLogarithmic 133
- IsOpenCloseFullWidth 52
- IsReversed 131
- IsShowing 40
- IsShowingClose 52
- IsShowingOpen 52
- legendAnchor 241
- legendIsShowing 241
- legendOrientation 241
- LineStyle 156, 173, 175
- Max 132
- Min 132
- MinSlices 55
- MultiChart reference 255
- Multiplier 260
- Origin 132
- Originator 260
- OriginBase 59
- OriginPlacement 132
- OtherLabel 55
- OtherStyle 55
- OutlineStyle 157
- parameter reference 315
- Pattern 55, 156
- RadarCircularGrid 136
- Rotation 134, 164
- rotation 256
- setting interactively at run-time 27
- Shape 157
- Size 157
- SortOrder 55
- special 47
- StartAngle 56
- StartLineStyle 183
- StartPoint 172
- SymbolStyle 157
- syntax 315
- Text 40
- ThresholdMethod 54
- TimeBase 126
- TimeFormat 126
- TimeUnit 126
- Title (axis) 134
- VertActionAxis 192
- View3D 239, 273
- Visible 138
- VisibleInLegend 175, 184
- Width 156
- xAnnotationMethod 235
- xAxisGridIsShowing 237
- xAxisIsLogarithmic 236
- xAxisIsShowing 236
- xAxisMinMax 236
- xAxisNumSpacing 236
- xAxisTitleText 235
- yAnnotationMethod 235
- yAxisGridIsShowing 237
- yAxisIsLogarithmic 236
- yAxisIsShowing 236
- yAxisMinMax 236
- yAxisNumSpacing 236
- yAxisTitleText 235
- property parameter, definition 395
- property sheet, definition 395
- property summary
  - ChartDataView 277
  - ChartDataViewSeries 280
  - ChartText 281
  - JCAAnno 282
  - JCAreaChartFormat 284
  - JCAxis 284
  - JCAxisFormula 290
  - JCAxisTitle 290
  - JCBarChartFormat 292
  - JCCandleChartFormat 292
  - JCChart 293
  - JCChartArea 295
  - JCChartLabel 296
  - JCChartLabelManager 297
  - JCChartStyle 297
  - JCFillStyle 299
  - JCGrid 299
  - JCGridLegend 300
  - JCHiloChartFormat 301
  - JCHLOCCChartFormat 301
  - JCLegend 302
  - JCLineStyle 303
  - JCMarker 304
  - JCMultiColLegend 305
  - JCPieChartFormat 306
  - JCPolarRadarChartFormat 307
  - JCSymbolStyle 308
  - JCThreshold 309
  - JCValueLabel 310
  - PlotArea 310
  - SimpleChart 311
- property, definition 395
- PS file format 187, 188

## R

- r value, definition 395
- radar 28
- radar chart 25
  - FastUpdate 190
- radar charts 62
  - annotation method 122
  - background 63
  - data array 74
  - data format 63

- definition 395
- mapping 192
- min value 130
- picking 193
- point labels 124
- RadarChartDraw class 63
- RadarChartDraw 63
- related documents 16
- rendered chart, definition 395
- rendering, definition 396
- result set
  - data set implementations 90
  - data, adding to a chart 87
  - review 86
- reversing an axis 130
- RGB
  - value, definition 396
- RMI, definition 396
- rotation 256
  - definition 396
- run-time
  - setting properties 27

## S

- scatter plot chart type 22
- scatter plot charts
  - definition 396
- series label, definition 396
- series-point tag 372
- set object properties 27
  - Java code 27
- set properties
  - IDE 28
  - interactively 27
  - run-time 27
- setFillGraphics() 146
- setLegendPopulator() 151
- setLegendRenderer() 151
- setText 154
- setting properties in an IDE 28
- setting properties interactively at run-time 27
- SimpleChart
  - 3D effects 239
  - axis annotation method 235
  - axis number intervals 236
  - axis orientation 237
  - axis properties 234
  - axis range 236
  - axis titles 235
  - Bean 233
  - chart types 238
  - data interpretation 238
  - data loading 242, 268
  - font 238

- footer 240
- foreground and background colors 239
- header 240
- hiding axes 236
- legend layout 241
- legend placement 241
- legends 240
- logarithmic notation 236
- property summary 311
- showing grids 237
- showing the legend 241
- tutorial 246
  - using Swing TableModel data object 270
  - using Swing TableModel data objects 243
- special terms 21
- stacking area chart 23
- stacking area charts
  - 100 percent axis 49
- stacking bar chart type 23
- stacking bar charts
  - 100 percent axis 51
  - 3D effect 164
  - definition 396
  - labelling with PointLabels 124
- standard area chart
  - definition 396
- start angle, definition 397
- StartAngle property 56, 329
- start-line-style tag 372
- stock data source, definition 397
- String, definition 397
- style 139
- subclass, definition 397
- subcomponent, definition 397
- support 17
  - JClass Community 18
- Swing TableModel
  - loading data 70
- Swing TableModel object
  - use with SimpleChart 243, 268, 270
- Swing, definition 397
- SwingDataModel 70
- symbol-style tag 373

## T

- TableModel 70
  - use with SimpleChart 243
- TableModel, use with SimpleChart 268, 270
- targeted data model 85
  - adding data from a result set 87
  - DefaultDataModel, creating an instance of 89
  - overview 86
  - setting, on the chart 90
  - understanding 112

- technical support 17
- terminology 21
- text 139
  - adding label 154
  - formatting label 154
  - internationalizing, in XML-based charts 229
- Text property 154
- TexturePaint 299
- theta, definition 397
- threshold tag 374
- thresholds
  - boundary lines, adding 183
  - fill style, setting 182
  - including values in data bounds calculations 184
  - intersecting 181
  - labels in legend 184
  - overlapping 181
  - overview 177
  - troubleshooting missing 184
  - x-axis, creating 178
  - y-axis, creating 179
- tick, definition 397
- time format 126
- time label
  - definition 397
- TimeLabels 125
- Title property (axis) 134
- title text
  - header and footer 139
- titled-border tag 374
- top-level object, definition 398
- transparent images 163
- transposed data, definition 398
- tutorial 35
  - SimpleChart Bean 245
- typographical conventions 14

## U

- underlying data model 67
- unmap 192, 199
- unpick 194, 199
- updating chart data source 82
- URL
  - loading DataSource from 68

## V

- valid modifier 191
- value annotation 235
- value labels
  - definition 398
- Value\_Labels notation 235
- value-label tag 375

- ValueLabels axis annotation 124
- values annotation 122
- valueToDate method 127, 128
- VisibleInLegend property 141

## X

- x-axis
  - full or half-range 61
  - markers, creating 168
  - thresholds, creating 178
  - when chart inverted 131
  - when logarithmic 133
- x-data tag 378
- XML 71
  - background 219
  - definition 388
  - DTD primer 220
  - internationalizing text 229
  - interpreter 71
  - labels and other parameters 73
  - links 220
  - loading data 70
  - primer 219
  - specifying data by point 72
- XSL, definition 389
- XSLT
  - definition 389

## Y

- y-axis
  - markers, creating 169
  - second y-axis 137
  - thresholds, creating 179
  - when chart inverted 131
- y-data tag 379

## Z

- zooming, definition 398