

JClass Chart 3D™

Programmer's Guide

Version 6.3

for Java 2 (JDK 1.3.1 and higher)

JClass Chart 3D – Stunning, Interactive 3D Charts for Java



8001 Irvine Center Drive
Irvine, CA 92618
949-754-8000
www.quest.com

© Copyright Quest Software, Inc. 2004. All rights reserved.

This guide contains proprietary information, which is 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.

Warranty

The information contained in this document is subject to change without notice. Quest Software makes no warranty of any kind with respect to this information. QUEST SOFTWARE SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTY OF THE MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Quest Software shall not be liable for any direct, indirect, incidental, consequential, or other damage alleged in connection with the furnishing or use of this information.

Trademarks

JClass, JClass Chart, JClass Chart 3D, JClass DataSource, JClass Elements, JClass Field, JClass HiGrid, JClass JarMaster, JClass LiveTable, JClass PageLayout, JClass ServerChart, JClass ServerReport, JClass DesktopViews, and JClass ServerViews are trademarks of Quest Software, Inc. Other trademarks and registered trademarks used in this guide are property of their respective owners.

World Headquarters
8001 Irvine Center Drive
Irvine, CA 92618
www.quest.com
e-mail: info@quest.com
U.S. and Canada: 949.754.8000

Please refer to our Web site for regional and international office information.

This product includes software developed by the Apache Software Foundation <http://www.apache.org/>.

The JPEG Encoder and its associated classes are Copyright © 1998, James R. Weeks and BioElectroMech. This product is based in part on the work of the Independent JPEG Group.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1.Redistributions of source code must retain the above copyright notice, this list of conditions, all files included with the source code, and the following disclaimer.
- 2.Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes software developed by the JDOM Project (<http://www.jdom.org/>). Copyright © 2000-2002 Brett McLaughlin & Jason Hunter, all rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1.Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
- 2.Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.
- 3.The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jdom.org.
- 4.Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management (pm@jdom.org).

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

Preface	1
Introducing JClass Chart 3D	1
Assumptions	2
Typographical Conventions Used in this Manual	3
Overview of Manual	3
API Reference	4
Licensing	4
Related Documents	4
About Quest	5
Contacting Quest Software	6
Customer Support	6
Product Feedback and Announcements	7

Part I: Using JClass Chart 3D with the Java 2 and Java 3D API

1 JClass Chart 3D Basics	11
1.1 Terminology	11
1.2 Startup Checklist	12
1.3 Instantiating a Chart in JClass Chart 3D	12
1.4 Data Types	13
1.5 Chart Types	14
1.6 Loading Data	16
1.7 Setting and Getting Object Properties	16
1.8 Other Programming Basics	18
1.9 Outputting JClass Chart 3D	19
1.10 JClass Chart 3D Inheritance Hierarchy	20
1.11 JClass Chart 3D Object Containment	21
1.12 UseDefault Properties	22
1.13 Batching Property Updates	23
1.14 Chart Colors	23
1.15 The JClass Chart 3D Customizer	25
2 Programming JClass Chart 3D: Common Functions	27
2.1 Properties	27

2.2	Axis Controls	28
2.3	Setting Axis Bounds	29
2.4	Legends	29
2.5	Perspective	37
2.6	Axis Scaling	38
2.7	Axis Labelling and Annotation Methods	39
2.8	Gridlines	45
2.9	Header and Footer Titles	47
2.10	Adding Header, Footer, and Labels	47
3	Programming JClass Chart 3D: Surfaces and Bars.	49
3.1	Fifteen Basic Types of Surfaces and Bars	49
3.2	Chart Types	52
3.3	Bar Charts and Histograms	55
3.4	Contours and Zone Display	58
3.5	Mesh Controls	61
3.6	Surface Colors	62
3.7	Solid Surface	63
4	Programming JClass Chart 3D: Scatter Plots	65
4.1	Overview	65
4.2	Three Basic Types of Scatter Plots	66
4.3	Controlling Symbol and Drop Line Style	66
4.4	Chart Styles	67
5	Data Sources	69
5.1	Overview	69
5.2	Pre-Built Chart DataSources	73
5.3	Loading Data from a File	74
5.4	Loading Data from a Swing TableModel	77
5.5	Loading Data from an XML Source	78
5.6	Data Binding using JDBCDataSource	81
5.7	JCData3dUtil class	81
5.8	Making Your Own Chart Data Source	82
5.9	HoleValueChartDataModel – Specifying Hole Values	87
5.10	Making an Updating Chart Data Source	88
5.11	Summary of JClass Chart 3D Data Interfaces	91

6	Advanced JClass Chart 3D Programming	93
6.1	4D Surface Graphs	93
6.2	4D Bar Charts	94
6.3	Customizing the Contour Levels	95
6.4	Customizing Contour Styles	96
6.5	Internationalization Support	98
7	Programming User Interaction	99
7.1	Default User Interaction	99
7.2	Listeners	104
7.3	Mapping and Picking	105
7.4	dragZValue Method	106
7.5	gridValue Method	107
8	Programming with the Java 3D API	109
8.1	Java 3D – Overview	109
8.2	System Set-up	110
8.3	Browsers and Java 3D	111
8.4	Java 3D API	111
8.5	SceneGraphObject class	114
8.6	Scene Graph Viewing Object Classes	119
8.7	BranchGroup and TransformGroup	119
8.8	Rendering	122
8.9	Behaviors	123
8.10	Java 3D-Enabled Charting Features	123

Part II: Reference Appendices

A	Interface Listing	135
A.1	Interface Summary	135
B	Object Property Listing	137
B.1	Chart3D	137
B.2	Chart3d.Event	150
B.3	Chart3d.j2d	150
B.4	Chart3d.j3d	150

C	Additional Common JClass Chart 3D 3D Methods	151
C.1	Chart3D	151
C.2	Chart3d.Event	159
Index	161

Preface

[Introducing JClass Chart 3D](#) ■ [Assumptions](#) ■ [Typographical Conventions Used in this Manual](#)
[Overview of Manual](#) ■ [API Reference](#) ■ [Licensing](#) ■ [Related Documents](#) ■ [About Quest](#)
[Contacting Quest Software](#) ■ [Customer Support](#) ■ [Product Feedback and Announcements](#)

Introducing JClass Chart 3D

JClass Chart 3D is a powerful Java 3D charting tool. It enables developers to build 3D data views and interactive displays, giving their applications a professional look and feel.

JClass Chart 3D allows you to create stunning 3D graphics using either the Java 2 API or the Java 3D API. Please note that all methods, functions, and so on, relate to both APIs unless specifically mentioned.

With JClass Chart 3D, you can add 3D functionality using Java 2 technology – meaning that the heavyweight Java 3D API won't need to be included in your applications. However, JClass Chart 3D gives you the option to leverage the elegance of the Java 3D API (3D will need to be on the client).

JClass Chart 3D is 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 3D properties programmatically.
- OO developers, instantiating and extending JClass Chart 3D objects.
- JavaBean developers, setting JClass Chart 3D properties using a third-party Integrated Development Environment (IDE).

JClass Chart 3D 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.

Note: The Java 3D version of JClass Chart 3D is supported only for JDK 1.3.1 with Java 3D 1.2.1_03 on Windows NT with OpenGL with an ATI RAGE XL PCI video card with these drivers: ati2mpad.sys 4.00.1381.1006 and ati2drad.dll 4.0.0.

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

Feature Overview

You can set the properties of JClass Chart 3D objects to determine how the chart will look and behave. You can control:

- Chart type (surface, bar, or scatter plot).

- Header and footer positioning, border style, text, font, and color.
- Number of data views, each having its own data, and chart type. Currently, only one data view is supported.
- Flexible data loading from files, URLs, input streams, and databases.
- Chart styles for scatter plots: line color, fill color, point size, point style, and point color.
- Contour levels: Use the default linear distribution or provide your own.
- Contour styles: Contour line styles and contour zones colors are all customizable.
- Plot cube scaling in x, y, z directions.
- Legend positioning, orientation, border style, layout style, distribution range limiting, anchor, font, and color.
- Chart positioning, border style, color, width, height and axes rotation.
- Axis labelling using Data labels, Value labels, or pleasing precompiled values.
- An X, Y and Z axes, each having its own minimum and maximum, gridlines, annotation method, font, and title.
- Control of user interaction with components including picking, mapping, Chart 3D Customizer, rotation, scaling, and translation.
- Ability to add a 4th dimension to the data via color.
- Contour maps through 2D projections of data.
- Hidden-line display, surface colors, and mesh colors.
- 3D rotation and perspective.

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, including useful references to the Java 3D API.

Typographical Conventions Used in this Manual

- Typewriter Font
- Java language source code and examples of file contents.
 - JClass Chart 3D 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 Manual

Part I – Using JClass Chart 3D with the Java 2 and Java 3D API – describes programming with JClass Chart 3D using the Java 2 and the Java 3D API.

Chapter 1, [JClass Chart 3D Basics](#), provides a programmer's overview of JClass Chart 3D. This chapter covers concepts and vocabulary used in JClass Chart 3D programming, and discusses class hierarchy, object containment, terminology, and programming basics.

Chapter 2, [Programming JClass Chart 3D: Common Functions](#), introduces properties, axis information (controls, scaling, labelling, and annotating), legends, gridlines, and header and footer titles.

Chapter 3, [Programming JClass Chart 3D: Surfaces and Bars](#), is a guide to the basic types of surfaces and bars; meshed, shaded, and transparent plots; contoured and zoned plots; bar charts and histograms; contours and zone display; surface colors; and solid surfaces.

Chapter 4, [Programming JClass Chart 3D: Scatter Plots](#), is all about scatter plots, including the three basic types of scatter plots (3D scatter plots, 3D scatter plots with drop lines, and 2D scatter plots), controlling symbol and drop line styles, and developing chart styles.

Chapter 5, [Data Sources](#), introduces you to data sources (including an extensive overview), pre-built chart datasources, loading data from a file and from an XML

source, data binding, specifying data from databases, and making your own chart data source and an updating chart data source.

Chapter 6, [Advanced JClass Chart 3D Programming](#), covers advanced JClass Chart 3D topics, including 4D surface graphs, 4D bar charts, customizing the contour levels, customizing contour styles, and internationalization support.

Chapter 7, [Programming User Interaction](#), provides a look at default user interaction, listeners (data listener, Chart3d listener, and pick listener), mapping and picking, and the interpolate method.

Chapter 8, [Programming with the Java 3D API](#), provides an overview of the Java 3D API, and then delves into leveraging the power of the Java 3D API with JClass Chart 3D.

Part II – Reference Appendices – contains detailed technical reference information.

Appendix A, [Interface Listing](#), summarizes the commonly used JClass Chart 3D interfaces.

Appendix B, [Object Property Listing](#), lists the properties for all commonly used classes for the Java 2 API.

Appendix C, [Additional Common JClass Chart 3D 3D Methods](#), lists the most frequently used classes for JClass Chart 3D.

API Reference

The [API](#) reference documentation (Javadoc) is installed automatically when you install JClass Chart 3D and is found in the `JCLASS_HOME/docs/api/` directory.

Licensing

In order to use JClass Chart 3D, you need a valid license. Complete details about licensing are outlined in the [JClass Desktop Views Installation Guide](#), which is automatically installed when you install JClass Chart 3D.

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 at <http://java.sun.com/beans/resources.html>
- For a comprehensive introduction to Java 3D, VRML97, MPEG-4/BIFS, and X3D, see *Core Web3D* from Prentice Hall. See the publisher's Web site at <http://vig.prenhall.com/> or the book's Web page at <http://www.CoreWeb3D.com>
- Learn about the Web3D Consortium (provides a forum for the creation of open standards for Web3D specifications) at <http://www.web3d.org/>
- For information about the Java 3D API, please see <http://java.sun.com/products/java-media/3D/>

These documents are not required to develop applications using JClass Chart 3D, but they can provide useful background information on various aspects of the Java programming language.

About Quest

Quest Software, Inc. (NASDAQ: QSFT) is a leading provider of application management solutions. Quest provides customers with Application Confidencesm by delivering reliable software products to develop, deploy, manage and maintain enterprise applications without expensive downtime or business interruption. Targeting high availability, monitoring, database management and Microsoft infrastructure management, Quest products increase the performance and uptime of business-critical applications and enable IT professionals to achieve more with fewer resources. Headquartered in Irvine, Calif., Quest Software has offices around the globe and more than 18,000 global customers, including 75% of the Fortune 500. For more information on Quest Software, visit www.quest.com.

Contacting Quest Software

E-mail	sales@quest.com
Address	Quest Software, Inc. World Headquarters 8001 Irvine Center Drive Irvine, CA 92618 USA
Web site	www.quest.com
Phone	949.754.8000 (United States and Canada)

Please refer to our **Web site** for regional and international office information.

Customer Support

Quest Software's world-class support team is dedicated to ensuring successful product installation and use for all Quest Software solutions.

SupportLink www.quest.com/support

E-mail support@quest.com

You can use SupportLink to do the following:

- Create, update, or view support requests
- Search the knowledge base, a searchable collection of information including program samples and problem/resolution documents
- Access FAQs
- Download patches
- Access product documentation, [API](#) reference, and demos and examples

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 DesktopViews Installation Guide](#) (available in HTML and PDF formats) for help with these types of problems.

To Contact JClass Support

Any request for support *must* include your JClass product serial number. Supplying the following information will help us serve you better:

- Your name, email address, telephone number, company name, and country

- The product name, version and serial number
- The JDK (and IDE, if applicable) that you are using
- The type and version of the operating system you are using
- Your development environment and its version
- A full description of the problem, including any error messages and the steps required to duplicate it

JClass Direct Technical Support	
JClass Support Email	support@quest.com
Telephone	949-754-8000
Fax	949-754-8999
European Customers Contact Information	Telephone: +31 (0)20 510-6700 Fax: +31 (0)20 470-0326

Product Feedback and Announcements

We are interested in hearing about how you use JClass Chart 3D, any problems you encounter, or any additional features you would find helpful. The majority of enhancements to JClass products are the result of customer requests.

Please send your comments to:

Quest Software
8001 Irvine Center Drive
Irvine, CA 92618

Telephone: 949-754-8000
Fax: 949-754-8999

Part I

*Using JClass
Chart 3D
with the
Java 2 and
Java 3D API*

JClass Chart 3D Basics

Terminology ■ *Startup Checklist* ■ *Instantiating a Chart in JClass Chart 3D*
Data Types ■ *Chart Types* ■ *Loading Data* ■ *Setting and Getting Object Properties*
Other Programming Basics ■ *Outputting JClass Chart 3D* ■ *JClass Chart 3D Inheritance Hierarchy*
JClass Chart 3D Object Containment ■ *UseDefault Properties* ■ *Batching Property Updates*
Chart Colors ■ *The JClass Chart 3D Customizer*

All elements mentioned in this chapter refer to both the Java 2 API and the Java 3D, API unless specifically noted.

Using JClass Chart 3D with the Java 3D API is discussed in detail in the chapter entitled [Programming with the Java 3D API](#).

If a class name has “Java3d” before the .java extension, it will most likely be specific to the Java 3D API. For instance, the `JCPlotCube` class in the Java 2 API is called `JCPlotCube.java`, while its counterpart in the Java 3D API, `JCPlotCubeJava3d`, is `JCPlotCubeJava3d.java`

This chapter covers concepts and vocabulary used in JClass Chart 3D programming, and provides an overview of the JClass Chart 3D class hierarchy.

1.1 Terminology

A JClass Chart 3D chart comprises four components: header, footer, chart area, and legend. The plot cube, contained within the chart area, contains the rendered chart.

The following illustration shows the terms used to describe the main components that make up a chart:

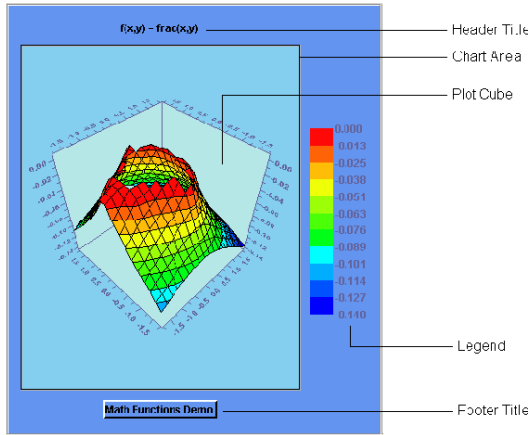


Figure 1 Elements contained in a typical chart.

Plot Cube

The *plot cube* is defined to be the smallest cube which encloses the entire 3D scene (including the axes). Some JClass Chart 3D properties, such as those that specify axis scaling and axis font cube sizes, have definitions that depend on the plot cube size.

The plot cube has properties that include background color, foreground color, floor and ceiling properties, and x, y, and z scaling.

1.2 Startup Checklist

Full details of how to get started with JClass Chart 3D are provided in the [JClass DesktopViews Installation Guide](#). The *JClass DesktopViews Installation Guide* is provided in PDF and HTML formats, and is automatically installed into `JCLASS_HOME/docs/getstarted/` when you install JClass Chart 3D.

Note: A reminder that `jchart3dj2d.jar` contains information for the Java 2 API version of JClass Chart 3D, while `jchart3dj3d.jar` contains information for both the Java 2 and Java 3D API versions of JClass Chart 3D. Both files are included in JClass Chart 3D.

1.3 Instantiating a Chart in JClass Chart 3D

To instantiate a chart, you may choose to use one of the three available factory methods in `JCChart3d`:

- `public static JCChart3d createJava2dChart();`
- `public static JCChart3d createJava3dChart();`
- `public static JCChart3d createJava3dChart(boolean fallback);`

The `createJava2dChart()` method creates an instance of the Java 2 version of JClass Chart 3D, which is the class `com.klg.jclass.chart.j2d.JCChart3dJava2d`.

The `createJava3dChart()` method creates an instance of the Java 3D version of JClass Chart 3D, which is the class `com.klg.jclass.chart.j3d.JCChart3dJava3d`.

The `createJava3dChart(boolean fallback)` method creates an instance of the Java 3D version of JClass Chart 3D, which is the class `com.klg.jclass.chart.j3d.JCChart3dJava3d`. Note that if this class cannot be created (for instance, because Java 3D is not installed) and if the ‘fallback’ parameter is true, then the Java 2 version of JClass Chart 3D will be created.

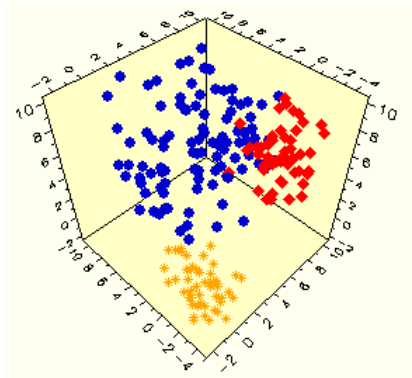
All three methods will return `null` if JClass Chart 3D could not create the desired classes.

1.4 Data Types

In JClass Chart 3D, there are two types of data: point data and grid data.

Point data

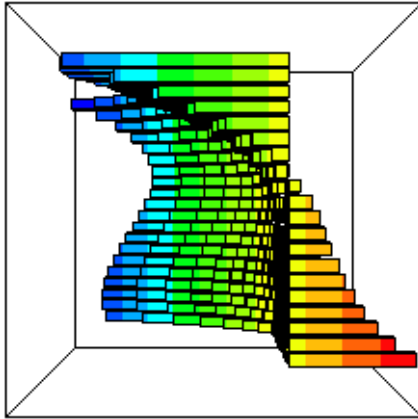
Point data comprises one or more series of points. Each of these series can have its own chart style.



Grid data

Grid data comprises an array of X-values, an array of Y-values, and a corresponding array of (x,y) values.

Note that the following figure uses grid data for plotting, and has been rotated so that the X-Y plane is vertical.

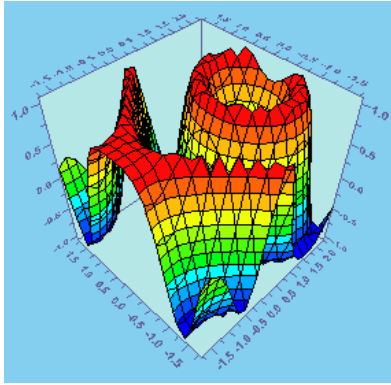


1.5 Chart Types

JClass Chart 3D contains three chart types: surface, bar, and scatter plot.

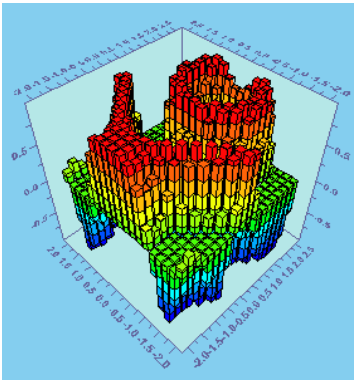
Surface Chart

A surface chart uses only grid data.



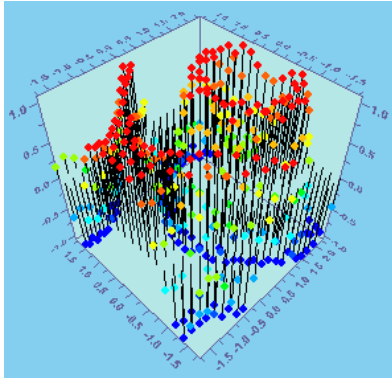
Bar Chart

A bar chart uses only grid data.



Scatter Plot Chart

A scatter plot chart can use either grid or point data.



1.6 Loading Data

Data is loaded into a chart by attaching one or more chart data sources to it. A chart data source is an object that takes real-world data and puts it into a form that JClass Chart 3D can use. Once your data source is attached, you can chart the data in a variety of ways.

Several stock (built-in) data sources are provided with JClass Chart 3D, enabling you to read data from an input stream, a file, and a URL, and databases. Loading data from a database is called ‘data binding’. In JClass Chart 3D, you are able to get data from a database. You can also create your own data sources. See the [Data Sources](#), in Chapter 5, for more information on loading data and creating your own data sources.

1.7 Setting and Getting Object Properties

There are three ways to set (and retrieve) JClass Chart 3D properties:

- By calling property set and get methods in a Java program.
- By using a Java IDE at design-time (JavaBeans).
- By using the JClass Chart 3D Customizer at run-time.

Each accessor method changes the chart property whose name matches the method. This manual therefore uses properties to discuss how features work, rather than using the method or Customizer tab that 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 [JClass Chart 3D Object Containment](#) diagram later in this chapter to determine how to access the properties of an object.

1.7.1 Setting Properties with Java Code

Every JClass Chart 3D property has a set and get method associated with it, unless it is read-only. For example, to retrieve the value of the `AnnotationMethod` property of the X-axis, the `getAnnotationMethod()` method is called:

```
method = c.getChart3dArea().getAxis(JCAxis.AXIS_X).
    getAnnotationMethod();
```

To set the `AnnotationMethod` property of the same axis, the `setAnnotationMethod` is called:

```
c.getChart3dArea().getAxis(JCAxis.AXIS_X).setAnnotationMethod(
    JCAxis.ANNOTATION_VALUES);
```

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 `Chart3dArea` property is a `JCChart3dArea` object. The chart area has an `Axes` property, the value of which is a collection of `JCAxis` objects. The X-axis is indexed using the `JCAxis.AXIS_X` enum value, and the axis has the desired `AnnotationMethod` property. Note that for convenience, the `JCChart3dArea` class has a `getXAxis()` method.

For detailed information on the properties available for each object, consult the [API reference documentation](#), which is automatically installed into *JCLASS_HOME/docs/api/index.html* when you install JClass Chart 3D.

1.7.2 Setting Properties with a Java IDE at Design-Time

JClass Chart 3D 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 JavaBean components into the IDE.

Please refer to the JClass and Your IDE chapter in the [JClass Desktop Views Installation Guide](#), which outlines detailed instructions and important notes. For instance, only the Java 2 version of the JClass Chart 3D JavaBean (`chart3dJava2d`) works in Borland JBuilder 4 or higher, and you will need to add *vecmath.jar* to your project. **The readme file contains the most current list of supported IDEs.**

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.

1.7.3 Setting Properties Interactively at Run-Time

If enabled by the developer, end-users can manipulate property values on a chart running in your application. Right-clicking the mouse launches the JClass Chart 3D Customizer. The user can navigate through the tabbed dialogs and edit the properties displayed.

For details on enabling and using the Customizer, see Section 1.15, [The JClass Chart 3D Customizer](#).

1.8 Other Programming Basics

1.8.1 Working with Object Collections

Many chart objects are organized into collections. For example, the contour styles are organized into a `java.util.ArrayList`. In JavaBeans terminology, these objects are held in indexed properties.

To access a particular element of a collection, you need to retrieve the collection and then specify the index that uniquely identifies this element. For example, the following code changes the line color of the third contour style to red.

```
import java.awt.Color;
import java.util.ArrayList;
import com.klg.jclass.chart3d.*;

ArrayList styles = c.getDataView(0).getContour().getContourStyles();
JCContourStyle cStyle = (JCContourStyle)styles.get(2);
cStyle.getLineStyle().setColor(Color.red);
```

Note that the index 0 refers to the first element of a collection.

1.8.2 Calling Methods

To call a JClass Chart 3D method, access the object that defines the method. For example, the following statement uses the `coordToDataCoord()` method, defined by the `Chart3dDataView` collection, to convert a pixel value to its equivalent in data coordinates:

```
javax.vecmath.Point3d dc = c.getDataView(0).coordToDataCoord(10,15);
```

Details on each method can be found in the API documentation for each class.

1.8.3 Eliminating Retained JCChart3dJava3d References

When an application or applet that has created an instance of `JCChart3dJava3d` is terminated, the resources used by Java 3D are automatically released. However, if a `JCChart3dJava3d` object needs to be garbage collected before the application or applet exits, then steps must be taken to ensure that all references to the `JCChart3dJava3d` instance are eliminated. At least one of the objects that are passed to Java 3D during the creation of a `JCChart3dJava3d` instance contains a reference to that instance.

The `dispose()` method of `JCChart3dJava3d` eliminates all references to the `JCChart3dJava3d` objects that are retained by Java 3D. It does this by calling the `removeAllLocales()` method of the `VirtualUniverse` object that contains the `Chart3d`. An application or applet must call the `dispose()` method at a point where the `JCChart3dJava3d` object is no longer needed.

1.9 Outputting JClass Chart 3D

Many applications require that the user has a way to get an image or a hard copy of a chart. JClass Chart 3D allows you to output your chart as a GIF, PNG, or JPEG image, to either a file or an output stream. If you are using the Java 3D version of JClass Chart 3D, the chart must be fully visible on the screen to be correctly encoded. This means that if part of the image is obscured, then the obscured portion will not draw to the requested image.

Please note that in order to enable GIF encoding, you must obtain a license from Unisys and send a copy of this license to Quest Software. Quest Software will send the enabling software for GIF encoding upon receipt of a valid proof of license. There are also public sources of Java image to GIF converters.

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.

1.9.1 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 (the chart), the `encoding` parameter refers to the type of encoding to use (a GIF, PNG, or JPEG), 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.

1.9.2 Encode example

The following code sample encodes a 3D chart into a JPEG file:

```
try {  
    JCEncodeComponent.Encoding encoding=JCEncodeComponent.JPEG;  
    JCEncodeComponent.encode(encoding, chart3d, new File(filename));  
}  
catch (EncoderException ee){  
    ee.printStackTrace();  
}  
catch (IOException IO){  
    IO.printStackTrace();  
}
```

1.10 JClass Chart 3D Inheritance Hierarchy

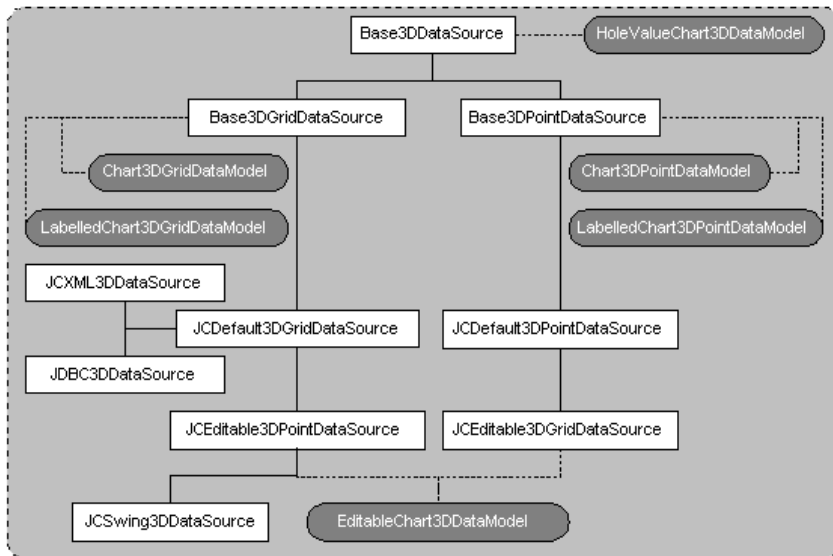
The following provides an overview of class inheritance of JClass Chart 3D.

Class Inheritance Hierarchy

Visuals:



Data Source:



Also:

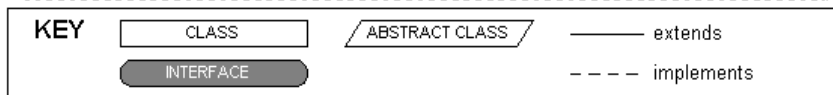
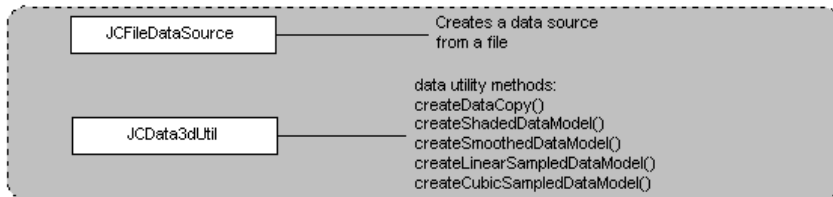


Figure 2 Class hierarchy of the com.klg.jclass.chart3d package.

1.11 JClass Chart 3D 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 3D.

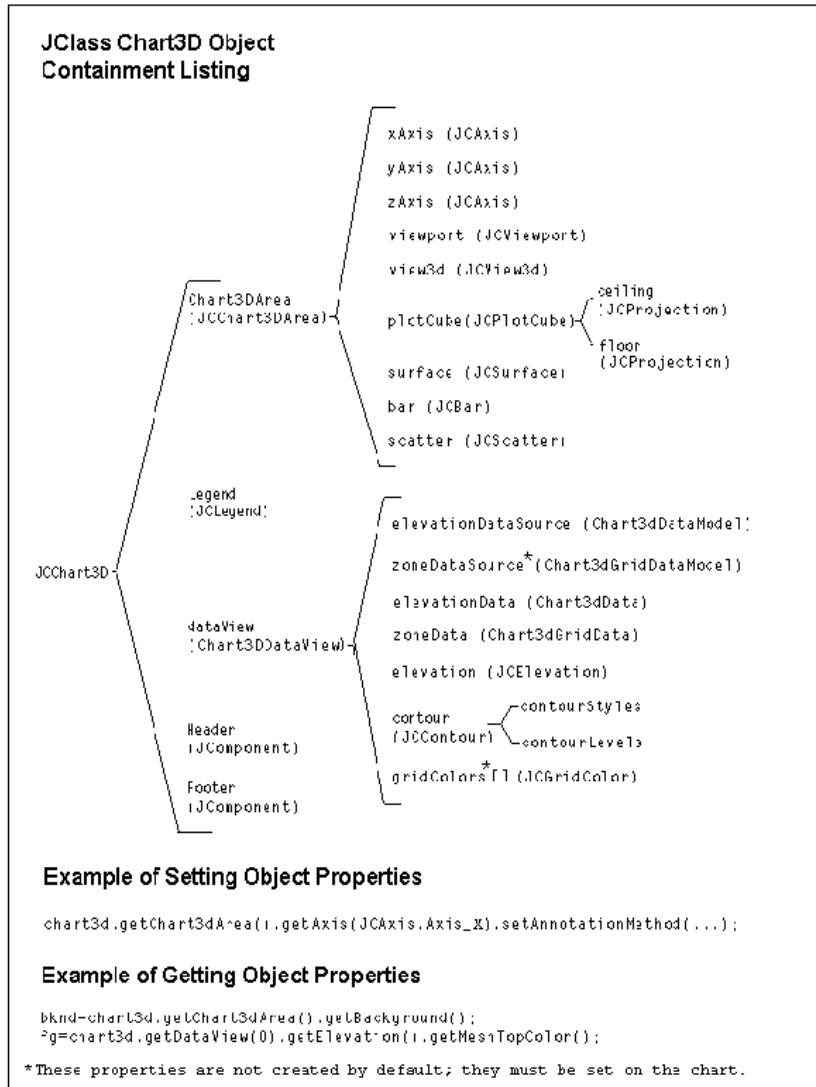


Figure 3 Objects contained in a chart; to access properties, traverse contained objects.

JCChart3D (the top-level object) manages header and footer JComponent objects, a legend (JCLegend), and the chart area (JCChart3DArea). The chart also contains a collection of data view (Chart3DDataView) objects. Please note that currently only one Chart3dDataView is supported.

The Chart3dArea contains many of the chart's actual properties because it is responsible for charting the data. It also contains and manages an xAxis, yAxis, and zAxis, all of type JCAxis.

The data view collection contains objects and properties (like the chart type) that are tied to the data being charted. Each data view manages two data sources: an **elevation data source** and a **zone data source**. The elevation data source is the main data source for plotting surfaces, bars, and scatter plots. Its two values determine the elevation of the surface, bar, or scatter point. It can be set to either grid data or point data.

The zone data source is used to add a fourth dimension to the elevation data via color. It can only be set to grid data and is only used if the elevation data source is also using grid data. Thus, the elevation data source must implement either the Chart3dGridDataModel or the Chart3dPointDataModel, while the zone data source must implement the Chart3dGridDataModel.

Note that the chart does not own the data itself, but instead merely views on the data. 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 3D's data source model, see [Data Sources](#), in Chapter 5.

1.12 UseDefault Properties

Three JClass Chart 3D properties have corresponding UseDefault properties:

- The max property of JCAxis.
- The min property of JCAxis.
- The levels property of JCContourLevels.

UseDefault properties are Booleans that determine whether JClass Chart 3D should calculate a default value for the property.

For example, if the minIsDefault property of a JCAxis object is true, every time the JClass Chart 3D data is changed, JClass Chart 3D will determine a reasonable default value for the axis minimum. If false, JClass Chart 3D will use the provided axis minimum.

A side effect of setting any property that has a corresponding UseDefault property is that the UseDefault property will be set to false.

The following code will freeze the value of `minIsDefault` at its current value. It will also have the side effect of setting `minIsDefault` to `false`.

```
JCAxis yAxis = c.getChart3dArea().getAxis(JAxis.Axis_Y);
yAxis.setMin(yAxis.getMin());
```

The following code will revert back to the default behavior, enabling JClass Chart 3D to calculate a default value for `minIsDefault` whenever it draws the graph.

```
yAxis.setMinIsDefault(true);
```

For the `levels` property of `JCContourLevels`, adding, removing, or setting a level directly will cause the `isDefault` property to be set to `false`. Also, when `isDefault` is `false`, the `numLevels` property becomes read-only.

1.13 Batching Property Updates

Normally property changes take effect immediately after the values are set. If you would prefer to make several changes to the chart's properties before causing a repaint, set the `setBatched()` method to `true`. The `setBatched()` method sets the value of the `Batched` property, which controls whether chart updates are accumulated; if set to `true`, chart updates will accumulate, and if set to `false`, the accumulated updates are forced to be processed.

You should normally set `setBatched()` to `true` after all your updates are made; this will initiate a repaint.

1.14 Chart Colors

Color can powerfully enhance a chart's visual impact. You can customize chart colors using Java color names or RGB values. Using an interactive tool like the JClass Chart 3D Customizer makes selecting custom colors quick and easy.

Note that the area backgrounds are transparent by default. The foreground colors default to the chart foreground color. Also note that inherited properties of the chart or `Chart3dArea` components, such as `backgroundColor`, are not controlled by the `Batched` property.

Each of the following visual elements in the chart has a background and foreground color that you can customize:

- the entire chart
- the Header and Footer titles
- the legend
- the chart area
- the plot cube

Other chart objects have color properties too, including `JCGridLines` and `JCChart3dStyles`. You can also specify colors for the top and bottom of the mesh, for surface shading, and for the contour lines and zone fills.

Color Defaults

All chart subcomponents are transparent by default with no background color. If made opaque, the legend and the chart area will inherit background color from the parent chart. The plot cube inherits its colors from the chart area. The same objects will always inherit the foreground color from the chart.

Headers and footers are independent objects that behave according to the rules of whatever object they are.

Please note that 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 color names defined in `java.awt.Color`. The following table summarizes these colors:

Built-in Colors in <code>java.awt.Color</code>		
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 `ContourStyles` and `Chart3dStyles` 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 3D` 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.

Transparency

If the JClass Chart 3D component is meant to have a transparent background, set the `opaque` property to `false`; then generated JPEGs, GIFs, and PNGs will also contain a transparent background (not currently available in the Java 3D API version of JClass Chart 3D).

1.15 The JClass Chart 3D Customizer

The JClass Chart 3D Customizer enables developers (and end-users if enabled by your program) to view and customize the properties of the chart as it runs.

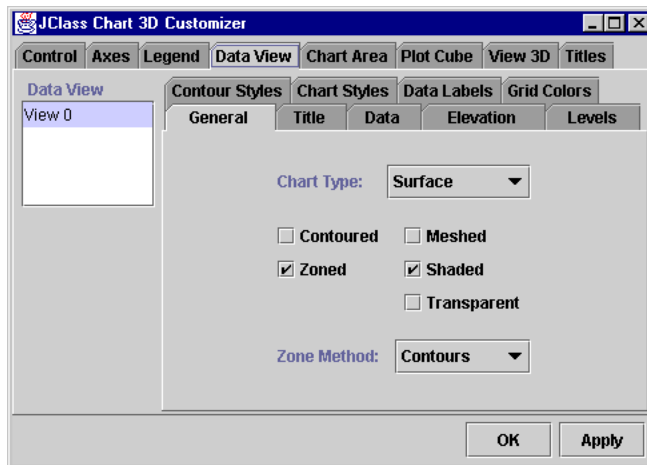


Figure 4 The JClass Chart 3D Customizer.

The 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.

1.15.1 Displaying the Chart Customizer at Run-Time

By default, the Customizer is disabled at run-time. To enable it, you need to set the chart's `AllowUserChanges` property to `true`. For example:

```
chart3d.setAllowUserChanges(true);  
chart3d.launchPropertyPage(new Point(x,y));
```

You can also launch the Customizer through the customize action. Please see [Programming User Interaction](#), in Chapter 7, for information on how to do this. Installing the default user interactions via the `addAllDefaultActions()` method causes the Customizer to be deployed when the right mouse button is clicked.

1.15.2 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 property values.
3. Select a property and edit its value.

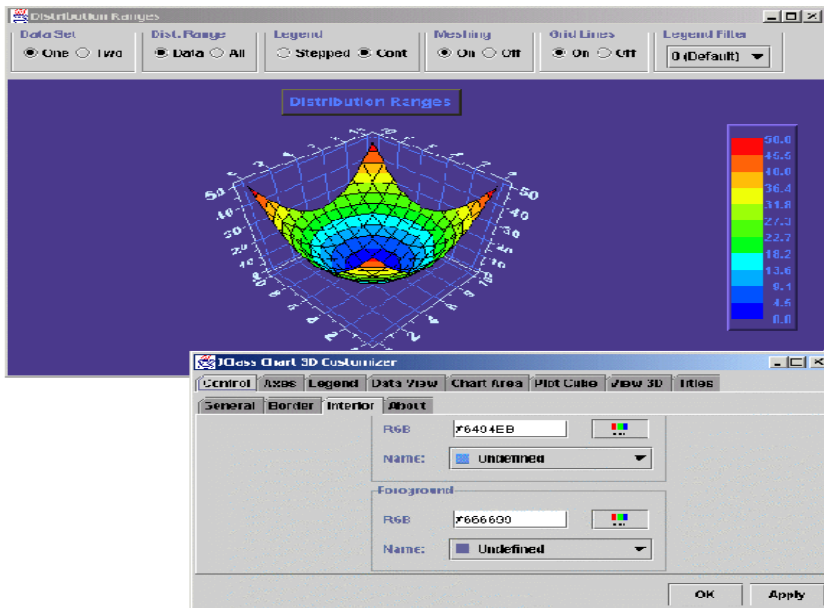


Figure 5 Editing a sample chart with the Customizer.

As you change property values, the changes are immediately applied to the chart, and will be displayed immediately only if the batched checkbox is not selected. You can make further changes without leaving the 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.

To close the Customizer, close its window (the actual steps differ for each platform).

Programming JClass Chart 3D: Common Functions

Properties ■ *Axis Controls* ■ *Setting Axis Bounds* ■ *Legends* ■ *Perspective Axis Scaling* ■ *Axis Labelling and Annotation Methods* ■ *Gridlines* ■ *Header and Footer Titles*
Adding Header, Footer, and Labels

All elements mentioned in this chapter refer to both the Java 2 API and the Java 3D API unless specifically noted.
--

2.1 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 JClass Chart 3D `JCAxis` class has a property called `annotationMethod`, which is used to indicate the style of annotation used on the axis. To set the property value, the `setAnnotationMethod()` method is used. To get the property value, the `getAnnotationMethod()` method is used.

It is not necessary to remember all the properties in order to program JClass Chart 3D effectively. For most charts, many properties may be left with their default settings. A full summary of the JClass Chart 3D properties for all commonly used classes is provided in [Appendix B](#). Scan through those tables to gather a basic understanding of the properties. Not all the properties are used for all types of charts: some are specific for Surface and Bar charts only, while others are for Scatter Plots only.

For complete details on how JClass Chart 3D’s object properties are organized, see [JClass Chart 3D Object Containment](#) and [Setting and Getting Object Properties](#), in Chapter 1.

2.1.1 Setting JavaBean Properties at Design-Time

JClass Chart 3D has two JavaBeans: `chart3dJava2d` for Java 2 and `chart3dJava3d` for the Java 3D API.

One of the features of any JavaBean component is that it can be manipulated interactively in a visual design tool (such as a Java IDE) to set the initial property values when the application starts. Consult your IDE's documentation for details on how to load third-party JavaBean components into the IDE.

For details on JClass Chart 3D's JavaBeans and IDEs, please refer to the JClass and Your IDE chapter in the [JClass DesktopViews Installation Guide](#). The *JClass DesktopViews Installation Guide* is available in HTML and PDF formats, and is included when you purchase JClass Chart 3D. **The readme file contains the most current list of supported IDEs.**

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.

2.1.2 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 3D Customizer. The user can navigate through the tabbed dialogs and edit the properties displayed.

For details on enabling and using the Customizer, please see [The JClass Chart 3D Customizer](#), in Chapter 1, as well as [Programming User Interaction](#), in Chapter 7.

2.2 Axis Controls

2.2.1 Axis Show

The `show` property of `JCAxis` tells JClass Chart 3D whether it should draw the axis at all. If set to `false`, the axis will not be drawn.

2.2.2 Axis Font and Size

The axis annotation is rendered using the font specified by the `annoFont` property of `JCAxis`. The font is scaled to be the size specified by the `annoFontCubeSize` property.

The `annoFontCubeSize` is measured in units, which are each 1/1,000 of the plot cube length. The default `annoFontCubeSize` is 80, which means the characters are 8% of the length of the plot cube high. Thus, if the plot cube changes size, so does the annotation.

2.2.3 Title

The `JCAxis` `title` property may be used to specify a title for each axis. Titles are rendered using `titleFont` and in a size specified by `titleFontCubeSize`.

2.3 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.

2.3.1 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 determine axis bounds automatically, based on the data bounds.

If the `minIsDefault` property of a `JCAxis` object is `true`, every time the `JClass Chart 3D` data is changed, `JClass Chart 3D` will determine a reasonable default value for the axis minimum. If `false`, `JClass Chart 3D` will use the provided axis minimum.

Please note that there is a restriction on the `min` and `max` properties of the Z-axis; these are not allowed inside the Z-range of the data.

A side effect of setting any property that has a corresponding `UseDefault` property is that the `UseDefault` property will be set to `false`.

The following code will freeze the value of `minIsDefault` at its current value. It will also have the side effect of setting `minIsDefault` to `false`.

```
JCAxis yAxis = c.getChart3dArea().getYAxis();
yAxis.setMin(yAxis.getMin());
```

The following code will revert back to the default behavior, enabling `JClass Chart 3D` to calculate a default value for `minIsDefault` whenever it draws the graph.

```
yAxis.setMinIsDefault(true);
```

2.4 Legends

A legend itemizes the visual attributes used to identify data in the chart. You can customize the labels in the legend and the position of the legend. The legend is a `JComponent`, and all properties apply.

The `getLegend()` method of `JCChart3d` returns an instance of the abstract class `JCLegend` so that any subclass may be used as a legend. To utilize some of the 3D-specific properties described below, such as `layoutStyle` and `distributionRange`, you will need to cast the class to `JCChart3dLegend`. For example:

```
JCChart3dLegend legend = (JCChart3dLegend) chart3d.getLegend();
```

2.4.1 Legend Display

JClass Chart 3D will generate a legend only when the legend's `Visible` property is set to `true`. By default, there is no legend displayed.

If contours are drawn and zones are not, a legend listing the contour lines is generated. If zones are drawn, the legend lists the fill colors for each level. If points are drawn, the legend lists the symbols used for each series of points.

By default, JClass Chart 3D will attempt to list the legend contents vertically and position the legend to the right (that is, east) of the graph area.

2.4.2 Legend Text, Orientation, and Positioning

Legend Text

How the legend displays the text depends on the type of chart and data being plotted. In most cases, the text is based on contour levels.

If a scatter chart is plotting point data, the text is based on the `label` property of each series in the data view. If a scatter chart is plotting grid data with both zones and contours turned off, the text is based on the `name` property of the grid data object.

Legend Orientation

Use the legend `Orientation` property to lay out the legend horizontally or vertically.

Legend Positioning

The `JCChart3dLegend` class positions items in a grid wherein every row has the same height and every column the same width. `JCChart3dLegend` is a subclass of the abstract class `JCLegend`, and all of `JCLegend`'s properties are inherited by `JCChart3dLegend`.

Use the legend `Anchor` property to specify where to position the legend relative to the `chart3dArea`. You can select from eight compass points around the `chart3dArea`.

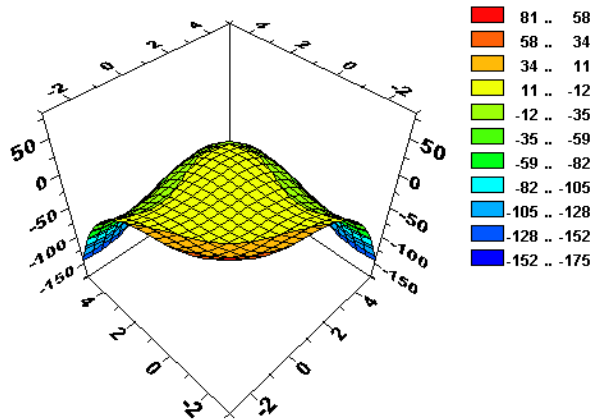


Figure 6 Vertically oriented legend anchored NorthEast.

2.4.3 Scatter Plot Charts

Scatter Plots of Point Data

For scatter plots of point data, the legend labels are derived from each `Chart3dPointSeries` `label` property (found in the `Chart3dPointData` object). Each label's corresponding symbol is determined from the `SymbolStyle` located in the series' `ChartStyle` property.

Scatter Plots of Grid Data

For scatter plots of grid data, the legend style depends on whether the contouring information has been computed (that is, if either of the `contoured` and `zoned` properties of the `dataView`'s `contour` object are `true`, the contours are computed). If contouring information has been computed, the legend entries are the same as they would be for surface plots and bar charts (see Section 2.4.4, [Surface and Bar Charts](#)). For each contour level there is also a symbol that is plotted (retrieved from the `SymbolStyle` property of the contour style for this level).

If the contours are not computed, the `LineStyle` and `SymbolStyle` from the `Chart3dGridData` object's `ChartStyle` property are used to draw a single legend entry. The label for this entry is derived from the `Name` property of the `Chart3dGridData` object.

2.4.4 Surface and Bar Charts

For surface and bar charts, the legend labels are determined by the value of the contours being used by the chart, while the color swatch is determined by each contour's

contourStyle object. There are also variations, depending on selected properties (detailed below) of the chart and legend.

When applicable, the LayoutStyle property of the legend determines the type of layout that the legend will use. Please see later in this section for more information.

Possible values for the LayoutStyle property are CONTINUOUS and STEPPED. CONTINUOUS is the default and displays the contour style fill colors as a continuous range, with contour level values placed at the transition points. STEPPED breaks down this continuous range into a series of smaller ranges defined by each contour level color.

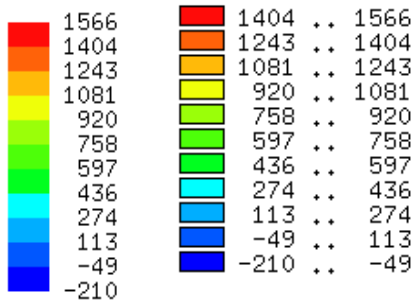


Figure 7 Continuous (left) and Stepped (right) Legends.

For example, an item in a STEPPED legend will show a box of color followed by a String of the form “min .. max”. The min and max values are determined by the values of the contour levels that bound this range. The color is determined by the fill color of the contour style of the top contour level in the range. In most cases, the existing contour ranges will be extended to the min and max of the data.

This LayoutStyle property is ignored when other properties interfere with the way things can be laid out. For example, if zones are not being drawn (isZoned()=false on the JCContour or JCProjection objects), the legend will be drawn as a series of items, each representing one contour line. The color of the line in the legend is derived from the LineStyle of the JCContourStyle of the contour being referenced.

If shading is not being done (isShaded()=false on the JCElevation object), then the items in the legend will appear as a series of lines, each representing a different contour zone. The color is determined from the FillStyle of the JCContourStyle of the contour zone.

The DistributionRange property determines the range of contour levels to be displayed in the legend. If using the default contours, the DistributionRange property has no effect. If the user has specified contour levels, a value of RANGE_DATA (the default) will restrict the contour levels in the legend to those in the range specified by the data min and max. A value of RANGE_ALL will cause all user-set contour levels to display.

There is also a modest number of small properties of `JCChart3dLegend` that the user can set to change the layout of legend items:

- `MarginGap` – the distance between the edge of the legend and the first legend item.
- `GroupGap` – the distance between groups of legend items; for instance, columns.
- `HorizontalItemGap` – the distance between legend items in a horizontal legend.
- `VerticalItemGap` – the distance between legend items in a vertical legend.
- `InsideItemGap` – the distance between the symbol and text portion of the legend item.

2.4.5 Overriding Labels

There are two ways a user may override the default labels generated for the legend:

In the first method, the `Labels` property takes a `List` object that contains a `List` object for each data view. (Currently, `JClass Chart 3D` supports only one data view.) Each internal `List` contains a series of `String` or `JCMultiFieldString` objects that are to be used to override the default legend labels.

The labels are to be placed in sequence, from bottom to top legend item. For example, the first label in the `List` will be used to represent the bottom item in the legend. If the user does not give enough labels to cover those needed, `JCChart3dLegend` will use default labels for the remainder of the unspecified labels.

In the second method, the `LabelGenerator` property takes any class that has implemented the `JCChart3dLegendLabelGenerator` interface. This interface has one method:

```
Object generateLegendLabel(JCChart3d chart3d, int level,  
                           double zmin, double zmax, Object label)
```

where `chart3d` is the `JCChart3d` instance to which this legend is attached, `level` is the contour level this label represents, `zmin` is the minimum value of the contour level range, `zmax` is the maximum value of the contour level range, and `label` is the value of the legend item's label so far (that is, either the default generated label or the label specified by the user with the `Labels` property). Please note that in some cases, `zmin` and `zmax` will be equal.

Any `String` or `JCMultiFieldString` returned by the implementor of this method will be used to label the legend item for the specified contour level. If null is returned, no legend item will be created for the specified contour level.

2.4.6 JCMultiFieldString

The `JCMultiFieldString` class handles multifield Strings, where each field may have a different alignment of text within its field. This class is used by the `JCChart3dLegend` class to manage specialized legend text.

JCMultiFieldString encapsulates a String that has multiple String fields, each with a potentially different alignment. Here is an example of how it is created:

```
new JCMultiFieldString("\rRight Text\cCenter Text\lLeft Text");
```

where `\r` represents right alignment of the field, `\c` represents center alignment of the field, and `\l` represents left alignment of the field. Each alignment character marks the beginning of a new field.

The JCMultiFieldString class converts this encoded String into an internal representation that can be drawn by JCCChart3dLegend such that all fields are aligned as indicated. Any number of fields may be present.

This object may be included in either the Labels property or the Object that is returned by the method implementing the JCCChart3dLegendLabelGenerator interface.

2.4.7 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.

JCCChart3dLegendManager is the class used by JClass Chart 3D to implement both the JCLegendPopulator and JCLegendRenderer interfaces, and to provide a built-in mechanism for itemizing range objects in a legend.

Custom Legends – Layout

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 JCCChart3d STEPPED legend, the text portion is the range between two contour levels, while the symbol portion is the box of color used to represent such values 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
Point pos;	Position of this legend item within the legend.
Point symbolPos;	Position of the symbol within the legend item.
Point textPos;	Position of the text portion within the legend item.
Dimension dim;	Full size of the legend item.
Dimension symbolDim;	Size of the symbol; provided by legend populator.
Dimension textDim;	Size of the text portion; provided by legend populator.
Rectangle pickRectangle;	The rectangle to use for pick operations; optional.
int drawType;	Determines drawing type; one of JCLegend.NONE, JCLegend.BOX, JCLegend.IMAGE, JCLegend.IMAGE_OUTLINED, JCLegend.CUSTOM_SYMBOL, JCLegend.BOX_PLAIN, JCLegend.LINE, or JCLegend.CUSTOM_ALL.
Object itemInfo;	Data related to this legend item; in JCCart3d, this is an instance of the LegendEntry class, which contains information on the data represented by the legend item and the style objects used to draw it.
Object symbol;	The symbol if other than the default type; usually null (means drawLegendItem decides).
Object contents;	The text portion; in JCCart3d, this is either a String or a JCMultiFieldString.

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 data 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. It is overridable by users who wish to use a different symbol size. One possible implementation is to use a symbol size identical to that which appears on the actual chart.

The easiest way to change default legend behavior without implementing all of the above is to subclass the existing `JCChart3dLegend` class. It implements all the above methods to lay out legend items in a straightforward grid.

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 3D`, 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`.

Custom Legends – Rendering

`JCLegendRenderer` is an interface that can be implemented by any user desiring to custom render legend items. This interface consists of five 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)
public void drawLegendItemText (Graphics gc, Font useFont,
    JCLegendItem this Item);
```

`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`, `JCLegend.BOX_PLAIN`, or `JCLegend.LINE`.

`drawLegendItemText()` provides a way for the user to define drawing for custom text objects. When the legend does not recognize the object in the contents field of a `JCLegendItem` (that is, it is not a `String`), this method will be called.

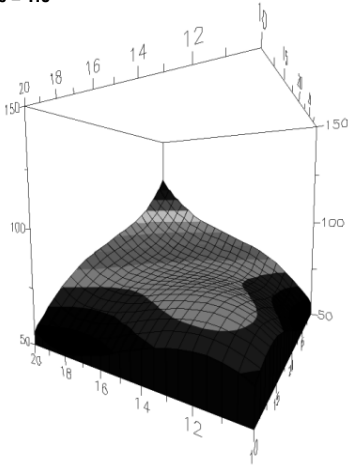
If implemented, the legend should be notified of the new renderer with the `setLegendRenderer()` method of `JCChart3dLegend`.

2.5 Perspective

The `JCView3d` class' `Perspective` property controls the perspective effect observed by projecting the plot cube onto the screen (the default value is 2.5). Small values exaggerate

the perspective effect, while large values diminish it. Valid values are between 1 and JCVIEW3D.MAX_PERSPECTIVE which is 500.

Perspective = 1.5



Perspective = 10

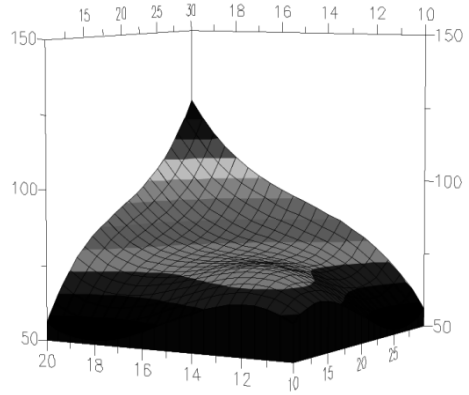


Figure 8 Perspective Depth Measurement.

2.6 Axis Scaling

The `JCPlotCube` class can be used to adjust the X, Y and Z dimensions of the unit cube relative to one another for Surface, Bar, or Scatter Plot charts. For example, if you would like the 3D display to be twice as long in the Y direction as in the X, set the Y scale to twice the X-scale. The default value of each `Scale` property is 1.0.

Here is a code snippet showing how to produce the right most chart of Figure 9:

```
JCPlotCube plotcube=c.getChart3dArea().getPlotCube();  
plotcube.setXScale(2.0);  
plotcube.setZScale(0.5);
```

Setting the Z scale higher or lower has the effect of flattening or stretching the surface view.

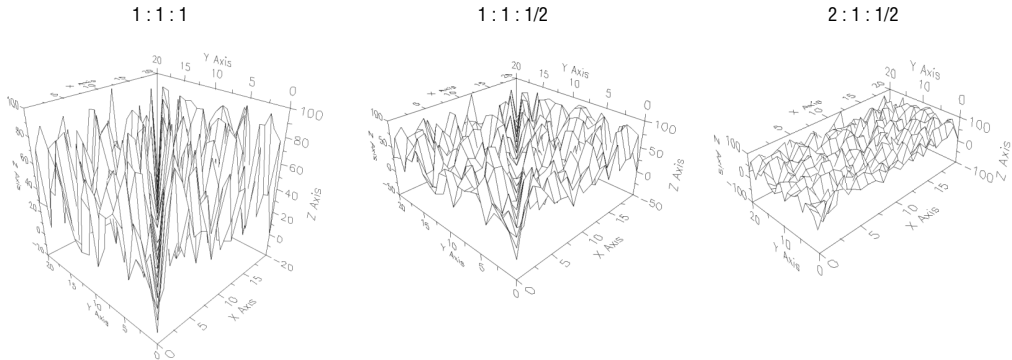


Figure 9 Axis scaling using various X: Y: Z ratios.

2.7 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 grid point in the chart (X- and Y-axis for grid data only); you can provide a label for specific values along the axis.

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.

2.7.1 Choosing 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:

`JCAxis.ANNOTATION_VALUES`
(default)

The chart chooses appropriate axis annotation automatically (with possible callbacks to a label generator), based on the data.

`JCAxis.ANNOTATION_DATA_LABELS`

The chart spaces the points based on the X- and Y-grid values and annotates them with text you specify (in the grid data source) for each point.

JCAxis.ANNOTATION_VALUE_LABELS

The chart annotates the axis with text you define for specific X-, Y-, or Z-axis coordinates.

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

2.7.2 Values Method

When the `annotationMethod` property is set to `ANNOTATION_VALUES`, JClass Chart 3D will automatically annotate the axis based on the range of data. This is the default annotation method. It is most suitable for the Z-axis, and for the X- and Y-axes when the chart type is surface.

2.7.3 Data Labels Method

If grid data is being plotted, individual lines in a surface, or a row/column of bars, can be labelled using the `ANNOTATION_DATA_LABELS` method.

This annotation method uses an array of Strings supplied as `xLabels` or `yLabels` through the `LabelledChart3dGridDataModel` interface to annotate each line from the grid. If a String in the list is null, a label is not drawn for that grid value, but the corresponding tick is drawn. The `xLabels` and `yLabels` array can also be set directly through methods provided in the `Chart3dGridData` class. Labels can be set as a list of Strings or as an `ArrayList` of objects of type `String`, `JCValueLabel`, or any other object whose `toString()` value is meant to be used as the label.

Data Label Clustering

Consecutive data labels can be combined into one label (in other words, when data labels are aggregated into a label cluster where only one label is displayed). In order for this to happen, ensure that:

- the annotation method is set to `JCAxis.ANNOTATION_DATA_LABELS`
- the `combineLabels` property of the corresponding `JCAxis` object is set to `true`
- the list of data labels contains one or more sub-lists of identical labels.

If data labels are specified via a list of `String` objects, and `m` consecutive labels are identical (they must point to the exact same `String`), they form a cluster of `m` labels, from which one is used to draw the label. If `m` is odd, the middle label of the cluster is used as the label. If `m` is even, a new label is positioned at the average value of the labels. The tick marks and gridlines in the cluster range are still drawn as if there was no clustering.

If data labels are specified via a list of `JCValueLabel` objects, the `label` property of the `valueLabels` in the same cluster must point to the same `String` object. In the same way that data labels are handled with a list of `String` objects, if consecutive `valueLabels` do not contain to the same `label` object, they will not belong to the same cluster (even if the `equals` method would return `true`). This allows the user to have two consecutive identical Strings without forcing them to be in the same cluster.

2.7.4 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`.

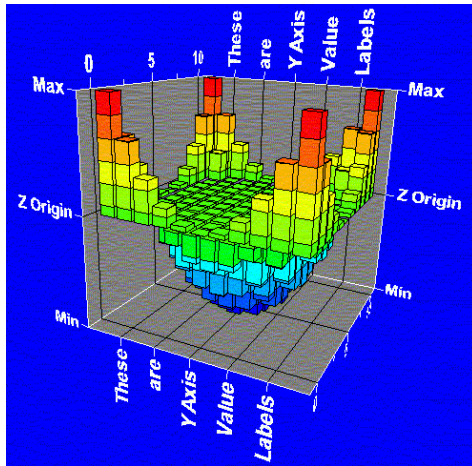


Figure 10 Using `ValueLabels` to annotate axes.

Every label displayed on the axis is one `ValueLabel`. Each `ValueLabel` has a `Value` property and a `Label` property. Additionally, the `tickDrawn`, `labelDrawn`, and `minorTick` properties control the display of the label and tick of the value label. The background and foreground of each label can also be set. These are null by default (in other words, no background is drawn, and the foreground color of the plot cube is used).

Please note that background colors for labels is currently not available in the Java 3D API version of `JClass Chart 3D`.

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

```
JCAxis x=c.getChart3dArea().getXAxis();
x.setValueLabels(0, new JCValueLabel(10, "Label 1"));
x.setValueLabels(1, new JCValueLabel(20, "Label 2"));
x.setValueLabels(2, new JCValueLabel(30, "Label 3"));
```

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

```
JCValueLabel v1
// this retrieves the label for the second Value-label
```

```
v1=c.getChart3dArea().getXAxis().
    getValueLabels(1);
// this retrieves the label at chart coordinate 2.0
v1=c.getChart3dArea().getXAxis().
    getValueLabels(2.0);
```

2.7.5 Custom Axes Labels

JClass Chart 3D 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.

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

The `makeLabel()` method is called for each label and tick that gets generated along the axis.

The `makeLabel()` method takes one parameter: `vLabel` (the internal value label that will be used to annotate the value). The user can then modify the `label` and the other properties of the value label.

- The `tickDrawn` property controls whether a tick mark is drawn for this value label.
- The `labelDrawn` property controls whether a label is drawn for this value label.
- The `minorTick` property determines whether a major or minor tick is drawn for this value label. If set to `true`, a minor tick is drawn. If set to `false` (the default value), a major tick is drawn instead. Please note that nothing is drawn if the `tickDrawn` property has been set to `false`.
- The foreground and background colors can be set for this value label.

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 10.

```
class MyLabelGenerator implements JCLabelGenerator
{
    public void makeLabel(JCValueLabel vLabel) {
        int intvalue = (int) vLabel.getValue();
        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 = "";
                vLabel.setTickOnly(true);
                return;
        }
        vLabel.setLabel(s);
        vLabel.setTickOnly(false);
    }
}
```

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

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

Also note that JClass Chart 3D calls the `makeLabel()` method for each *needed* label and tick. Thus, if JClass Chart 3D needs n labels and ticks, the `makeLabel()` method is called n times.

2.7.6 Rotating Axis Labels

The `annotationRotation` property allows you to specify the rotation of a chart's axis labels. The default value is `ANNOTATION_ROTATION_DEFAULT`. When explicitly setting the annotation rotation for an axis, please note that the number of labels may not change from the default case, even if more will fit on the axis.

Note: If the axis annotation rotation is explicitly set to horizontal or vertical, there may be overlapping labels.

<code>JCAxis.ANNOTATION_ROTATION_HORIZONTAL</code>	The given axis annotation will be horizontal (parallel to the axis).
<code>JCAxis.ANNOTATION_ROTATION_VERTICAL</code>	The given axis annotation will be vertical (orthogonal to the axis).
<code>JCAxis.ANNOTATION_ROTATION_DEFAULT</code>	JClass Chart 3D determines the axis annotation orientation. If the chart is a projection, the annotation orientation for the X-axis is horizontal and the annotation orientation for the Y-axis is vertical. If the chart is not a projection, the annotation method plays a roll in determining the annotation orientation. If the annotation method is <code>Values</code> , the axis annotation will be horizontal. If the annotation method is <code>ValueLabels</code> or <code>DataLabels</code> , the orientation is horizontal when the maximum label length is 1; otherwise, the orientation is vertical.

2.7.7 Label Selection and Clustering

Users can select labels through the normal “pick” mechanism of JClass Chart 3D. If the user instantiates a pick listener (this is part of the standard user actions; please see [Mapping and Picking](#), in Chapter 7), the chart will return a `JCData3dLabelIndex` object if a label has been selected. The `JCData3dLabelIndex` has `labelIndex` and `valueLabel` properties which respectively store the selected label's index and the internal value label corresponding to the selected label. Note that in the event of label clustering the `labelIndex` will be the index of the first label in the cluster.

Please note that this is currently not available in the Java 3D API version of JClass Chart 3D.

2.8 Gridlines

Gridlines can be displayed on each of the three primary planes – the XY plane, the XZ plane, and the YZ plane – using the properties of the `JCGridLines` class. This class specifies which gridlines are drawn for a given axis, via a plane mask. Planes are selected by “or”ing in the appropriate plane constants.

These constants of the `JCGridLines` class specify the possible planes:

Constant	Value	Description
<code>XY_PLANE</code>	1	The XY plane.
<code>XZ_PLANE</code>	2	The XZ plane.
<code>YZ_PLANE</code>	4	The YZ plane.

For instance, the X-axis controls gridlines that are perpendicular to the X-axis in the XY and the XZ plane.

To draw X-gridlines in all applicable planes, set `JCGridLines` to the value `(XY_PLANE | XZ_PLANE)`. To remove gridlines completely, set the value to 0.

As another example, here is a code sample showing how to set a plane mask:

```
JCGridLines gl=c.getChart3dArea().getZAxis().getGridLines();
gl.setPlaneMask(XZ_PLANE|YZ_PLANE);
```

Gridlines are drawn where annotation is drawn on the axis, regardless of the annotation method.

2.8.1 Gridline Styles

The gridline’s style – color, pattern, and width – are controlled by the `JCLineStyle` class. A `JCGridLines` object initializes its `lineStyle` so that by default, a solid black line of width 1 is drawn. The properties of the initial default `lineStyle` can be changed or a new `lineStyle` can be set on the `JCGridLines` object.

Here’s an example showing the code to set the Y-axis gridlines to blue, have a `dash_dot` pattern, and be 4 pixels in width:

```
JCGridLines gl=c.getChart3dArea().getYAxis().getGridLines();
gl.setLineStyle(new JLineStyle(4,Color.blue,JLineStyle.DASH_DOT));
```

Color

To set the color of the gridlines, choose the AWT color class representing the color to be used to draw the lines. If null, the current color of the `Graphics` object is used. The following table summarizes the built-in color names defined in `java.awt.Color`:

Built-in Colors in <code>java.awt.Color</code>		
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).

Take care not to choose a gridline color that is also used to display data in the chart. The default `JLineStyle` 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 3D` 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.

Pattern

To set the pattern, use the `Pattern` property, which dictates the pattern used to draw a line. Choices for gridline pattern are:

- none
- solid
- long_dash
- short_dash
- lsl_dash
- dash_dot

You can also create a custom user-defined pattern for the gridlines. Here is a method declaration showing how to do this:

```
public void setPattern(float[] patternArray, float[]  
    legendPatternArray)
```

where `patternArray` is an array of floats representing the pattern to use when drawing this line, and `legendPatternArray` is an array of floats representing the pattern to use

when drawing this line in the legend. A note of caution: if the values get too small, nothing will be drawn.

Width

The width of the gridlines can be set using the `setWidth()` method. A positive integer is required.

2.9 Header and Footer Titles

JClass Chart 3D can have two titles, called the header and footer. By default they are `JLabel` instances and behave accordingly (a `JLabel` class is a Swing class). A `JLabel` object can display text, an image, or both.

You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are left-aligned, by default. Image-only labels are horizontally centered by default.

A title consists of one or more lines of text with an optional border, both of which you can customize. You can also set the text alignment, positioning, colors, and font used for the header or footer.

See “[How to Use Labels](http://java.sun.com/docs/books/tutorial/uiswing/components/label.html)” in the Java Tutorial (<http://java.sun.com/docs/books/tutorial/uiswing/components/label.html>) for further information.

2.10 Adding Header, Footer, and Labels

JClass Chart 3D will always try to produce a reasonable chart display, even if very few properties have been specified. JClass Chart 3D 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, the properties of the Header and Footer objects contained in the chart need to be set. For example, the following code sets the `Text` and `Visible` properties for the footer:

```
// Make footer visible
chart3d.getFooter().setVisible(true);
// By default, footer is a JLabel - set its Text property
((JLabel)chart3d.getFooter()).setText("1963 Quarterly Results");
```

The `Visible` property controls whether the header or footer is displayed. `Text` specifies the text displayed in the header or footer.

By default, the header and footer are instances of `JLabel`.

Programming JClass Chart 3D: Surfaces and Bars

Fifteen Basic Types of Surfaces and Bars ■ *Chart Types* ■ *Bar Charts and Histograms*
Contours and Zone Display ■ *Mesh Controls* ■ *Surface Colors* ■ *Solid Surface*

All elements mentioned in this chapter refer to both the Java 2 API and the Java 3D API, unless specifically noted.

3.1 Fifteen Basic Types of Surfaces and Bars

One of the types of data that can be attached to JClass Chart 3D is grid data. Grid data that has been attached to the chart can be displayed in a surface, bar, or scatter plot representation. Grid data is supplied to the chart via `Chart3dDataView`'s `elevationDataSource` property. The chart then processes the data and stores it in an internal data object of type `Chart3dGridData`. For elevation data, this internal object can be referenced via the `elevationData` property. The user can retrieve this internal object, query it for data values, and set certain properties on it.

The chart uses elevation data to draw a surface plot, bar plot, or scatter plot, depending on the `chartType`, which will be `SURFACE`, `BAR`, or `SCATTER`.

An `elevationDataSource` must implement the `Chart3dDataModel` interface. To be grid data, it must also implement the `Chart3dGridDataModel` interface. This promises the chart that it can supply an `xGrid` array, a `yGrid` array, and a doubly indexed `zValues` array. If the `datasource` wants to provide X- and Y-data labels to the chart, it should implement the `LabelledChart3dGridDataModel` interface. Similarly, a hole value, other than the default, can be provided through the `HoleValueChart3dDataModel`. Data values can be edited through the `EditableChart3dDataModel` and data changes can be monitored if the `datasource` implements the `Chart3dDataManager` interface.

The `elevationDataSource` can also take points as its data type. This type of data can only be used for scatter plots.

Note that using a data set of type `point` with a chart type of `SURFACE` or `BAR` will produce a blank plot.

When grid data is used and the chart type is either `SURFACE` or `BAR`, `JClass Chart 3D`'s four basic display Boolean properties – `Meshed`, `Shaded`, `Contoured`, and `Zoned` – combine to create 15 different basic surface and bar displays. No graph is displayed when all four Booleans are `false`.

3.1.1 **JCElevation: Meshed, Shaded, and Transparent Plots**

The `JCElevation` object determines whether surface and bar plots are `Meshed`, `Shaded`, or `Transparent`. The `JCElevation` constructor initializes the `Meshed`, `Shaded`, and `Transparent` properties (each of which takes a Boolean value). The defaults for each of these are `Meshed=true`, `Shaded=false`, and `Transparent=false`.

Meshed

Surfaces: When `Meshed` is `true`, `JClass Chart 3D` displays the X-Y grid projected onto the 3D surface in a 3D view with a Z-axis. You can use the `xMeshShow` and `yMeshShow` properties of `JCSurface` to individually control whether the X- and Y-mesh lines are showing.

The `xMeshFilter` and `yMeshFilter` properties of `JCSurface` allow every *n*th mesh line to be drawn. By default, the chart chooses a pleasing filtering. The `meshTopColor` and `meshBottomColor` properties allow the user to control the top and bottom colors of the mesh lines.

Bars: When `Meshed` is `true`, `JClass Chart 3D` will draw the outline of all the bars. All bars with a value greater than or equal to the origin of the Z-axis will be outlined using the `meshTopColor`, and all bars with a value less than the origin will be outlined using the `meshBottomColor`. When `Transparent` is `true`, all the lines of every bar will be visible.

Shaded

Surfaces: When `Shaded` is `true`, `JClass Chart 3D` displays the data as a flat shaded surface in a 3D view with a Z-axis. The surface color is controlled with the `shadedTopColor` and the `shadedBottomColor` properties (not currently available in the Java 3D API version of `JClass Chart 3D`).

Bars: When `Shaded` is `true`, `JClass Chart 3D` draws each bar as a solid bar. All bars with a value greater than or equal to the origin of Z-axis will be drawn using the `shadedTopColor`, while all bars with a value less than the origin will be drawn using the `surfaceTopColor`.

The following code snippet shows setting these properties:

```
JCElevation elevation=c.getDataView(0).getElevation();
elevation.setMeshed(true);
elevation.setShaded(true);
elevation.setTransparent(false);
```

3.1.2 JContour: Contoured and Zoned Plots

The `JContour` class deals with information about contours and zones. Two of its properties are `Contoured` and `Zoned`. Both default to `false`.

Contoured

Surfaces: When `Contoured` is `true`, `JClass Chart 3D` examines the distribution of the data, using the `JContourLevels` class, and draws contour lines demarcating each of the contour levels. The contour line style, thickness, and color are controlled with the `contourStyles` property.

Bars: When `Contoured` is `true`, `JClass Chart 3D` examines the distribution of the data, using the `JContourLevels` class, and draws contour lines around the bars, demarcating each of the contour levels. The contour line style, thickness, and color are controlled with the `contourStyles` property.

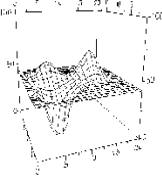
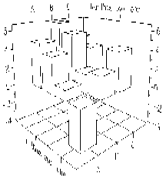
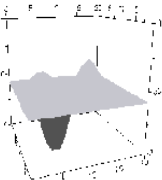
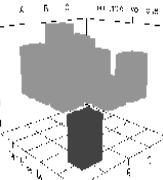
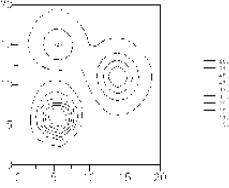
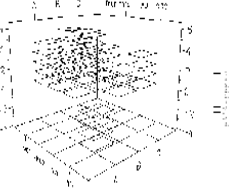
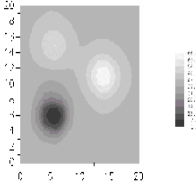
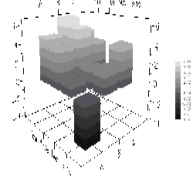
Zoned

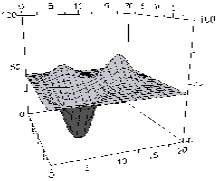
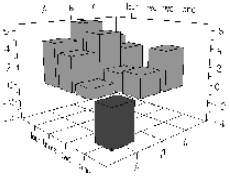
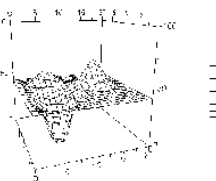
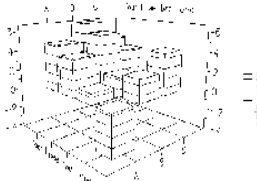
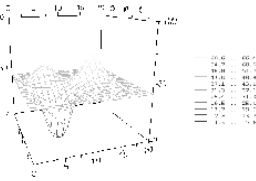
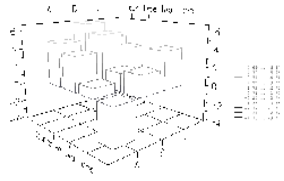
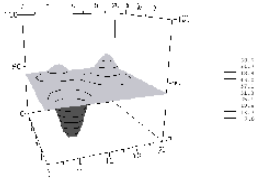
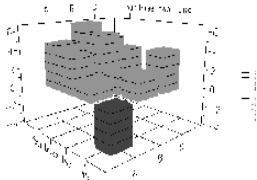
Surfaces: When `Zoned` is `true`, `JClass Chart 3D` examines the distribution of the data, using the `JContourLevels` class, and fills each level with a solid color. (Unless `Meshed` is `true` and `Shaded` is `false`, in which case the fill color is used to draw each level's mesh lines.) The color for each level is specified with the `contourStyles` property.

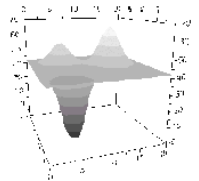
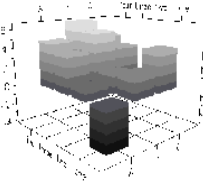
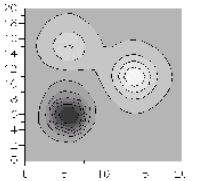
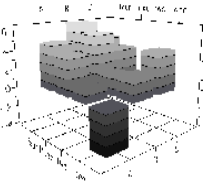
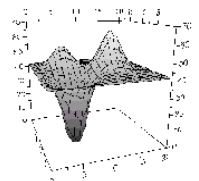
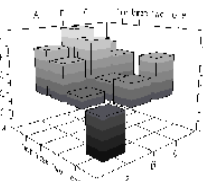
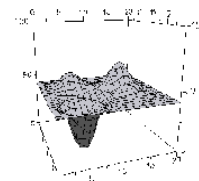
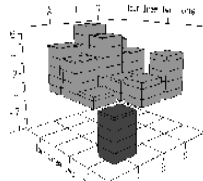
Bars: When `Zoned` is `true`, `JClass Chart 3D` examines the distribution of the data, using the `JContourLevels` class, and fills each level within each bar with a solid color. (Unless `Meshed` is `true` and `Shaded` is `false`, in which case the fill color is used to draw each level's mesh lines.) The color for each level is specified with the `contourStyles` property. If `zoneDataSource` is supplied to the data view, each bar is filled with a solid color. Otherwise, the bar is segmented by height according to the contour levels.

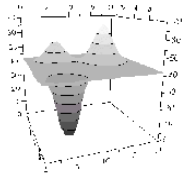
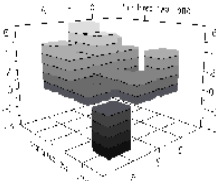
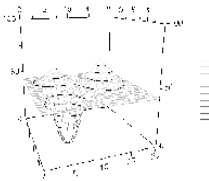
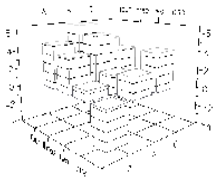
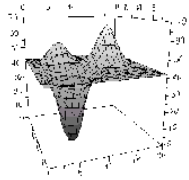
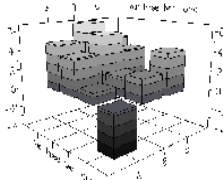
3.2 Chart Types

The following table shows the 15 basic graph types:

GraphChart Type	Meshed	Shaded	Contoured	Zoned	Surface Example	Bar Example	Comments
1	T	F	F	F			Meshed. Displays surface as a mesh and bars in outline.
2	F	T	F	F			Shaded. Displays surface and bars in a flat shade. Top and bottom colors may be set.
3	F	F	T	F			Contoured. Contour lines are automatically drawn between distribution levels in the data.
4	F	F	F	T			Zoned^a. Similar to Contoured, except that each distribution level is displayed in a solid color.

GraphChart Type	GraphChart Type				Surface Example	Bar Example	Comments
	Meshed	Shaded	Contoured	Zoned			
5	T	T	F	F			Meshed, Shaded. Draws surface as a mesh and bars in outline. Surface and bars are flat shaded.
6	T	F	T	F			Meshed, Contoured. Displays surface as a mesh and bars in outline. Also draws contour lines along borders between distribution levels in the data.
7	T	F	F	T			Meshed, Zoned. Displays surface as a mesh and bars in outline. Uses zoning colors for mesh and bar outlines.
8	F	T	T	F			Shaded, Contoured. Displays a flat-shaded surface or bars with contour lines superimposed.

GraphChart Type	Meshed	Shaded	Contoured	Zoned	Surface Example	Bar Example	Comments
9	F	T	F	T			Shaded, Zoned. Zone colors are used to flat shade the surface or bars.
10	F	F	T	T			Contoured, Zoned^a. Displays contour lines and flat shaded zone colors to demarcate levels in the data.
11	T	T	F	T			Meshed, Shaded, Zoned. Similar to Shaded, Zoned, but with a mesh or bar outlines superimposed.
12	T	T	T	F			Meshed, Shaded, Contoured. Similar to Meshed, Shaded, but contour lines are superimposed.

GraphChart Type					Surface Example	Bar Example	Comments
	Meshed	Shaded	Contoured	Zoned			
13	F	T	T	T			Shaded, Contoured, Zoned. Similar to Shaded, Zoned, but contour lines are superimposed.
14	T	F	T	T			Meshed, Contoured, Zoned. Similar to Meshed, Zoned, but with contours superimposed.
15	T	T	T	T			Meshed, Shaded, Contoured, Zoned. The sum of all basic options.

a. In this release, Zoned is the same as Zoned and Shaded for bar charts. Application developers are urged to use the Zoned and Shaded combination for this view, since the interpretation of the Zoned combination may change in a future release.

3.3 Bar Charts and Histograms

When `chartType` is `BAR`, the elevation data will be displayed as a bar chart (if grid data is supplied). Each data point will be represented by a single bar. The spacing between adjacent elements in the grid is honored.

Bar Z Origin

Bars start from the `z` Axis origin, which can be controlled through `JCAxis`'s `origin` property. The default value of the origin is `0.0`. When `Shaded` is `true`, bars that have values greater than the origin are rendered in the `shadedTopColor` property of the

JCElevation class. Negative bars (that is, bars with values less than the origin) are rendered in the shadedBottomColor.

Bar Spacing

The amount of space occupied by a bar, as a percentage of the maximum amount possible, is controlled through the xSpacing and ySpacing properties of the JCBAR class. The default is 80%. Setting it smaller results in thinner bars. Setting bar spacing to 100% (the maximum) results in bars that about one another.

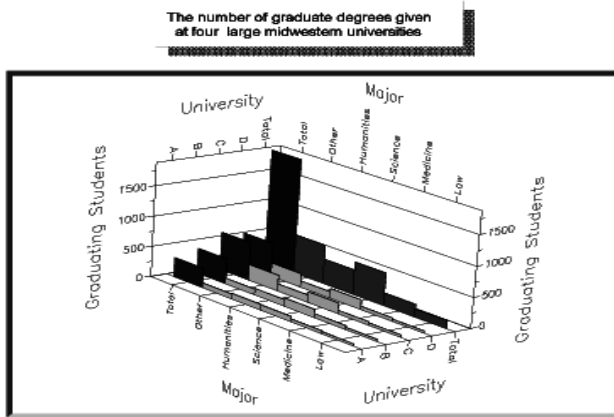


Figure 11 Fixed Bar Chart with X-Spacing set to 1% and Y Spacing set to 100%.

Histograms

To display a histogram, set both the xFormat and yFormat properties of JCBAR to HISTOGRAM (default is FIXED). Thus, the X-axis and Y-axis can be independently switched between fixed and histogram formats.

If the X-axis is switched to a histogram, each bar's left edge will be drawn aligned with corresponding X-values in the data. The width of each bar will be the distance between subsequent X-values in the data. Since the width of each bar is derived from the spacing between its neighbors, there is always one fewer bar along a histogram axis than there is if the same data is displayed along a fixed axis.

Histograms usually make good use of data whose grid values are not equally spaced. This gives control over spacing in the data grid.

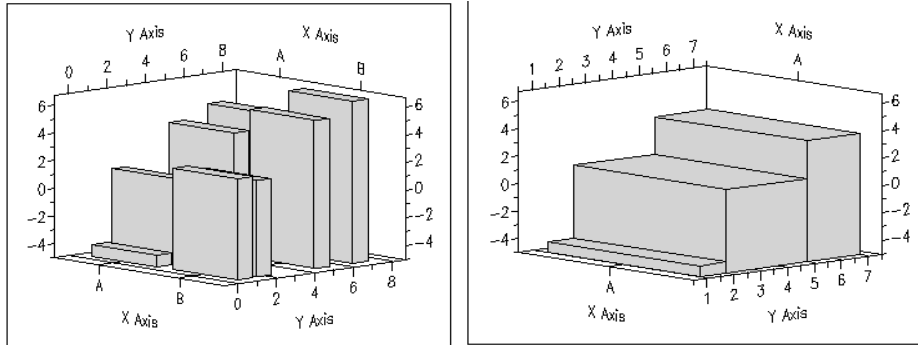


Figure 12 Fixed versus Histogram Display of Identical Data.

Grid Colors

If `Shaded` is true and `Zoned` is false, the colors of the individual bars can be controlled. Normally, all bars with a value below the origin are colored with `shadedBottomColor`, and all others are colored with `shadedTopColor`. However, in some situations, it is useful to color a row, column, or individual bar in the chart with a distinct color.

The `JCGridColumn` class is used to specify the fill color of a group of bars, or of an individual bar. The `Chart3dDataView` class' `gridColors` property stores a list of `JCGridColumn` objects.

The `JCGridColumn` class has a `dataIndex` property and a `color` property. The `dataIndex` is a reference to a `JCData3dGridIndex` object which stores an index to a bar or group of bars. The `color` property indicates what color to make the bars referenced by the `dataIndex`.

An individual bar can be colored by specifying its indices in the `dataIndex` object. For example, to set the bar in the third X-data line, second point to red, set the `xIndex` property of the `dataIndex` to 2, the `yIndex` to 1, and the color to `Color.red`.

To specify the color of an entire row of bars, set the other index to `JCData3dIndex.ALL`. For instance, the entire fifth X line of bars is set to green by setting the `xIndex` property of the `dataIndex` to 4, the `yIndex` to `JCData3dIndex.ALL`, and the color to `Color.green`. To set the color of all the bars, set both the `xIndex` and `yIndex` of the `dataIndex` to `JCData3dIndex.ALL`.

`Chart3dDataView` has several convenience methods for manipulating the list of `JCGridColumn`s. It has `addGridColumn()` and `removeGridColumn()` for adding and removing grid colors. It also has a `findGridColumn()` method which retrieves the first grid color that matches a given (`xIndex`, `yIndex`) index starting from the end of the list; this is what the

chart uses to determine the color of a given bar. If `findGridColor()` returns null, the bar color falls back to either the `shadedTopColor` or the `ShadedBottomColor`.

JClass Chart 3D's `Chart3dDataView` will only maintain one entry per $(xIndex, yIndex)$ combination. Whenever a second entry for the same indices is supplied, the first entry is removed. Later entries take precedence over earlier entries.

The following code created the bar coloring in Figure 13:

```
dataView.addGridColor(JCData3dIndex.ALL, JCData3dIndex.ALL,
    new Color(255, 208, 0));
dataView.addGridColor(1, JCData3dIndex.ALL, Color.blue);
dataView.addGridColor(JCData3dIndex.ALL, 1, new Color(176, 32, 240));
dataView.addGridColor(1, 1, Color.red);
```

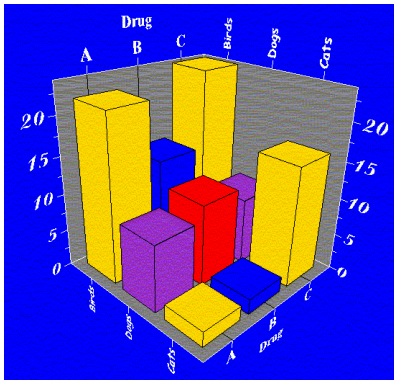


Figure 13 3D Chart demonstrating numerous grid colors.

Note that even though the index of the bar at $(1, 1)$ matches every entry in the list, the bar is colored red because it matches the last entry in the list.

3.4 Contours and Zone Display

When `Contoured` or `Zoned` is true, JClass Chart 3D marks each contour level from an array of 100 built-in contour styles.

Each contour style contains information about the contour line style, width, pattern, color, and zone color (used to mark each *level*). Contour styles can be customized by an application. Please see [Customizing Contour Styles](#), in Chapter 6 for details.

Contour Styles Used

JClass Chart 3D determines which contour style to use for a particular level automatically, evenly distributing the styles through the number of levels, as shown by Figure 14.

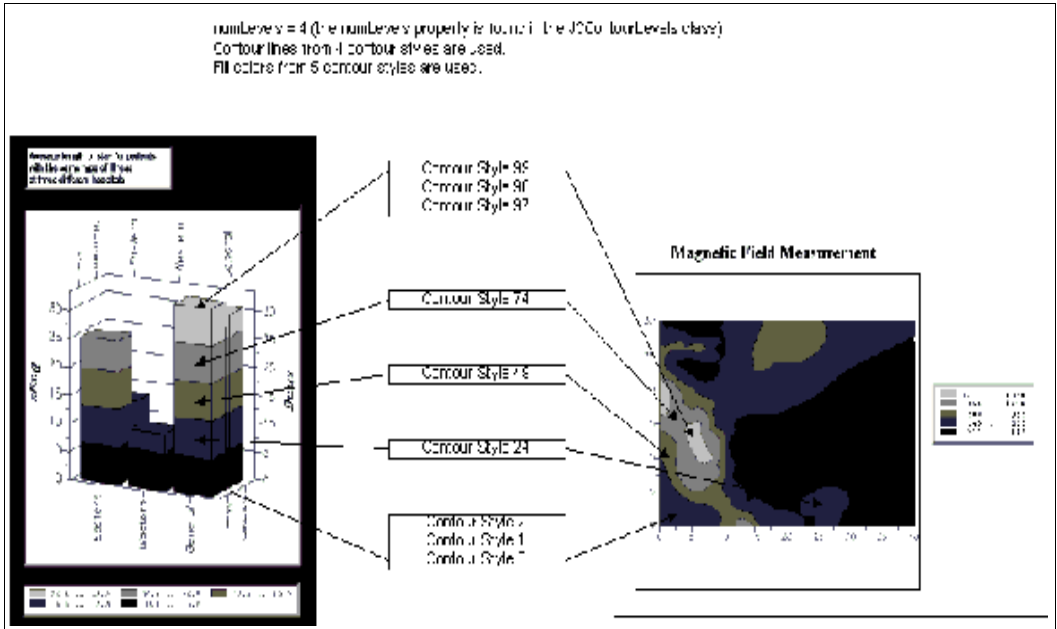


Figure 14 A Sampling of Supplied Contour Styles Used as Needed.

By implementing the `JCContourMapping` interface and using the `setContour.Mapping()` method of the `JCContour` class, you can override the default level to contour style mapping.

Also, by using the `JCContourLevels` class, you can specify your own contour levels (such as an array of doubles). You can also use your own contour styles by setting the `contourStyles` property of the `JCContour` class. For further information, please see [Advanced JClass Chart 3D Programming](#), in Chapter 6.

Contour/Zone Projection

The `JCProjection` class specifies information about the Contoured and Zoned projections on the top and bottom of the plot cube. For instance, you can use the `setContoured` method to set the projection to be Contoured, and you can use the `setZoned` method to set the projection to be Zoned.

The `JCPlotCube` `floor` property is a `JCProjection` that indicates whether a projection should be drawn on the bottom of the plot cube. If either the Contoured or Zoned

property of the floor projection is `true`, a projection is drawn. Similarly, the ceiling projection controls what type of projection is drawn on the top of the plot cube.

These properties do not depend on the values of `Contoured` and `Zoned` of the `JCContour` class. However, any other property that affects contour generation or rendering affects projected contours/zones. Floor and ceiling projections are ignored in 2D graphs, bar charts, and scatter plots.

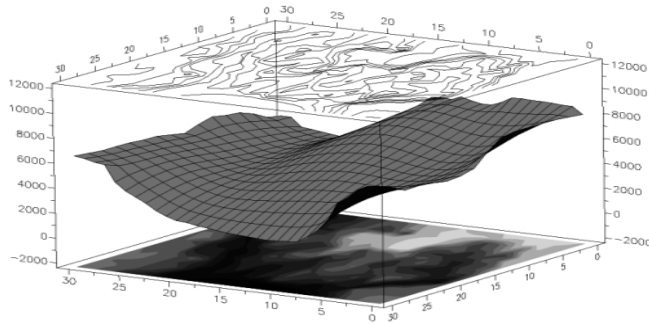


Figure 15 Projecting Contours and Zones.

Zone Method

An application can control the method used to fill each zone region by leveraging the `ZoneMethod` property of the `JCContour` class. By default, `JClass Chart 3D` fills between each contour interval (`ZONE_CONTOURS`). When set to `ZONE_CELLS_AVERAGE`, `JClass Chart 3D` fills entire cells in the grid based on the average of the four corners of the cell. Figure 16 illustrates the difference visually. When set to `ZONE_CELLS_CORNER`, entire cells are filled using the value at the bottom left corner of the cell.

Cell zoning produces a coarser-looking surface, but offers significant performance advantages over contour zoning. However, the visual difference between the two types of zoning diminishes with larger grids.

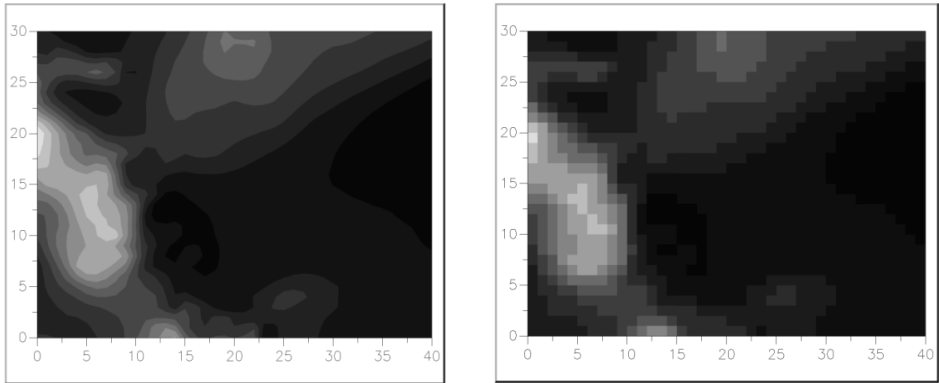


Figure 16 Contour Zoning (left) and Cell Average Zoning (right).

3.5 Mesh Controls

Mesh Colors

Mesh colors are controlled by methods in the `JCElevation` class.

The bottom and top colors of the mesh drawn when `Meshed` is true can be set with `setMeshTopColor` and `setMeshBottomColor`. They are both “black” by default. See [Chart Colors](#), in Chapter 1 for details on setting colors.

Mesh Filtering

Methods in the `JCSurface` class control mesh filtering. This class contains properties that pertain to surface plots.

The `xMeshFilter` and `yMeshFilter` properties specify how the mesh is filtered before being displayed. By default, no filtering is performed. When set to 0, `JClass Chart 3D` automatically filters the mesh to provide a pleasing display, and changes the filter as the graph is scaled or the data changes.

You can hard-code a mesh filter by setting these properties to any positive integer. For example, a value of 5 filters the mesh so that every 5th line is drawn.

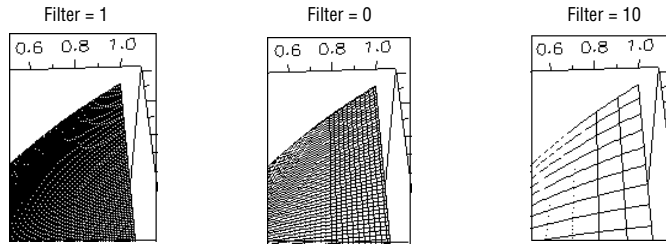


Figure 17 Effect of Mesh Filtering.

Hidden Mesh Lines

When `Meshed` is true and `Shaded` is false, grid and contour lines that are obscured from view by intervening portions of the scene are not displayed by default. To display these lines, set the `Transparent` property of `JCElevation` to true.

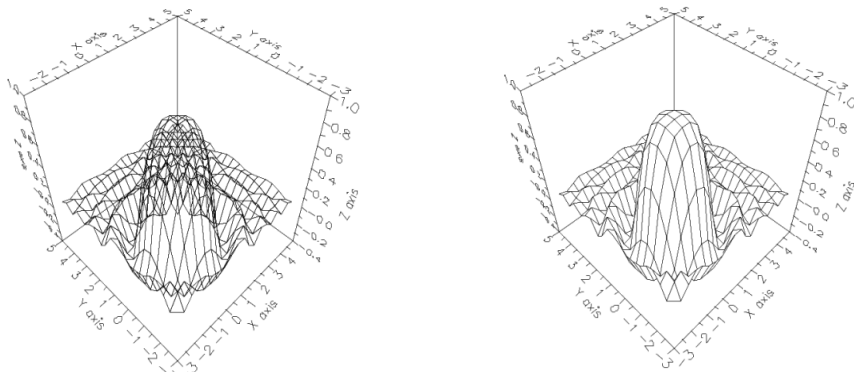


Figure 18 Hidden line removal.

3.6 Surface Colors

The bottom and top colors of the shaded surface drawn when `Shaded` is true can be set with `shadedTopColor` and `shadedBottomColor` of the `JCElevation` class. By default the bottom color is “dim grey” or `RGB(112,112,112)` and the top color is “light grey” or `RGB(211,211,211)`. See [Chart Colors](#), in Chapter 1 for details on setting colors. Please note that `shadedTopColor` and `shadedBottomColor` are currently not available for surfaces in the Java 3D API version of `JClass Chart 3D`.

3.7 Solid Surface

Setting the `Solid` property of `JCSurface` to `true` will cause `JClass Chart 3D` to draw a skirt around the data, thereby joining the edge of the surface to a plane at the minimum Z value, as shown in Figure 19.

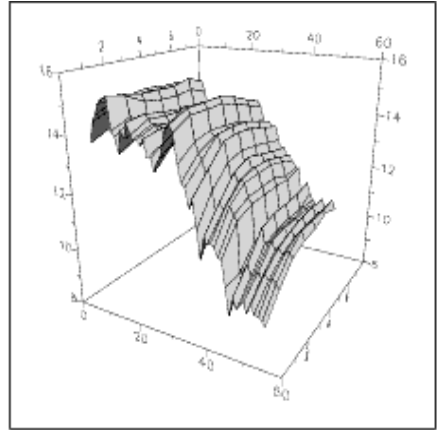
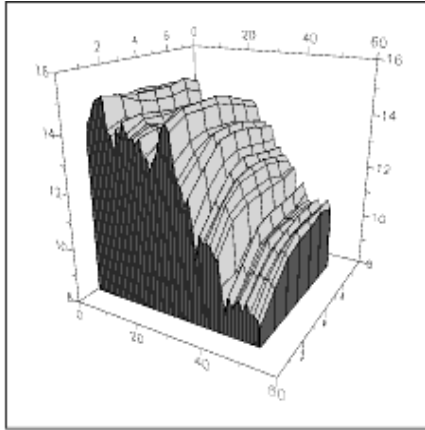


Figure 19 Setting Solid Surface On and Off.

Programming JClass Chart 3D: Scatter Plots

Overview ■ *Three Basic Types of Scatter Plots*
Controlling Symbol and Drop Line Style ■ *Chart Styles*

All elements mentioned in this chapter refer to both the Java 2 API and the Java 3D API, unless specifically noted.

4.1 Overview

As noted in the previous chapter, if grid data is provided to the chart via the `elevationDataSource` property of the `Chart3dDataView` class, then surface, bar, or scatter plots can be drawn. The previous chapter dealt with surface charts and bar charts; this chapter deals with scatter plots.

A scatter plot can be drawn using grid data, but the most usual way of providing data to a scatter plot is to provide the `elevationDataSource` with point data. This type of data supports an arbitrary number of series, each containing an arbitrary number of points.

An object can be a point data source if it implements the `Chart3dPointDataModel`. When a point data source is passed to a `Chart3dDataView` object, the data is extracted and stored in an internal point data object of type `Chart3DPointData`. This object can be retrieved via the `elevationData` property of `Chart3dDataView`.

More sophisticated point data sources can implement the `LabelledChart3dPointDataModel`, `EditableChart3dDataModel`, `HoleValueChart3dDataModel`, and the `Chart3dDataManager` interfaces.

Please see [Data Sources](#), in Chapter 5, for further details.

Each series can have a symbol that makes it distinct from that of the other series. When a chart of type `SCATTER` is used in conjunction with grid data, each grid point will define a point in the scatter plot.

4.2 Three Basic Types of Scatter Plots

JClass Chart 3D offers three basic types of scatter plots: 3D scatter plots, 3D scatter plots with drop lines, and 2D scatter plots. (3D scatter plots with drop lines are not currently available in the Java 3D API version of JClass Chart 3D.)

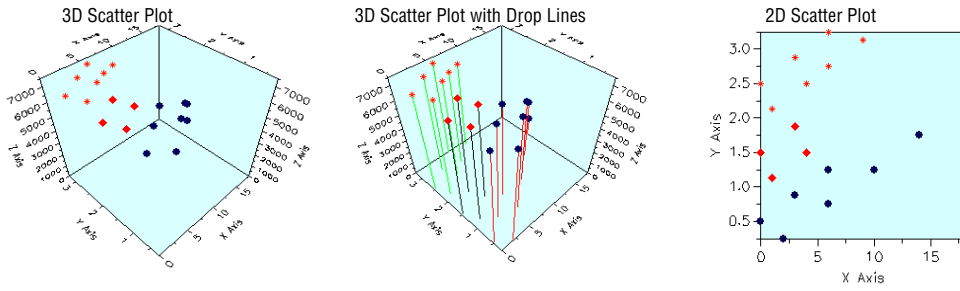


Figure 20 Scatter Plot examples.

Drop lines are lines drawn from each point on the 3D scatter plot down to the matching (x,y) position on the Z-axis origin of the plot cube. The drop line joins the point (x, y, z) to the matching point (x, y, origin) . Drop lines are not used for 2D scatter plots. To draw drop lines on 3D scatter plots, set the `dropLines` property of the `JCScatter` class to `true`.

2D scatter plots use only the X- and Y-values, and ignore the z components. They are “flat” charts. These plots are created by setting both the `meshed` and the `shaded` properties of the `JCElevation` object to `false`.

4.3 Controlling Symbol and Drop Line Style

4.3.1 Point Data

The style of symbols and drop lines used in scatter plots is determined on a per series basis. The `chartStyle` property of the `Chart3dPointSeries` specifies the line and symbol style information for a series.

A list of `Chart3dPointSeries` objects can be obtained from the `series` property of the internal point data object. This object, of type `Chart3dPointData`, can be retrieved via the data view’s `elevationData` property.

Here is a code sample that changes the line style of the third series of a scatter plot. This code sample assumes that a point data source has already been set on the data view.

```
Chart3dPointData pointData=(Chart3dPointData)dataView.getElevationData();
ArrayList series=pointData.getSeries();
Chart3dPointSeries pointSeries=(Chart3dPointSeries)series.get(2);
JCChartStyle chartStyle=pointSeries.getChartStyle();
chartStyle.setLineStyle(new JCLineStyle(2, color.blue, JCLineStyle.SOLID));
```

Line pattern, color, and width are controlled by the chart style's `LineStyle` property, and symbol color, size, and pattern by the chart style's `SymbolStyle` property.

4.3.2 Grid Data

If either the `zoned` or `contoured` properties of `JCContour` are true, the symbol color and style, along with the dropline style, will be chosen through the appropriate `contourStyle` for the contour level to which the point belongs.

Note: This makes use of the `JCContourStyle` class' `LineStyle` and `SymbolStyle` properties.

If both `zoned` and `contoured` are false, the symbol color and style, along with the dropline style, will be chosen through the `chartStyle` property if the internal `Chart3dGridData` object is associated with the grid data.

4.4 Chart Styles

In a scatter plot, how a data value looks when it is displayed (for instance, color, line pattern, symbol style, line thickness, and so forth) depends on the chart style that has been defined for that data value. For example, the values in the third series of data will be rendered on screen using the chart style associated with the third series of the point data.

4.4.1 Default Chart Styles

When a new `Chart3dPointSeries` object is created, a default `chartStyle` is created that possesses default line and symbol styles. The defaults are cycled, so consecutive calls will return different chart styles. A user may choose to override these defaults.

The `JCChartStyle` class has two main components: a `JCSymbolStyle` object that stores symbol information and a `JCLineStyle` object that stores line style information. For convenience, the user can either set or get the `symbolStyle` or `LineStyle` properties of the `JCChartStyle` class, or the user can set the individual `symbolStyle` or `LineStyle` properties directly on the `chartStyle`.

For example, the following line of code:

```
chartStyle.setLineWidth(2);
```

will change the line width of the `chartStyle`'s `LineStyle` object.

The `JCChart3dStyle` data structure contains all the information about how a set of data will be represented graphically. The properties are broken down as follows:

<code>symbol color</code>	The color used when drawing symbols.
<code>symbol size</code>	The size is the diameter in pixels of the scatter symbol. It must be greater than or equal to 1.
<code>symbol shape</code>	The shape of the symbol. For example, <code>JCSymbolStyle.DOT</code> . Available shapes are dot, box, triangle, diamond, star, vertical line, horizontal line, cross, circle, square, and rectangle for the Java 2 API version of JClass Chart 3D, and sphere, cube, cone, point, cylinder, and tetrahedron for the Java 3D API version of JClass Chart 3D.
<code>line pattern</code>	The line pattern used for drop lines. Available line patterns are none, solid, long dash, short dash, long-short-long dash, and dash-dot. Default is solid. Custom patterns are also possible.
<code>line color</code>	The color used when drawing drop lines.
<code>line width</code>	The line width used for drop lines. Must be greater than or equal to 1.

The following method will print out the symbol size being used for the series of data, and double it:

```
public void double SymbolSize (Chart3dDataView dataView, int seriesIndex)
{
    Chart3dPointData pointData=(Chart3dPointData)dataView.
        getElevationData();
    ArrayList series=pointData.getSeries();
    Chart3dPointSeries pointSeries=(Chart3dPointSeries)series
        (seriesIndex);
    JCChartStyle chartStyle=pointSeries.getChartStyle();
    system.out.println("symbol size:"+chartStyle.getSymbolSize());
    chartStyle.setSymbolStyle(2*chartStyle.getSymbolSize());
}
```

4.4.2 Grid Data

For grid data, the `JCSymbolStyle` object and the `JCLineStyle` object that determine how each point is drawn are obtained from either a `JCContourStyle` object or a `JCChartStyle` object. See [Grid Data](#), in Chapter 4, for more information.

5

Data Sources

Overview ■ *Pre-Built Chart DataSources* ■ *Loading Data from a File*
Loading Data from a Swing TableModel ■ *Loading Data from an XML Source*
Data Binding using JDBCDataSource ■ *JCData3dUtil class* ■ *Making Your Own Chart Data Source*
HoleValueChartDataModel – Specifying Hole Values
Making an Updating Chart Data Source ■ *Summary of JClass Chart 3D Data Interfaces*

All elements mentioned in this chapter refer to both the Java 2 API and the Java 3D API, unless specifically noted.

5.1 Overview

Data is loaded into a chart by attaching one or more data sources to it. A chartable data source is an object that takes real-world data and puts it into a form that JClass Chart 3D can use. Once your data source is attached, you can chart the data in a variety of ways.

The design of JClass Chart 3D makes it possible to chart data from virtually any real-world source. There is a toolkit you can use to create custom chartable objects (data sources) for your real-world data.

Creating your own data sources can be time-consuming. For that reason, JClass Chart 3D provides pre-built chartable data sources for most common real-world data: files, XML sources, and databases.

To understand data sources better, it is important to differentiate between point data and grid data. To refresh your understanding, please review [Data Types](#), in Chapter 1.

This chapter describes how to use the pre-built data sources and how to create your own.

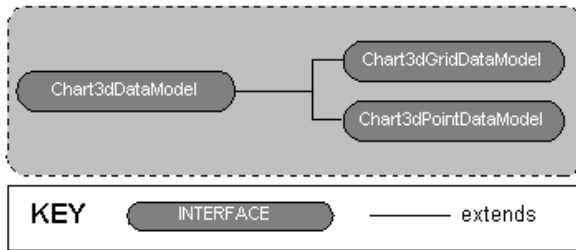
Before delving into how to use the pre-built data sources and how to create your own, a quick review of key elements is important. Please read all of this overview section before proceeding.

5.1.1 Nomenclature

A **data model** is an abstract model, ie, an interface. A **data source** is an implementation of the data model. It is the object that actually creates and manipulates the data.

5.1.2 Chart Data Model Hierarchy

Data Model



5.1.3 Responsibility for Data

It is the application's responsibility to create and manage all required data objects.

The application creates a data source by implementing a data model or set of data models. The application then sets the data source on the `Chart3dDataView` via either its `elevationDataSource` or `zoneDataSource` property. The data view then extracts the data from the data source and stores references to it in an internal data object. The data is not copied.

The application can then get a reference to the internal data object through the data view's `elevationData` and `zoneData` properties. The application can then query the internal object for its data values and set certain properties on it.

Grid Data Versus Point Data

A **grid data source** must implement the `Chart3DGridDataModel`. When it is set on the data view, an internal data object of type `Chart3dGridData` is created. A `Chart3dGridData` object allows you to query its data values and to set certain properties, such as the `xLabels`, `yLabels`, and `chartStyle`.

A **point data source** must implement the `Chart3DPointDataModel`. When it is set on the data view, an internal data object of type `Chart3dPointData` is created. A `Chart3dPointData` object allows you to query its data values (stored in `Chart3dPointSeries` objects) and to set certain properties on the `Chart3dPointSeries` objects, such as its label and its `chartStyle`.

Note that the `elevationDataSource` property can take either a grid data source or a point data source, whereas the `zoneDataSource` property can take only a grid data source. Also, if you set the `zoneDataSource` property, it must have the same number of X- and Y-values as the `elevationDataSource` property.

5.1.4 Changing data

Changing data uses an event mechanism. There are two ways to alert the chart to changes:

- use the `setElevationDataSource` and `setZoneDataSource` methods (the recommended way)
- through the `Chart3dDataEvent` mechanism

In the first method, you change the data in the data source object and then reset the data on the data view. The data is re-extracted and the chart will update.

The second option uses the `Chart3dDataEvent` mechanism. In the data source, when the data changes, a data event is sent to a data listener, and a reaction to the event occurs. `Chart3dDataView` is registered as a data listener for all the data sources it currently references via the `elevationDataSource` or the `zoneDataSource`. Your data source must implement the `Chart3dDataManager` interface for this mechanism to work.

(As an aside, while you can change the data reference to the data source without setting the data source [without calling a listener], it is strongly not recommended, in that the effects will not be known until a redraw is done.)

5.1.5 Internal data

Internal data is meant as a read-only object. You should not set data values through internal data objects. However, you may use internal data to:

- set chart styles
- set data labels – (x,y) for grid data; (series) for point data

5.1.6 Chart3DDataModel interface

In order for a data source object to work with JClass Chart 3D, it must implement the `Chart3dDataModel` interface. The simplest of these are the `Chart3dPointDataModel` interface and the `Chart3dGridDataModel` interface.

Chart3dPointDataModel interface

The `Chart3dPointDataModel` interface is the core point data model interface for JClass Chart 3D. In JClass Chart 3D, point data is specified in terms of a doubly subscripted array of `Point3d` objects.

The `Chart3dPointDataModel` interface has a single method: `getPoints()`, which retrieves a doubly subscripted array of `Point3d` objects (arranged by series):

```
import javax.vecmath.Point3d;
public Point3d[][] getPoints();
```

Thus, if:

```
Point3d[][] points=getPoints();
```

then the third point of the second series is referred to by:

```
points[1][2]
```

When the data view extracts data from the data source, it creates a `Chart3dPointSeries` for each series of points and stores a singly subscripted array of points in each one.

Here is a code sample that gets the point list for the second series from the internal data object.

```
Chart3dPointData pointData=(Chart3dPointData) dataView.getElevationData();
ArrayList series=pointData.getSeries();
Chart3dPointSeries pointSeries=(Chart3dPointSeries) series.get(1);
Point3d[] pointSecond=pointSeries.getPoints();
```

Chart3dGridDataModel interface

The `Chart3dGridDataModel` interface is the core grid data model interface for JClass Chart 3D. In JClass Chart 3D, grid data is specified in terms of an X-array of grid values, a Y-array of grid values, and a doubly subscripted array of z data values.

There are three methods associated with the `Chart3dGridDataModel` interface:

- `getXGrid()` – retrieves the X-grid values array, returning an array of double values representing X-grid points. The values in this array must be strictly increasing.
- `getYGrid()` – retrieves the Y-grid values array, returning an array of double values representing Y-grid points. The values in this array must be strictly increasing.
- `getZValues()` – retrieves the z values -- one for each (x,y) grid point, returning doubly subscripted array of double values representing the z values

When the data view extracts data from a grid data source, references to the `xGrid`, `yGrid`, and `zValue` arrays are stored in a `Chart3dGridData` object. The application can retrieve these from a `Chart3dGridData` object if desired.

The following table outlines the differences between point and grid data – this concept is crucial in working with JClass Chart 3D.

Point Data	JClass Chart 3D Data	Grid Data
<code>Chart3DPointData</code>	Base data model	<code>Chart3DGridData</code>
<code>Chart3DPointDataModel</code>	Interface	<code>Chart3DGridDataModel</code>
<code>series, points</code>	Basic elements	X grid, Y-grid, Z values
<code>Point3D[][]points</code>	Data types	<code>double [] xGrid</code> <code>double [] yGrid</code> <code>double [][] zValues</code>

Point Data	JClass Chart 3D Data	Grid Data
elevationData elevationDataSource	Relevant Chart3DDataView properties	elevationData elevationDataSource zoneData zoneDataSource
Chart3DPointData	Internal data	Chart3DGridData
Chart3DPointSeries --> chartStyle -->label	Internal data setable properties	chartStyle xLabels yLabels xLabelsArrayList yLabelsArrayList
Scatter chart	Chart Types	Bar, surface, or scatter chart

5.2 Pre-Built Chart DataSources

The pre-built DataSources for JClass Chart 3D are located in the `com.klg.jclass.chart3d.data` package. Their names and descriptions follow.

DataSource name	Description
Base3dDataSource	Empty base class that implements the <code>Chart3dDataModel</code> and the <code>HoleValueChart3dDataModel</code> . It extends the <code>Chart3dDataSupport</code> class which implements the <code>Chart3dDataManager</code> interface.
Base3dGridDataSource	Base for any data source that chooses to store data internally using the data arranged in a grid. It implements the <code>Chart3dGridDataModel</code> .
Base3dPointDataSource	Base for any data source that chooses to store data internally using a series of points. It implements <code>Chart3dPointDataModel</code> .
JCDefault3dGridDataSource	Extends <code>Base3dGridDataSource</code> to create a more useful default container for JClass Chart 3D data.
JCDefault3dPointDataSource	Extends <code>Base3dPointDataSource</code> to create a more useful default container for JClass Chart 3D data.
JCEditable3dGridDataSource	Extends <code>JCDefault3dGridDataSource</code> with convenience methods that permit data editing. It implements <code>EditableChart3dDataModel</code> .

DataSource name	Description
JCEditable3dPointDataSource	Extends JCDefault3dPointDataSource with convenience methods that permit editing of data. It implements EditableChart3dDataModel.
JCFile3dDataSource	Convenience class that parses data from a file.
JCSwing3dDataSource	A 3d DataSource that converts a Swing TableModel into a form usable by JClass Chart 3D. Extends the JCEditable3dGridDataSource.
JCXML3dDataSource	Parses data from an XML file. Extends the JCDefault3dGridDataSource.
JDBC3dDataSource	Extends JCDefault3dDataSource to create a data source for use with JDBC.

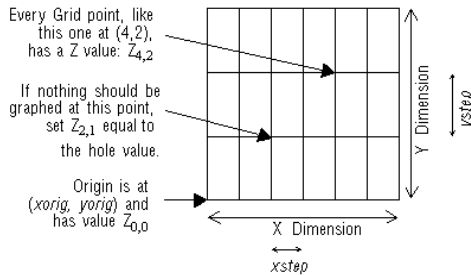
5.3 Loading Data from a File

Data that is read from a file is read as regular grid data or irregular grid data.

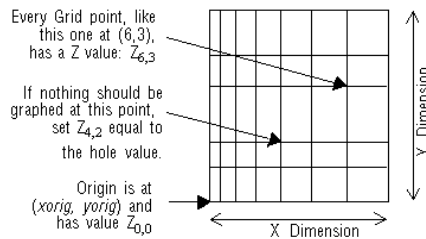
5.3.1 Regular and Irregular Grid Data

Basically, regular grid data has X- and Y-values at regular intervals. Irregular grid data does not.

Regular Grid Data



Irregular Grid Data



An easy way to bring data into a chart is to load it from a formatted file using `JCFile3dDataSource`. To load data this way, you create a data file that follows `JClass Chart 3D`'s standard format, as outlined in Section 5.3.2, [Standard file format](#).

Then you call `JCFile3dDataSource`'s static `createDataSourceFromFile` method to create a data source from a file. Here's a code snippet showing this action:

```
Chart3dDataModel dataSource=JCFile3dDataSource.createDataSourceFromFile  
("file.dat");
```

or

```
chart3d.getDataView(0).setElevationDataSource(JCFile3dDataSource.create  
createDataSourceFromFile ("igrid.dat"));
```

5.3.2 Standard file format

The `JCFile3dDataSource` class is a convenience class that parses data from a file. Do not create an instance of this class; rather, use the static method `createDataSourceFromFile(String fileName)` as described in the previous section.

This method returns an object of type `Base3dDataSource` which must be cast to the appropriate type (either `JCEditable3dGridDataSource` or `JCEditable3dPointDataSource`). The input file can either contain an irregular or regular

grid data or point data. The key words GRID, IGRID, or POINT identify the data format the file contains.

A regular grid file has the following format:

```
# Comments use the # sign
# For this example the grid has 5 by 3 points
# The grid dimensions are optionally followed by xLabels and yLabels
GRID
5 'A A' 'B' 'C' 'D' 'E'

3 'Y1' 'Y2' 'Y3'

# Holes have value 100.0
# The Grid increases in
# X steps of 1.0 and Y steps of 2.0
# The origin of the Grid is x = -20.0 and y = 50.0
100.0 1.0 2.0 -20.0 50.0

# 15 data points would follow, 1 for each point
49.875000 43.765625 38.50000 33.984375 30.12400
26.828125 24.0000 21.656875 19.375000 17.39062 16.222 18.444 23.555
58.664 37.894564
```

An Irregular Grid would supply all the X- and Y-values as well as the data points. An irregular grid file has the following format:

```
# Irregular grid has 5 by 3 points
# The grid dimensions are optionally followed by xLabels and yLabels
IGRID
5 'A A' 'B' 'C' 'D' 'EE'

3 'Y1' 'Yahoo' 'Y3'

# Holes have value 100.0
100.0

# 5 x values are given
# 3 y values are given
20 21.1 22.3 23 24.4
50.3 51.3 52.6

# 15 Data values follow
23.34343 12.89239 11.99423 15.781212 18.18989
26.828125 24.0000 21.656875 19.375000 17.39062 16.222 18.444 23.555
58.664 37.894564
```

A point data file would look like this:

```
# There are 3 series
# The series number is optionally followed by series labels
# The hole value is -1000
POINT 3 'Series 1' 'Series 2' 'Series 3'
-1000

# The are 5 points in series 1
# 5 points follow in (x, y, z) format
5
5.65 6.24 1.78
7.41 7.26 4.21
5.45 5.44 1.43
0.97 9.66 3.41
3.86 1.42 0.20

# The are 4 points in series 2
4
6.57 7.43 8.37
3.79 3.63 2.65
7.89 3.48 5.65
0.78 7.03 0.65

# The are 6 points in series 3
6
9.91 7.54 1.74
6.53 4.62 1.99
8.41 3.49 5.06
7.85 9.16 0.64
6.96 9.18 8.95
3.47 3.19 6.29
```

5.4 Loading Data from a Swing TableModel

The `JCSwing3dDataSource` class enables you to use any data object that implements Swing's `TableModel` interface as a `JClass Chart 3D` data source. The `TableModel` interface is typically used for Swing `JTable` components, so your application may already have created a data object of this type.

`JCSwing3dDataSource` “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 `elevationSwingDataModel` and the `zoneSwingDataModel` properties in `JClass Chart 3D`'s `JavaBeans`. To use them, prepare your data in a `Swing TableModel` object and set the `SwingDataModel` property to that object.

5.5 Loading Data from an XML Source

5.5.1 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, and so on.

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.

Here are links to more information on XML.

- <http://www.w3.org/XML/> – another W3C site; contains exhaustive information on standards. Of particular note are the XML schema 1 (structures) and XML schema 2 (datatypes) working drafts. They make up an extension that specifies how to constrain XML documents to particular schema. This is important if you want to represent database data or object-oriented data as XML.
- <http://www.java.sun.com/docs/index.html> – Sun’s XML site
- <http://www.oasis-open.org/cover/xml.html> – thorough list of links to XML papers and ongoing work

5.5.2 Using XML in JClass

In order to work with XML in your programs or even to compile the JClass XML examples, you will need to have *jaxp.jar* in your CLASSPATH. Additionally, you will need at least a DOM level 2 parser, such as *crimson.jar*. Both of these JAR files are distributed with JClass Chart 3D – you can find them in *JCLASS_HOME/lib/*.

Please note that XML may be used for grid data only, not point data.

The *JCXML3dDataSource* class parses data in XML format. Specification for the XML data can be found in the *chart3d.dtd* file (*JCLASS_HOME/com/kg/jclass/xml-dtd/*).

Example of XML in JClass

For an XML data source example, the XML Chart example is in *JCLASS_HOME/examples/chart3d/j2d/data*. This example uses the *3d.xml* data file.

XML Constructor

The `JCXML3dDataSource` constructor takes an `InputStream` in XML form and may also take a `Reader` that contains XML, a file, a `String`, or other input source.

Example XML data file

Here is an example of an XML data file specifying chart data according to the supported .DTD file. Labels are optional, as are hole and name. Grids are required, and each must contain at least one `val` element.

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "chart3d.dtd">
<data hole="-100" name="my data">

  <xgrid>
    <xval>0.78</xval>
    <xval>1.565</xval>
    <xval>2.00</xval>
  </xgrid>

  <ygrid>
    <yval>1.00</yval>
    <yval>2.0</yval>
    <yval>3.0</yval>
    <yval>4.0</yval>
    <yval>5.0</yval>
  /ul>
</ygrid>

  <zgrid>
    <zval>15.64</zval>
    <zval>23.4546</zval>
    <zval>45.4545</zval>
    <zval>20.4546</zval>
    <zval>14.4545</zval>
  </zgrid>

  <zgrid>
    <zval>18.5656</zval>
    <zval>23.884</zval>
    <zval>35.6454</zval>
    <zval>21.47</zval>
    <zval>37.45</zval>
  </zgrid>

  <zgrid>
    <zval>16.58</zval>
    <zval>10.5656</zval>
    <zval>17.65</zval>
    <zval>19.645</zval>
    <zval>34.4561</zval>
  </zgrid>

  <xlabel>January</xlabel>
  <xlabel>February</xlabel>
  <xlabel>March</xlabel>

  <ylabel>0rcs</ylabel>
  <ylabel>Zombies</ylabel>
  <ylabel>Skeletons</ylabel>

```



```
<ylabel>Vampires</ylabel>
<ylabel>Werewolves</ylabel>
</data>
```

5.6 Data Binding using JDBCDataSource

JDBC3dDataSource is not a full data binding solution. It is a data source that you can use to chart data from an SQL Result Set. It does not perform any binding operations such as connecting to or querying the database. You will have to provide that functionality in your application.

To use it, you just attach an instance of JDBC3dDataSource to your chart and pass it a Result Set from your application, as follows:

```
chart.getDataView(0).setElevationDataSource(new JDBC3dDataSource
(myResultSet));
```

5.7 JCData3dUtil class

The JCData3dUtil class (com.klg.jclass.chart3d.data.JCData3dUtil) is a utility class that contains convenience methods for manipulating, filtering, and copying JClass Chart 3D data models.

The source for all of the following methods is the Chart3dGridDataModel class.

- Use the createDataCopy() method to create a copy of Chart3dGridDataModel's X-grid, Y-grid, and Z values.
- Use the dataCopy() method to copy the xGrid, yGrid and zValues from a source to a destination data model.
- Use createShadedDataModel(double sweepAngle, double riseAngle, double brightness, double ambient) to create a data model that simulates light reflecting off the surface of the source data model with Z values ranging from 0 (no reflection) to 1 (full reflection), as well as an ambient light source. The data model returned from this method may then be used as zone data for the source to produce a grey scale effect. See the Shaded demo, which comes with your JClass Chart 3D distribution, for an example of how to use this method.
- Use createSmoothedDataModel(double centerWeight) to create a “smoothed” data model based on the source data model. The amount of smoothing may be controlled by varying the centerWeight argument. When it is 0, the “smoothed” Z value of a data point is based entirely on the weighted average of its neighbors. When it is 1, no smoothing takes place. When it is between 0 and 1, the result is influenced by its neighbors in proportion to the centerWeight.

- Use `createCubicSampledDataModel (double[] xSamples, double[] ySamples)` to create a data model which is a subset of source using cubic Interpolation with an xGrid described by `xSamples` and a yGrid described by `ySamples`. An alternative to this method is to use `createLinearSampledDataModel (double[] xSamples, double[] ySamples)` method since the linear interpolation method used to calculate the Z values is generally faster, though it usually produces a coarser approximation of the source data. These last two methods are also available in several other flavours. Please refer to the JClass Chart 3D API for more examples.

5.8 Making Your Own Chart Data Source

5.8.1 The Simplest Chart Data Source Possible

In order for a data source object to work with JClass Chart 3D, it must implement the `Chart3dDataModel` interface. The `EditableChart3dDataModel` interface can be used in conjunction with the `Chart3dDataModel` when you want to allow the data source to be editable. The `LabelledChart3dDataModel` and the `HoleValueChart3dDataModel` interfaces can also be used in conjunction with `ChartData3dModel` to extend its functionality to allow for label values (via the `LabelledChart3dDataModel` interface) and hole values (via the `HoleValueChart3dDataModel` interface). The `LabelledChart3dDataModel` has an extension for grid data and for point data, which calls methods specific to the type of data.

The `Chart3dDataModel` interface is intended for use with existing data objects. If a data object implements its extensions, `Chart3dGridDataModel` and `Chart3dPointDataModel`, it provides a way for the Chart to extract the data from the data source. For example, the `Chart3dGridDataModel` allows JClass Chart 3D to ask the data source for the x-grid values, y-grid values, and for z values. The interface looks like this:

```
public double[] getXGrid();
public double[] getYGrid();
public double[][] getZvalues();
```

The values returned by `getXGrid()` and `getYGrid()` do not have to be equally spaced, but they must be in strictly increasing order. The length of the first dimension of the doubly subscripted array returned by `getZvalues()` should be the same as the length of the xGrid array. Also, the length of each array that makes up the second dimension of zValues should be the same as the yGrid array. If this is not true, the shortest length will be used for all related arrays.

As an example, consider the following code snippets, taken from *JCChart3d.java*. This is the actual default data source for JClass Chart 3D.

The following three methods would allow an object to implement the `Chart3dGridDataModel`.

```
/**
 * Method required to implement Chart3dGridDataModel interface.
 * Retrieves the <i>x</i> grid values
 *
 * @return array of double values representing <i>x</i> grid points
 */
public double[] getXGrid()
{
    double xarray[] = new double[11];
    for (int i = 0; i < xarray.length; i++) {
        xarray[i] = (double)i;
    }
    return(xarray);
}

/**
 * Method required to implement Chart3dGridDataModel interface.
 * Retrieves the <i>y</i> grid values
 *
 * @return array of double values representing <i>y</i> grid points
 */
public double[] getYGrid()
{
    double yarray[] = new double[11];
    for (int i = 0; i < yarray.length; i++) {
        yarray[i] = (double)i;
    }
    return(yarray);
}

/**
 * Method required to implement Chart3dGridDataModel interface.
 * Retrieves the <i>z</i> values -- one for each (<i>x</i>,<i>y</i>)
 * grid point
 *
 * @return doubly subscripted array of double values representing
 * <i>z</i> values
 */
public double[][] getZValues()
{
    double zvals[][] = new double[11][11];
    for (int i = 0; i < zvals.length; i++) {
        for (int j = 0; j < zvals[i].length; j++) {
            double xval = (double)(i-5);
            double yval = (double)(j-5);
            zvals[i][j] = xval*xval + yval*yval;
        }
    }
    return(zvals);
}
```

The following method would allow an object to implement the `Chart3dPointDataModel`.

```
/**
 * Method required to implement Chart3dPointDataModel interface.
 * Retrieves a list of points for a scatter plot.
 */
public Point3d[][] getPoints()
{
    Point3d[][] series = new Point3d[5][];
    int sizes[] = {3, 5, 6, 4, 2};

    for (int i = 0; i < series.length; i++) {
        Point3d[] points = new Point3d[sizes[i]];
        for (int j = 0; j < points.length; j++) {
            points[j] = new Point3d((double)(20 + 2*i),
                (double)(30 + j), (double)(10*i + j));
        }
        series[i] = points;
    }
    return(series);
}
```

5.8.2 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 `LabelledChart3dDataModel` interface. For grid data, one should implement the `LabelledChart3dGridDataModel`, and for point data, the `LabelledChart3dPointDataModel`. Both extend the `LabelledChart3dDataModel`.

The `LabelledChart3dDataModel` interface contains one method:

```
public String getDataSourceName();
```

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

LabelledChart3dGridDataModel and LabelledChart3dPointDataModel

The `LabelledChart3dGridDataModel` interface specifies X-labels and Y-labels for a JClass Chart 3D data model. `xLabels` and `yLabels` are used for the X- and Y-axis respectively if the annotation type of the axis is `JCAxis.ANNOTATION_DATA_LABELS`. This interface also specifies the number of X-grid and Y-grid values.

The `LabelledChart3dPointDataModel` interface specifies the number of series and the series labels for a JClass Chart 3D data model. Series labels are displayed in the legend.

Note that both interfaces are used only in conjunction with the `Chart3dDataModel` interface, which means that, in order for an object to be recognized as a chart data source, it needs to implement the `Chart3dDataModel` interface.

The `Base3dGridDataSource` class – the base for any data source that chooses to store data internally using data arrayed in a grid – implements the `LabelledChart3dGridDataModel`

interface. The `Base3dPointDataSource` class – the base for any data source that chooses to store data internally using a series of points – implements the `LabelledChart3dPointDataModel` interface.

5.8.3 EditableChartDataModel – Modifying Your Data

If you want users to modify data using the edit action in JClass Chart 3D, your data source must implement the `EditableChart3dDataModel` interface. This interface is used in conjunction with a `Chart3dGridDataModel` or a `Chart3dPointDataModel`, and the data object must also implement one of these interfaces to be recognized as a data source. The `EditableChart3dDataModel` has a single method:

```
public boolean setZValue(JCData3dIndex index, double newValue);
```

For grid data, the index will be of type `JCData3dGridIndex` and the X- and Y-indices of the `newValue` can be extracted from the index object. For point data, the index will be of type `JCData3dPointIndex` and the series and point indices can again be extracted from the index object. Note that the `EditableChart3dDataModel` only allows for Z values to be changed. In other words, `newValue` is a z value.

Here is a code example showing how to set the `zValue` of a point for a given series and point index for a point data array (stored in `points`):

```
/**
 * Sets the zValue of a point for a given series and point index. This
 * series and point index is retrieved from the passed in data index,
 * which must be of type JCData3dPointIndex.
 * @param index The data index from which the series and point indices
 * of the point to be edited is obtained.
 * @param newValue <code>Point3d</code> object for that position
 * @return Whether the edit succeeded
 */
public boolean setZValue(JCData3dIndex index, double newValue)
{
    if (index == null || !(index instanceof JCData3dPointIndex)) {
        return(false);
    }
    JCData3dPointIndex pointDataIndex = (JCData3dPointIndex)index;
    int seriesIndex = pointDataIndex.getSeries();
    int pointIndex = pointDataIndex.getPoint();
    if (seriesIndex < 0 ||
        seriesIndex >= points.length ||
        pointIndex < 0 ||
        pointIndex >= points[seriesIndex].length)
    {
        return(false);
    }
    Point3d point = points[seriesIndex][pointIndex];
    point.z = newValue;
    int type = Chart3dPointDataEvent.RELOAD_POINT;
    fireChart3dDataEvent(new Chart3dPointDataEvent(this, type,
        pointDataIndex));
    return(true);
} //setZValue
```

Here is a code example showing how to set the `zValue` for a given data index for grid data (stored in the `zValues` array):

```
/**
 * Sets the point for a given data index which must be of type
 * JCData3dGridIndex.
 *
 * @param index The data index from which the x and y indices of the
 * point to be edited is obtained.
 * @param newValue The new z value for the point
 * @return Whether edit succeeded
 */
public boolean setZValue(JCData3dIndex index, double newValue)
{
    if (index == null || !(index instanceof JCData3dGridIndex)) {
        return(false);
    }
    JCData3dGridIndex gridIndex = (JCData3dGridIndex)index;
    int xIndex = gridIndex.getX();
    int yIndex = gridIndex.getY();
    if (xIndex < 0 ||
        xIndex >= zValues.length ||
        yIndex < 0 ||
        yIndex >= zValues[xIndex].length)
    {
        return(false);
    }
    zValues[xIndex][yIndex] = newValue;
    int type = Chart3dGridDataEvent.RELOAD_ZVALUE;
    fireChart3dDataEvent(new Chart3dGridDataEvent(this, type,
        gridIndex));
    return(true);
} //setDataValue
```

In this example, the value is saved back into the `zValues` array from `JCDefault3dGridDataSource`, using the `xIndex` and `yIndex` values to index to the appropriate array member.

The `EditGrid` and `EditPoint` examples (found in the [JCLASS_HOME/examples/chart3d/j2d/data](#) directory) demonstrate how to use the `EditableChartDataModel` interface.

5.9 HoleValueChartDataModel – Specifying Hole Values

If you want to supply a specific hole value along with your data, your data source must implement the `HoleValueChart3dDataModel` interface.

A hole value is a particular value for which the chart will not draw anything each time the hole value is encountered in the data. For a surface chart, the facets surrounding the value are not drawn. For a bar chart, that particular bar is not drawn. For a scatter plot, the point is not drawn.

The `HoleValueChart3dDataModel` interface has one method, `getHoleValue()`. This method retrieves the hole value for the data source.

5.10 Making an Updating Chart Data Source

Quite often, the data shown in JClass Chart 3D is dynamic. This kind of data requires creation of an updating data source. An updating data source is capable of informing a chart that a portion of the data has been changed. JClass Chart 3D can then act on the change.

JClass Chart 3D uses the standard AWT/Swing event/listener mechanism for passing changes between the chart data source and JClass Chart 3D. At a very high level, JClass Chart 3D is a listener to data source events that are fired by the data source.

5.10.1 Chart Data Source Support Classes

There are a number of data source related support classes included with JClass Chart 3D. These classes make it easier to build updating data sources.

Chart3dDataEvent and Chart3dDataListener

The `Chart3dDataListener` interface is implemented by objects interested in receiving `Chart3dDataEvents`. Most often, the only `Chart3dDataListener` is JClass Chart 3D itself. `Chart3dDataEvent` and `Chart3dDataListener` give data sources a way to send update messages to JClass Chart 3D.

The `Chart3dDataListener` interface has only one method:

```
public void chart3dDataChange(Chart3dDataEvent e);
```

Thus, this mechanism uses the `Chart3dDataEvent` class to inform the listener of a change. *In most systems, only JClass Chart 3D need implement this interface.* The `Chart3dDataView` is the class that implements the `Chart3dDataListener` interface within JClass Chart 3D.

The `Chart3dGridDataEvent` class, which extends `Chart3dDataEvent`, is used to encapsulate a JClass Chart 3D grid data change event. This class has two methods: `getX` and `getY`. `getX` returns the X-index of the affected data, returning `JCData3dIndex.ALL` if all X-values are affected. `getY` returns the Y-index of the affected data, returning `JCData3dIndex.ALL` if all Y-values are affected.

The `Chart3dPointDataEvent` class, which also extends `Chart3dDataEvent`, retrieves the point index associated with the event. This class has two methods: `getPoint` and `getSeries`. `getPoint` returns the index of the point affected, returning `JCData3dIndex.ALL` if all points are affected. `getSeries` retrieves the series index associated with the event, returning `JCData3dIndex.ALL` if all points are affected.

The `Chart3dDataEvent` has a type property that indicates the message type of the event. The message type delineates what type of update the data source has made.

Properties of the `Chart3dDataEvent` class – note that these properties are relevant for **both** grid and point data:

Message Type	Meaning
RELOAD_DATA_SOURCE_NAME	Enum value indicating the data source name is attached.
RESET	Enum indicating that everything – data, labels, name, etc – has changed.
RELOAD	Enum indicating that the data needs to be reloaded.
RELOAD_HOLE_VALUE	Enum value indicating the hole value is affected.

Properties in the `Chart3dGridDataEvent` class:

Message Type	Meaning
RELOAD_ALL_XLABELS	Enum value indicating all xLabels are affected.
RELOAD_ALL_YLABELS	Enum value indicating all yLabels are affected.
RELOAD_XGRID	Enum value indicating all X-values are affected.
RELOAD_XLABEL	Enum value indicating a particular xLabel is affected.
RELOAD_XVALUE	Enum value indicating a given X-value is affected.
RELOAD_YGRID	Enum value indicating all Y-values are affected.
RELOAD_YLABEL	Enum value indicating a particular yLabel is affected.
RELOAD_YVALUE	Enum value indicating a given Y-value is affected.
RELOAD_ZALL	Enum value indicating the entire doubly indexed array of zValues is affected.
RELOAD_ZARRAY	Enum value indicating one array of zValues is affected.
RELOAD_ZVALUE	Enum value indicating one z Value is affected.

Properties in the `Chart3dPointDataEvent` class:

Message Type	Meaning
ADD_SERIES	Enum value indicating a data series has been added to the end of the set of data series.
INSERT_SERIES	Enum value indicating a data series has been inserted at a particular index in the set of data series.

Message Type	Meaning
RELOAD_ALL_SERIES_LABELS	Enum value indicating all series labels need to be reloaded.
RELOAD_POINT	Enum value indicating a single data value has changed.
RELOAD_SERIES	Enum value indicating a data series has changed.
RELOAD_SERIES_LABEL	Enum value indicating a particular series label needs to be reloaded.
REMOVE_SERIES	Enum value indicating a data series has been removed.

Chart3dDataManager

The `Chart3dDataManager` interface is used by a data source to tell JClass Chart 3D that it will be sending a `Chart3dDataEvent` to JClass Chart 3D. Without this interface, there is no way for JClass Chart 3D to know that it has to attach itself as a `Chart3dDataListener` to the data source.

The two methods involved to add and remove a JClass Chart 3D data listener:

- `addChart3dDataListener(Chart3dDataListener n)` – adds a chart 3D data listener.
- `removeChart3dDataListener(Chart3dDataListener n)` – removes a chart 3D data listener.

A `Chart3dDataManager` is an object that knows how to register and deregister `Chart3dDataListeners`. Chart uses this object to register itself as a listener to events from the data source.

The quickest way to get such a data source set up is to extend or use the `Chart3dDataSupport` class.

Chart3dDataSupport

`Chart3dDataSupport` provides a default implementation of `Chart3dDataManager`. It will manage a list of `Chart3dDataListeners`. It also provides two convenience methods for firing events to the listeners.

The first `fireChart3dDataEvent` method fires a chart data event to any registered listeners. This version of `fireChart3dDataEvent` will create a `Chart3dDataEvent` object out of the message and a `JCData3dIndex`. For example:

```
public void fireChart3dDataEvent(int type, JCData3dIndex index)
```

where `type` is a valid message type from `Chart3dDataEvent`, `Chart3dGridDataEvent`, or `Chart3dPointDataEvent`, and `index` is the data index which tells which (x, y) or (series, point) to which this event refers.

If you already have a `Chart3dDataEvent`, the second `fireChart3dDataEvent` method fires this event to any registered listeners. For example:

```
public void fireChart3dDataEvent(Chart3dDataEvent evt)
```

where `evt` is the event to send to registered listeners.

Creating an Updating Data Source

If your `datasource` either extends or contains `Chart3dDataSupport`, sending updates from the data source to the chart is easy. Simply call `fireChart3dDataEvent()` with the event you wish to send.

```
fireChart3dDataEvent(Chart3dDataEvent.RESET,  
    new JCData3dGridIndex(x,y));
```

To have `JClass Chart 3D` automatically added as a listener, your data source needs to implement the `Chart3dDataManager` interface.

If you do implement this interface, then the `Chart3dDataView` object will automatically register itself as a listener when you set the data source on either its `elevationDataSource` or `zoneDataSource`. Note that the listener will remove itself when another data source replaces your data source in the data view.

5.11 Summary of JClass Chart 3D Data Interfaces

Characteristic	Base	Grid Data	Point Data
Data	<code>Chart3dDataModel</code>	<code>Chart3dGridDataModel</code>	<code>Chart3dPointDataModel</code>
Hole value	<code>HoleValueChart3dDataModel</code>	none	none
Labels – data source name	<code>LabelledChart3dDataModel</code>	<code>LabelledChart3dGridDataModel</code>	<code>LabelledChart3dPointDataModel</code>
Setting new Z values	<code>EditableChart3dDataModel</code>	none	none
Managing listeners	<code>Chart3dDataManager</code>	none	none
Listening for data events	<code>Chart3dDataListener</code>	none	none

Advanced JClass Chart 3D Programming

4D Surface Graphs ■ *4D Bar Charts* ■ *Customizing the Contour Levels*
Customizing Contour Styles ■ *Internationalization Support*

All elements mentioned in this chapter refer to both the Java 2 API and the Java 3D API, unless specifically noted.

This section covers topics that programmers of advanced JClass Chart 3D applications will find useful. It assumes that you are already familiar with JClass Chart 3D.

6.1 4D Surface Graphs

For surface and bar charts, JClass Chart 3D can be used to display 4D charts using color as a fourth dimension. The additional color information is provided to JClass Chart 3D as a second data source using the `Chart3dDataView` class' `zoneDataSource` property.

A full description of data sources, including grid data sources, is provided in [Data Sources](#), in Chapter 5.

For zone data, use the `Chart3dGridDataModel` interface. The `Chart3dGridDataModel` interface, which extends `Chart3dDataModel`, is the core grid data model interface. In JClass Chart 3D, grid data is specified in terms of an X-array of grid values, a Y-array of grid values, and a doubly subscripted array of z data values.

To create a 4D chart:

- Set the `Zoned` property of the `JCContour` class and the `Shaded` property of the `JCElevation` class to `true`.
- Set a grid data source in the `Chart3dDataView`'s `elevationDataSource` property.
- Set a grid data source in the `Chart3dDataView`'s `zoneDataSource` property; this will be used for deriving the zoning and contouring colors as zone data.

- Ensure that the data array sizes of the two data sources match up. The xGrid and yGrid arrays of the two data sources should be identical. The zValues array should have exactly the same number of values in both the X- and Y-dimensions (that is, the same number of rows and columns).

Note: If any of these conditions are not met, a 4D chart will not be displayed.

If the zone data has a hole that is not in the surface data, the surface in the region of the hole will be displayed as if the zone data were not attached.

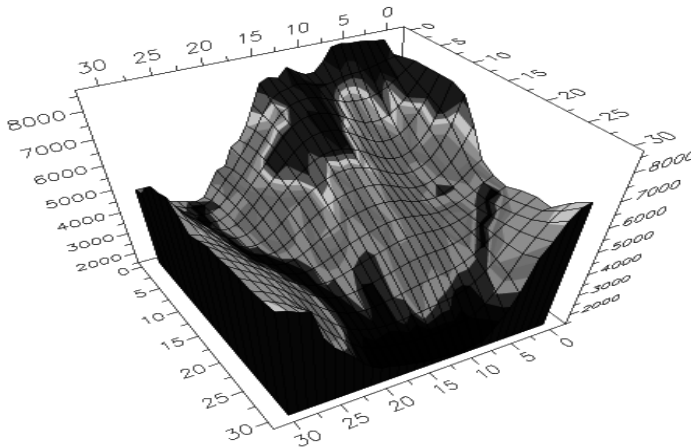


Figure 21 4D chart – zone/contour data is different from surface data.

6.2 4D Bar Charts

When a zone data source is supplied for a bar chart, the values in the zone data are used in conjunction with the contour levels to apply zone colors to the bars in the grid.

When zone data is supplied and the `Zoned` property of the `JCContour` class is `true`, the bar is not broken up into separate colored segments. Rather, each bar is individually colored

according to the zoned height of the bar. Contours are never drawn when zone data is supplied. Figure 22 gives an example of a 4D bar chart.

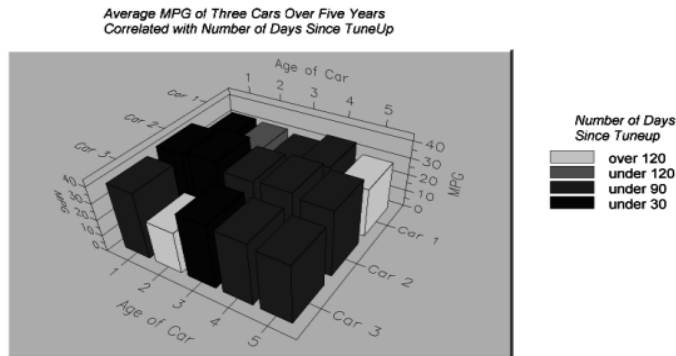


Figure 22 A 4D Bar Chart.

In a bar chart, the zone data structure is only referenced when `Zoned` is `true`. A legend is generated based on the contour levels. The legend labels can be replaced by supplying a list of labels to the `JCChart3dLegend` class via its `labels` property.

Please see [Legends](#), in Chapter 2 for full details on legend Strings.

6.3 Customizing the Contour Levels

To customize contour levels, manipulate the `JCContourLevels` class. This object deals with contour levels; it calculates default levels (if `isDefault` is `true`) but also allows users to set their own levels.

The `JCContourLevels` class has five properties:

- `isDefault` – ascertains whether linear contour levels are generated automatically (based on `numLevels`)
- `levels` – a strictly increasing array of contour levels
- `max` – the contour maximum, calculated from the data (read only)
- `min` – the contour minimum, calculated from the data (read only)
- `numLevels` – the number of contour levels

To specify your own contour levels, set the `levels` property to a new array of doubles. This array must be in strictly increasing order. You can also manipulate the levels using the `addLevel()` and `removeLevel()` methods. Changing the levels in any of the above ways has the side effect of setting the `isDefault` property to `false`. When `isDefault` is `false`, the `numLevels` property automatically takes on the value of the length of the `levels` array

and hence becomes read only. If `isDefault` is set to `true`, the `numLevels` property can be changed to specify the number of default levels to calculate.

In user-specified contour levels, the default is to display just the range of data that is spanned by the data (`JCChart3dLegend.RANGE_DATA`). If the `distributionRange` property of the `JCChart3dLegend` class is set to `JCChart3dLegend.Range_ALL`, the entire contour level array is shown in the legend.

6.4 Customizing Contour Styles

To customize contour styles, manipulate the `contourStyles` property of the `JCContour` class. The `contourStyles` property reference an `ArrayList` of `JCContourStyle` objects. The `JCContourStyle` class defines the style used to draw contours and zones.

The `JCContourStyle` class has three properties:

- `fillStyle` – the `JCFillStyle` object to be used for this contour style; used to draw the contour zones
- `lineStyle` – the `JCLineStyle` object to be used for this contour style; used to draw the contour lines
- `symbolStyle` – the `JCSymbolStyle` object to be used for this contour style; used to draw symbols for contoured grid scatter plots

6.4.1 Default Contour Styles

By default, for surface and bar charts, JClass Chart 3D provides an array of 100 contour styles. By default, the fill styles are solid, the contour lines are black lines of width 1, and the symbols are of type `JCSymbolStyle.DOT` of size 6. The fill colors and symbol colors are chosen from a predefined array of colors chosen to provide a pleasing color distribution.

You will need to provide custom contour styles if:

- it is important to your application to specify the precise contour style for any particular level;
- you want to display more than 100 levels; or
- you want to uniquely identify contour lines. The JClass Chart 3D default contour styles use only black solid lines of width 1.

It is usually easiest to specify more contour styles than will be needed for the number of contour levels. If *nstyles* contour styles are provided, and the number of contour levels is *nlevels*, JClass Chart 3D will calculate the index into the contour styles array for level *i* ($0 \leq i \leq nlevels$) as follows:

$$(int) \text{Math.round}(i * (nstyles - 1) / nlevels)$$

If you wish to change this index calculation, implement the `JCContourMapping` interface by creating the method:

```
public int contourIndex(int level);
```

Then set the object that implements the interface on the `contourMapping` property of the `JCContour` class.

The `JCContourStyle` class contains information about how `JClass Chart 3D` should display contour style objects. The fields are broken down as follows:

<code>fillcolor</code>	The color used to demarcate the level when zoned is true.
<code>fillpattern</code>	The pattern used to fill the zones. 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> .
<code>linecolor</code>	The color used to demarcate the level's contour line when contoured is true.
<code>linewidth</code>	The line width used to demarcate the level's contour line when contoured is true. Must be greater than or equal to 0. When <code>linewidth</code> is zero, no line is drawn.
<code>linepattern</code>	The line pattern used to demarcate the level's contour line when contoured is true. Line patterns are only honored for surface plots that are 2D projections (available in the Java 2 API version of <code>JClass Chart 3D</code> by setting the <code>meshed</code> and <code>shaded</code> properties of the <code>JCElevation</code> class to false). The line pattern must be one of the patterns listed in Figure 23.




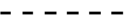


	<code>JCLineStyle.NONE</code>
	<code>JCLineStyle.SOLID</code>
	<code>JCLineStyle.LONG_DASH</code>
	<code>JCLineStyle.SHORT_DASH</code>
	<code>JCLineStyle.LSL_DASH</code>
	<code>JCLineStyle.DASH_DOT</code>

Figure 23 Different Line Patterns.

Here is a code sample that doubles the line width of the 50th contour style.

```
JCContour contour=dataView.getContour();
ArrayList contourStyles=getContourStyles();
JCContourStyle style=(JCContourStyle)contourStyles.get(49);
style.setLineWidth(2*style.getLineWidth());
```

6.5 Internationalization Support

6.5.1 Internationalization

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

Localization is the process of making internationalized software run appropriately in a particular environment. All Strings used by JClass that need to be localized (that is, Strings that will be seen by a typical user) have been internationalized and are ready for localization. Thus, while localization stubs are in place for JClass, this step must be implemented by the developer of the localized software. These Strings are in resource bundles in every package that requires them. Therefore, the developer of the localized software who has purchased source code should augment all .java files within the */resource/* directory with the .java file specific for the relevant region; for example, for France, *LocaleInfo.java* becomes *LocaleInfo_fr.java*, and needs to contain the translated French versions of the Strings in the source *LocaleInfo.java* file. (Usually the file is called *LocaleInfo.java*, but can also have another name, such as *LocaleBeanInfo.java* or *BeanLocaleInfo.java*.)

Essentially, developers of the localized software create their own resource bundles for their own locale. Developers should check every package for a */resources/* directory; if one is found, then the .java files in it will need to be localized.

For more information on internationalization, go to:
<http://java.sun.com/j2se/1.4.2/docs/guide/intl/index.html>.

Programming User Interaction

Default User Interaction ■ *Listeners* ■ *Mapping and Picking* ■ *dragZValue Method*
gridValue Method

All elements mentioned in this chapter refer to both the Java 2 API and the Java 3D API, unless specifically noted.

This chapter describes the user-interaction features of JClass Chart 3D – how a user can interact with the chart and how an application can control interaction.

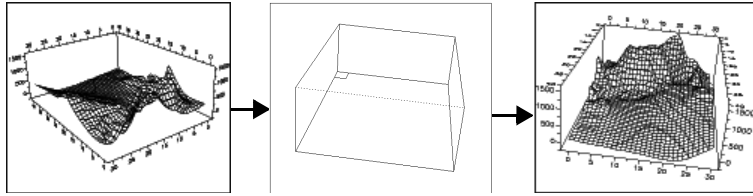
7.1 Default User Interaction

By default, an end-user can rotate, translate, scale, and zoom into the plot cube. The rotate action is implemented by manipulating the three rotation angles of the `JCView3d` class. Scaling, translating, and zooming are achieved by appropriately setting the scale, horizontal shift, and vertical shift properties of the `JCViewport` class.

Figure 24 shows the user interactions enabled by JClass Chart 3D's default translations.

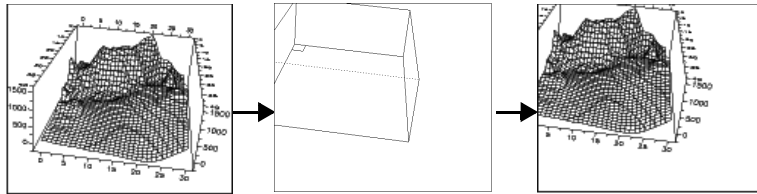
Rotation

- Press x, y, z, e, or n to select an axis. Selected axis defaults to none (or n).
- Press Ctrl and hold down MB1.
- Move mouse counter-clockwise to rotate view clockwise if no X- or Y-axis is selected. Otherwise move mouse perpendicular to the selected axis.



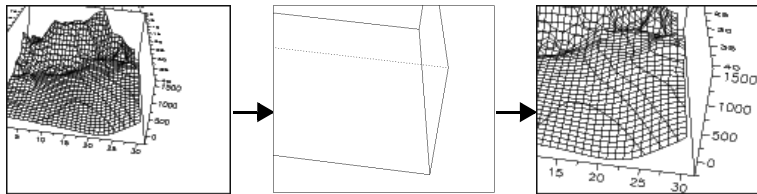
Translation

- Press Shift and hold down MB1 both mouse buttons
- Move mouse to shift the chart



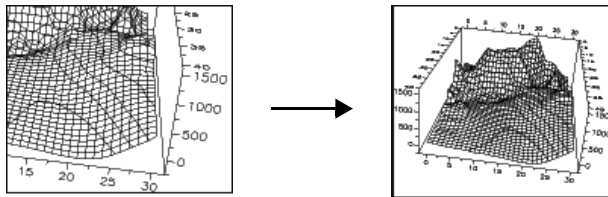
Scaling

- Press Alt and hold down MB1 both mouse buttons
- Move mouse down to zoom in
- Move mouse up to zoom out



Return to Default

- Press "r"
- All scaling, translation, and zooming removed



Zooming

- Press Ctrl and Shift and hold down MB1 left mouse button
- Move mouse to select the area to zoom into

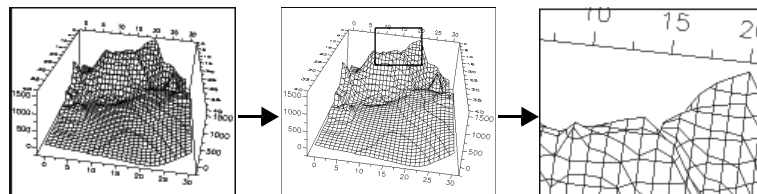


Figure 24 JClass Chart 3D's default user interactions (MB1 Mouse Button 1).

Note that the blank preview cube gets drawn only if the `previewMethod` property of the `JCViewport` class is set to `PREVIEW_CUBE` and you are using the Java 2 version of `JClass Chart 3D`.

7.1.1 **JCViewport class**

The `JCViewport` class stores information concerning the viewport through which the user views the plot cube. The default viewport is determined by `JClass Chart 3D`. The user can modify this default by scaling it and translating it within the `JCChart3dArea`. `JCViewport` comprises these properties:

- `horizontalShift` – indicates horizontal shift as a multiple of the original viewport size.
- `verticalShift` – indicates vertical shift as a multiple of the original viewport size.
- `normalized` – indicates whether the viewport is normalized (that is, is the scale equal to 1 and the shift equal to 0).
- `previewMethod` – controls what is drawn as the user rotates scales, or translates the cube interactively; can either be `PREVIEW_CUBE` (displays a wireframe cube [default]) or `PREVIEW_FULL` (displays the entire surface).
- `scale` – zoom factor. A value less than 1 means a viewport smaller than the default. A value greater than 1 means a viewport greater than the default. The scale value must be between `MIN_VIEW_SCALE` and `MAX_VIEW_SCALE`.

Rotation

The rotate actions manipulate the `xRotation`, `yRotation`, and `zRotation` properties of the `JCView3d` class in various ways. The standard rotation (no selected axis) implements a track ball type rotation. This causes the chart to rotate in the direction of the mouse. You can also rotate about one of the three standard axes (X, Y, or Z). There is also one more type of rotation, called eye rotation, that allows you to rotate about a line from the center of the cube to the user's eye position.

7.1.2 **JCActionTable Class**

The `JCActionTable` class is responsible for keeping track of the mappings from a user input event to a `JClass Chart 3D` action. For example, an association can be made to map from the `Shift+left` mouse button set of events to the `Translate` action.

`JClass Chart 3D` has a rich set of predefined associations that handle scaling, translation, rotation, and many other functions. To enable the standard set of actions, add the following line of code to your program:

```
chart3d.getActionTable().addAllDefaultActions();
```

`JCActionTable` stores a set of mappings between an `ActionInitiator` and the name of a class that implements the `JCAction` interface. An `ActionInitiator` captures the essence of a user input event via its subclasses – `KeyActionInitiator` and `MouseActionInitiator`. A `MouseActionInitiator` defines the mouse button and

modifiers that the chart will recognize when the user presses a mouse button. For example, JClass Chart 3D's `Rotate` method is invoked when the user presses the left mouse button, while holding down the `Ctrl` key.

You can define a `MouseEventInitiator` for this combination:

```
MouseEventInitiator mai = new MouseEventInitiator(MouseEvent.  
    BUTTON1_MASK, InputEvent.CTRL_MASK);
```

You can associate the `Rotate` method with the initiator by specifying its pathname as a `String`. For users of the Java 2 version of JClass Chart 3D, the pathname is `com.klg.jclass.chart3d.j2d.actions.RotateAction`. For users of the Java 3D version of JClass Chart 3D, the pathname is `com.klg.jclass.chart3d.j3d.actions.RotateAction`.

Both of the above `RotateAction` classes implement the `JCAction` interface, with the most important methods being the `start()`, `animate()`, and `end()` methods. These correspond to the user's mouse button down, drag, and release events.

When JClass Chart 3D detects that the user has pressed the left mouse button, it will dynamically instantiate the specified class, and use it to handle the user's mouse events. To minimize memory usage, this class will be created only once and then be reused for subsequent interactions.

Adding and removing individual actions

JClass Chart 3D's `JCActionTable` defines all of the commonly used interactions, so individual actions can be added or removed, and custom actions can be added. For example, to add the above `Rotate` method, simply use:

```
JCActionTable at = chart3d.getActionTable();  
at.addAction(JCActionTable.DEFAULT_MOUSE_ROTATE_ANY_ACTION,  
    at.getDefaultRotateActionClass());
```

A `KeyActionInitiator` defines the `keyCode` and modifiers that JClass Chart 3D will recognize when the user presses a key. For example, if you want to detect when the user presses the `Shift-r` key combination, create a `KeyActionInitiator` with the following code:

```
KeyActionInitiator key = new KeyActionInitiator(KeyEvent.VK_R,  
    InputEvent.SHIFT_MASK);
```

To make this key combination invoke the `Reset` action, use:

```
JCActionTable at = chart3d.getActionTable();  
at.addAction(key, at.getDefaultResetActionClass());
```

Custom actions

Custom actions can be added by writing a class that implements the `JCAction` interface, and associating it with an `ActionInitiator` by using the `addAction()` method and specifying its name as a `String`. For example, if you wanted to associate the above key

with your own action handler that existed in `com.yourcompany.yourproduct.Reset` class, you would specify:

```
JCActionTable at = chart3d.getActionTable();  
at.addAction(key, "com.yourcompany.yourproduct.Reset");
```

If you later wanted to remove this specific action, use:

```
chart3d.getActionTable().removeAction(key);
```

To remove all actions, including default and custom actions, use:

```
chart3d.getActionTable().removeAllActions();
```

The full set of default actions and mouse and key combinations are described in the `JCActionTable` class description in the API Reference Javadocs, which are included when you purchase `JClass Chart 3D`.

For an overview of `JClass` actions, please review the following table. Note that except for the `Edit` action, these are the actions that get set on the chart when the `addAllDefaultActions` method is called.

Action Name	Default Binding		Description
	Mouse Button (MB)	Key	
Cancel	–	c	Cancel the current action.
Customize	MB3	Alt + Enter	Show the Customizer.
Edit	MB1	–	Edit the chart. Must be explicitly added. Overrides <code>Pick</code> .
Pick	MB1	–	Call the chart's <code>Pick</code> method.
Reset*	–	r	Read the viewport to default.
Rotate	Ctrl + MB1	Left, Right, Up, Down	Rotate using a trackball mechanism.
RotateEye*	Ctrl + MB1	–	Rotate about the eye.
RotateX*	Ctrl + MB1	–	Rotate constrained to an XAxis rotation.
RotateY*	Ctrl + MB1	–	Rotate constrained to a YAxis rotation.
RotateZ*	Ctrl + MB1	–	Rotate constrained to a ZAxis rotation.

Scale	Alt + MB1	Page Up, Page Down	Scale the chart interactively.
SwitchRotateAny*	–	n	Switch Rotate type to trackball rotation.
SwitchRotateEye*	–	e	Switch Rotate type to eye rotation.
SwitchRotateX*	–	x	Switch Rotate type to X-rotation.
SwitchRotateY*	–	y	Switch Rotate type to Y-rotation.
SwitchRotateZ*	–	z	Switch Rotate type to Z rotation.
Translate	Shift + MB1	Shift + Left, Shift + Right, Shift + Up, Shift + Down	Shift the plot cube left, right, up, down.
Zoom*	Ctrl + Shift + MB1	–	Scale using a rubber banded rectangle.

* Not available for the Java 3D API version of JClass Chart 3D

7.1.3 JCChart3dEvent class

The `JCChart3dEvent` class is used to encapsulate an action (for example, rotate, zoom, or scale) on a JClass Chart 3D. It holds the chart that was acted upon. The `JCChart3dEvent` class gets sent to all Chart3D listeners through the `changeChart()` method (see below).

7.2 Listeners

JClass Chart 3D uses the standard AWT/Swing event/listener mechanism to tell an application that certain events have happened on the chart. A listener is called when these events have completed.

There are three listeners in JClass Chart 3D, each of which is discussed below:

- data
- chart3d
- pick

Data Listener

The `Chart3dDataListener` interface is a template for event listener interfaces for chart data events. Its method, `chart3dDataChange`, is called whenever the JClass Chart 3D data

has changed; interested listeners should implement this method and register the object with a data source that implements the `Chart3dDataManager` interface.

Chart3d Listener

The `JCChart3dListener` interface is the event listener interface for `JClass Chart 3D` events. It has two methods. The `changeChart` method is called whenever the chart has been changed through a user action. The `sendEvent()` method of the `JCChart3d` class tells interested listeners that the chart has changed once the action is completed. Interested listeners should implement this method and register the object with `JClass Chart 3D`. The `paintChart` method is called whenever the paint method of the `JCChart3d` object is called.

Pick Listener

The `JCPick3dListener` interface is the event listener interface for chart pick events. It has a single method called `pick`, which is called whenever a `JClass Chart 3D` has been picked. Interested listeners should implement this method and register the object with `JClass Chart 3D`.

7.3 Mapping and Picking

Mapping

The `map()` method of the `Chart3dDataView` object takes (x, y) pixel coordinates and maps them to a point in data space. For grid data, this is a point interpolated from the nearest grid values. For point data, it is the nearest data point. If the pixel point is not within the chart, a point with its x , y , and z values set to `Double.MAX_VALUE` is returned. The user can also call the `Chart3dDataView`'s `coordToDataCoord()` method which is a wrapper around the `map()` method.

Unmapping

Unmapping is the opposite of mapping. It maps from data space coordinates to pixel coordinates. It basically applies the current data transformation that `JClass Chart 3D` uses to transform the data point to pixels. `Chart3dDataView`'s `dataCoordToCoord` method is a wrapper around its `unmap()` method.

JCData3dIndex class

The `JCData3dIndex` class contains a unique index to an object in `JClass Chart 3D` consisting of either an (x,y) grid data value, a (series, point) point data value, a label, or a contour range depending on whether the index is of type `JCData3dGridIndex`, `JCData3dPointIndex`, `JCData3dLabelIndex`, or `JCData3dContourIndex`, respectively. This class is used by the `JClass Chart 3D`'s `pick()` and `unpick()` methods and contains information related to these operations.

Picking

The `pick()` method of `JClass Chart 3D` takes an (x, y) pixel position and selects the internal component in which it is contained. This could be the header, the footer, the `chart3dArea`, or the legend. The object property of the returned `JCData3dIndex` indicates which component has been selected. If the pixel position is not contained in any of these components, `null` is returned. Note that the pixel position is assumed to be relative to the `JClass Chart 3D` component, not the internal components.

If the `chart3dArea` is selected, `pick` returns the index of the point closest to the specified pixel position via the returned `JCData3dIndex` object. This object also contains the distance from the pixel position to the selected point. If the pixel position misses the chart, the index returned will have each of its indices set to -1.

If a label is selected, `pick` returns a `JCData3dLabelIndex` which contains the internal value label of the selected label, as well as the index of that label within the internal value labels array. (See [Label Selection and Clustering](#), in Chapter 2 for more information.)

Note: Label selection and coloring is currently not available in the Java 3D API version of `JClass Chart 3D`.

If the legend is selected, `pick` returns a `JCData3dContourIndex` if contour level element appears in the legend (grid data only). For point data, a `JCData3dPointIndex` object with the selected series is returned.

`Chart3dDataView`'s `coordToDataIndex` method is a convenient wrapper for picking on the `chart3dArea`.

Unpicking

The `unpick` method is the opposite of `pick`. It determines the pixel position for a given `JCData3dGridIndex` or `JCData3dPointIndex` object (the type of the index needs to match the type of the current `elevationData`). `Chart3dDataView`'s `dataIndexToCoord` method is a convenient wrapper for the `unpick` method.

7.4 dragZValue Method

The `dragZValue` method in the `Chart3dDataView` class finds a new Z value for a given point index based on a given pixel position. Given a start point A (for grid data, the index refers to a point on the grid; for point data, it refers to one of the points in the list of series) and a point P on the screen, project the line AP (in 3D-space) onto the line through A parallel to the Z axis to find the Z value that corresponds to P on the projected line.

The `dragZValue` method returns the new computed z-value. This method takes several parameters:

- `data` – the data for which this operation is to take place (an instance of either `Chart3dGridData` or `Chart3dPointData`).

- `index` – the data index of the point; for grid data, this must be an instance of `JCData3dGridIndex`, which corresponds to an X- and Y-grid position specification, while for point data, this must be an instance of `JCData3dPointIndex` which corresponds to the series and point number of the point.
- both the X- and Y-value of the screen position (in relation to the chart component, not the `chart3dArea` component).

This procedure can be used to support the interactive modification of a grid or point value. For example, when the user clicks somewhere on the chart, it calls the `pick()` method of the `JCChart3d` object to determine the index of the closest point. Then as the mouse is dragged, new pixel values are passed along with the chosen index to `dragZValue()`, which returns the new Z value for the index. The new Z value can be set on the data source and the chart updated. The edit action contains a built-in implementation of this mechanism.

7.5 gridValue Method

The `gridValue` method in the `Chart3dDataView` class returns an estimate of the surface value at (x, y) calculated using bilinear interpolation. The method finds the four closest grid points, interpolates, and returns the estimated Z value. The parameters are:

- `data` – the internal data object for which this operation is to take place. The data must be grid data (an instance of `Chart3dGridData`).
- `x` – the data space X-value
- `y` – the data space Y-value.

Programming with the Java 3D API

Java 3D – Overview ■ *System Set-up* ■ *Browsers and Java 3D* ■ *Java 3D API SceneGraphObject class* ■ *Scene Graph Viewing Object Classes* ■ *BranchGroup and TransformGroup Rendering* ■ *Behaviors* ■ *Java 3D-Enabled Charting Features*

As noted in the Preface, JClass Chart 3D allows you to create stunning 3D graphics using either the Java 2 API or the Java 3D API. In this section of the JClass Chart 3D manual, we provide an overview of the Java 3D API, and then delve into leveraging the power of the Java 3D API with JClass Chart 3D.

For your reference, here is the URL of the Java 3D API:
<http://java.sun.com/products/java-media/3D/download.html>

8.1 Java 3D – Overview

Java 3D is a standard suite of classes that extend Java 2's core Java platform capabilities to add 3D graphics and sound capabilities to applets and applications. Featuring interactive 3D graphics, behaviors, and spatialized sound capabilities, Java 3D can be integrated with standard Java programs. As a layered API, the high-level Java 3D API sits atop low-level 3D graphics APIs such as OpenGL and Direct3D.

The Java 3D API provides high-level constructs for generating and manipulating 3D geometry, and provides structures for rendering this geometry. Java 3D provides functions for creating imagery, animations, and interactive 3D graphics application programs.

Java 3D programs can be written to run as stand-alone applications, as applets, or both. (For information on applets, please see Section 8.3, [Browsers and Java 3D](#).)

Please note that using Java 3D requires a sound knowledge of Java programming. Please see [Related Documents](#) in the [Preface](#) for helpful information.

8.1.1 Scene Graph Programming Model

Java 3D is based on a scene graph programming model. Here's a high level look at this model.

Java 3D developers use Java 3D classes to construct nodes. Nodes contain fields that a developer can manipulate in order to alter node properties. A collection of nodes is a scene graph. These nodes that make up a scene graph are rooted to a Locale object, which in turn is rooted to a VirtualUniverse object. A virtual universe, then, describes a 3D space populated with 3D objects.

Each scene graph has just one VirtualUniverse. Having only one instance of a VirtualUniverse in a Java 3D program is recommended. Also, while a VirtualUniverse object may reference many Locale objects, most Java 3D programs have just one Locale object.

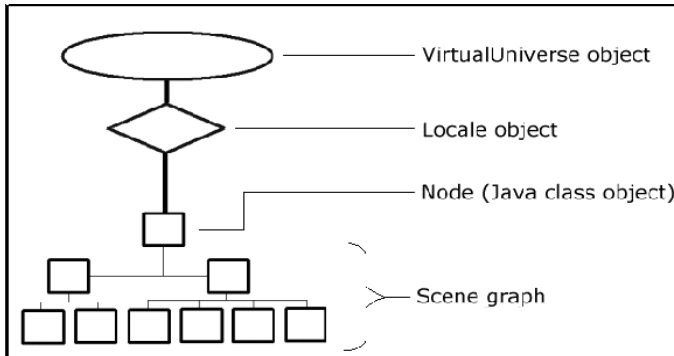


Figure 25 Basic scene graph programming model. Java 3D scene graphs are linked to a Locale object, which is attached to a VirtualUniverse object.

For each Java 3D scene object, transform, or behavior, a programmer needs to create a new object instance (that is, a new node), set the fields of this instance, and add the instance to the scene.

To summarize, the Java 3D program creates instances of Java 3D objects (called nodes) and places them into a scene graph data structure (an arrangement of 3D objects in a tree structure that completely specifies the content of a virtual universe, and how it is to be rendered).

8.2 System Set-up

Here's a checklist of what you need installed on your system in order to work with Java 3D.

- Java 2 Java Runtime Environment (JRE)

The Java 2 JRE can be installed by itself (visit Sun's site at <http://java.sun.com/j2se/1.3/jre/>) or you can install the Java 2 SDK, which contains the Java 2 JRE (visit Sun's site at <http://java.sun.com/j2se/>)

- Java 3D class libraries

You can download the Java 3D class libraries from Sun's site at <http://java.sun.com/products/java-media/3D/index.html>

- Java Development Kit (JDK)

Download JDK 1.3.1 and higher

- JAR files

After you install Java 3D, verify that you have the *vecmath.jar*, *j3dutils.jar* and *j3dcore.jar* in your */jre/lib/ext/* directory.

While not essential to running a Java 3D application, it is highly recommended that you install the Javadoc for the Java 3D API.

The Javadocs for the entire Java 3D API can be downloaded from <http://java.sun.com/products/java-media/3D/download.html>

Hardware acceleration

Java 3D supports performance-enhancing features (such as hardware acceleration) of the underlying platform. Because Java 3D – which is layered atop of your graphics API – is a high-level API, Java 3D shields developers from the platform-specific details. If hardware acceleration is available, the Java runtime system will harness it without any effort from the developer.

8.3 Browsers and Java 3D

As noted above, Java 3D requires a Java 2 JRE. Two popular browsers – Internet Explorer and Netscape Navigator – ship with versions of Java that are older than Java 2. This means that end-users using these standard browsers will not be able to view Java 3D applets (an applet is a Java program to be included in an HTML page).

Thus, in order to experience Java 3D applets through a Web browser, end-users must install a Java 2 JRE, followed by the Java 3D libraries. This means that Java 3D applets require that Sun's Java Plug-in be installed and activated. This plug-in, which redirects Java 3D applets from the browser's internal JRE to the Java 2 JRE, is automatically installed when the Java 2 JRE is installed.

8.4 Java 3D API

There are about 150 classes in the Java 3D API. These classes are organized into three key packages. Java 3D applets and applications are constructed using classes found in the

`javax.media.j3d`, `javax.vecmath`, and `com.sun.j3d` packages. The last package (`com.sun.j3d`) contains convenience classes.

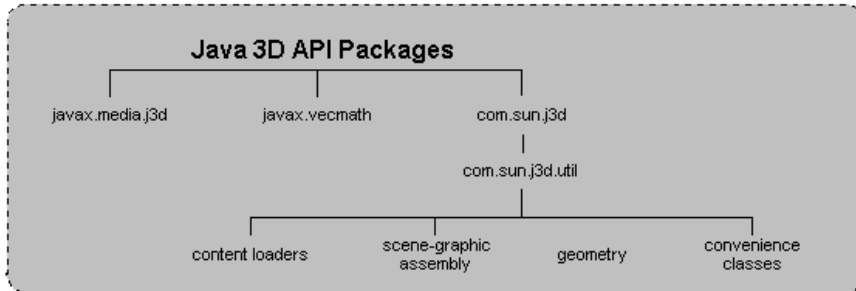


Figure 26 The Java 3D API package hierarchy.

As an aside, Java package names with a “`javax`” preface typically indicate an extension to the core Java 2 platform. For instance, the main Java 3D package is `javax.media.j3d`.

javax.media.j3d

This package is required in the Java 3D API. This package, with over 100 classes, provides the core functionality of Java 3D. Every Java 3D program is created using at least one class from this package.

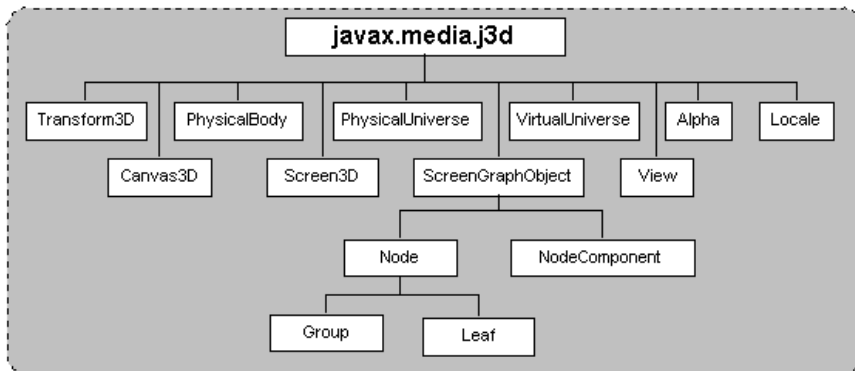


Figure 27 The `javax.media.j3d` hierarchy.

javax.vecmath

This package is required in the Java 3D API. This package defines vector mathematics classes for points, vectors, matrices, and other mathematical objects that are used in 3D object representation and manipulation. Many core Java 3D classes rely on classes in `javax.vecmath`.

Because the `javax.vecmath` package contains classes that are useful outside of Java 3D, they are packaged outside of the main `javax.media.j3d` package.

com.sun.j3d

This package is optional in the Java 3D API. This package contains convenience classes for Java 3D. The utility classes are placed into `com.sun.j3d.util` and are grouped into four categories: content loaders, scenegraph assembly aids, geometry classes, and convenience classes.

8.5 SceneGraphObject class

The `SceneGraphObject` class, the base class for nearly every object in a Java 3D scene graph, is an abstract class that defines a number of properties common to its two subclasses: `Node` and `NodeComponent`.

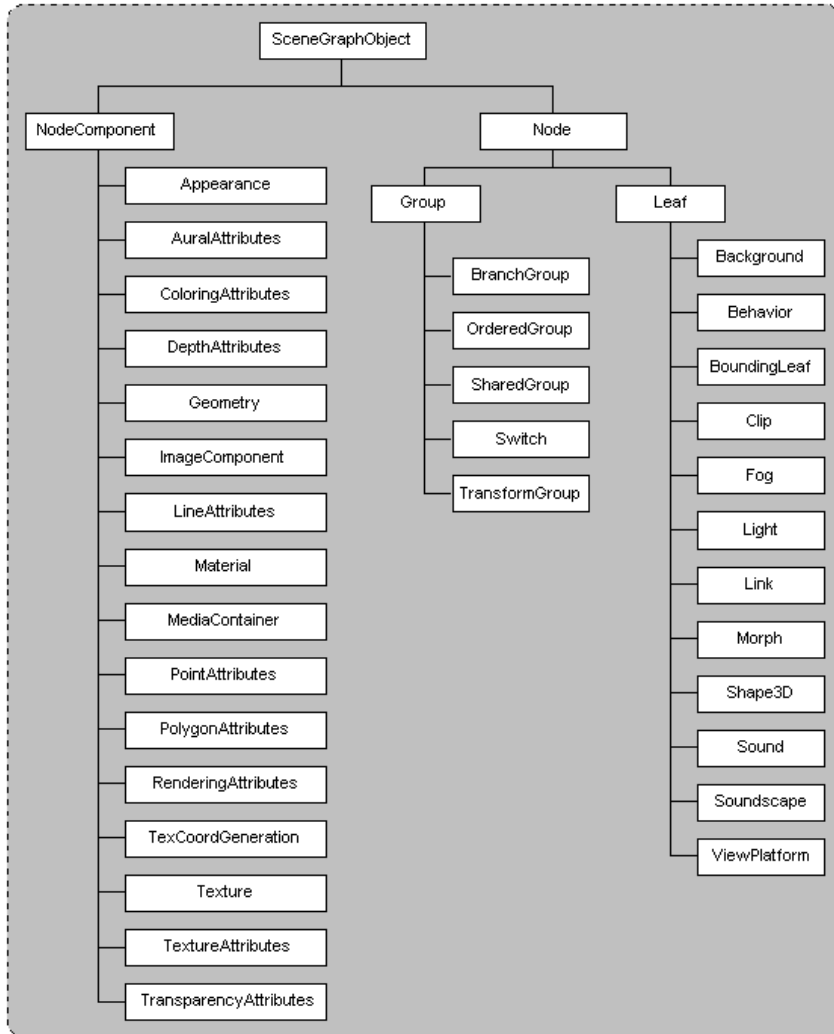


Figure 28 The `javax.media.j3d.SceneGraphObject` hierarchy.

8.5.1 Node class

The `Node` class is the abstract superclass of `Group` and `Leaf` classes. This `Node` class provides a template for scene graph objects, defining important common methods for its subclasses. The subclasses of `Node` provide most objects in the scene graph. A `Node` object is either a `Group` node or a `Leaf` node object. `Group` nodes can contain children, while `Leaf` nodes cannot. `Group` and `Leaf` are superclasses to several subclasses.

Group class

The `Group` class is the superclass to a family of classes that are used to specify the orientation and location of scene graph objects (visual objects in the virtual universe). As mentioned above, `Group` nodes can contain children. The role of a `Group` object is mainly to act as the parent of other `Group` nodes and `Leaf` nodes.

All subclasses of `Group` are considered to be grouping nodes.

Here are the subclasses of `Group`:

- `BranchGroup` – used to create `BranchGroup` objects, which serve as the root for individual scene graph branches (please see [BranchGroup](#) and [TransformGroup](#), in Chapter 8 for more information).
- `OrderedGroup` – used to ensure a specific rendering order.
- `SharedGroup` – allows sharing of a subgraph among different portions of a scene graph tree.
- `Switch` – allows the Java 3D program to choose which children will be rendered.
- `TransformGroup` – creates a `TransformGroup` object that generally is used to orient and position all children contained within it.

Two of these subclasses, `BranchGroup` and `TransformGroup`, are key to using `JClass Chart 3D`, so these will be expanded later on.

Leaf Class

The appearance, sound, and behavior of visual objects in the virtual universe are specified using subclasses of the `Leaf` class. Some of the subclasses of `Leaf` are `Background`, `Behavior`, `Fog`, `Light`, `Shape3D`, and `Sound`. `Leaf` nodes cannot contain children, but may reference `NodeComponents`.

Thus, `Leaf` nodes specify the shape, sound, and behavior of scene graph objects. As well, `Leaf` nodes provide a view platform that is used by the virtual universe to position and orient a view of the scene.

Here's a list of the top-level `Leaf` subclasses:

- `Background` – defines the background (for example, solid color or an image) that fills the window when a new frame is rendered.
- `Behavior` – abstract class that defines properties common to all Java 3D components that can modify a scene graph at runtime.

- **BoundingLeaf** – defines a bounding region object that can be referenced by other Leaf nodes to define a region of influence (for Fog and Light nodes), an activation region (Background, Clip, and Soundscape nodes), or a scheduling region (Sound and Behavior nodes).
- **Clip** – specifies the back, or far, clip distance used to clip objects in the virtual universe.
- **Fog** – abstract class that outlines a set of attributes common to fog environmental effects, as well as the region of influence for the Fog node.
- **Light** – abstract class that defines properties common to all lights.
- **Link** – allows an application to reference a shared graph, rooted by a SharedGroup node, from within a branch graph or another shared graph.
- **Morph** – allows a Java 3D program to morph between multiple GeometryArray objects.
- **Shape3D** – specifies all geometric objects; contains a list of one or more Geometry component objects (these define the shape node's geometric data) and a single Appearance component object (specifies that object's appearance attributes, including color, material, and transparency).
- **Sound** – abstract class that defines properties common to all sound sources.
- **Soundscape** – defines characteristics of the listener's environment as it pertains to sound.

- `ViewPlatform` – controls the position, orientation, and scale of the viewer; a viewer navigates through the virtual universe by changing the transform in the scene graph hierarchy above the `ViewPlatform`.

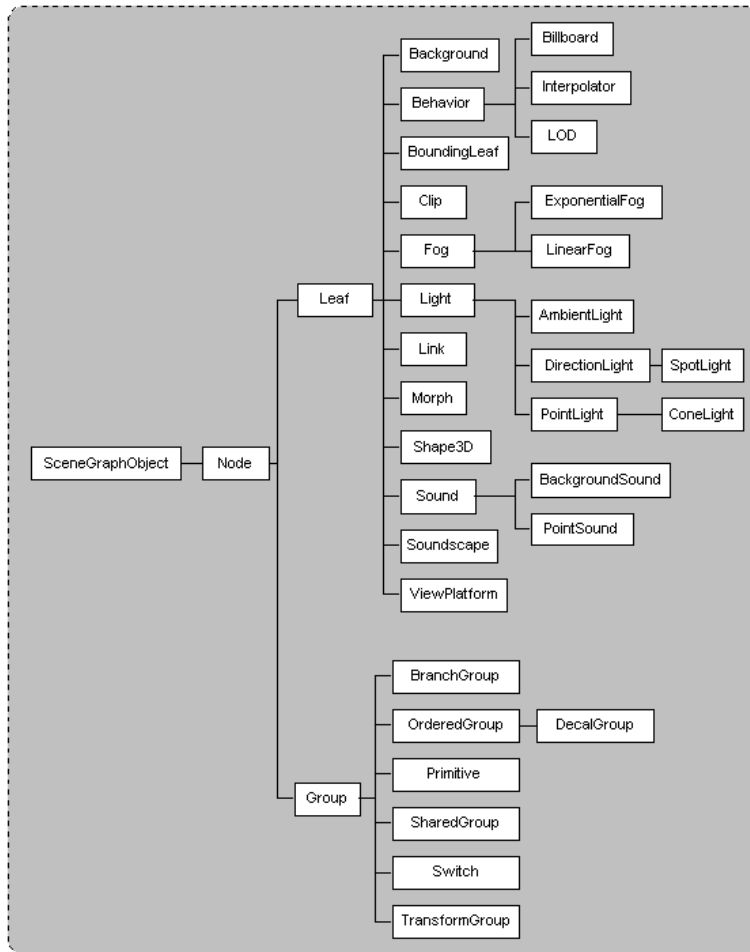


Figure 29 SceneObjectGraph hierarchy showing subclasses of Leaf and Group.

8.5.2 NodeComponent Class

The `NodeComponent` class is the superclass used to specify the geometry, appearance, texture, and material properties of a `Shape3D` (Leaf) node. `NodeComponent`s are not part of the scene graph, but are referenced by it. A `NodeComponent` may be referenced by more than one `Shape3D` object.

Following is a list of the classes that are direct children of the `NodeComponent` abstract parent class:

- `Alpha` – provides common methods for converting a time value into an alpha value.
- `Appearance` – creates objects that are a component of a `Shape3D` node; appearance nodes define all rendering states of a `Shape3D` node (coloring attributes, line attributes, point attributes, polygon attributes, rendering attributes, transparency attributes, material, texture, texture attributes, texture coordinate generation, and texture unit state).
- `AuralAttributes` – defines environmental audio parameters that affect sound rendering, such as atmospheric rolloff; this class creates objects that are a component object of a `Soundscape` node.
- `ColoringAttributes` – defines attributes used in color selection and shading model.
- `DepthComponent` – abstract base class that defines a 2D array of depth (*Z*) values.
- `Geometry` – abstract class that specifies the geometry component information required by a `Shape3D` node.
- `ImageComponent` – abstract class that defines 2D or 3D `ImageComponent` classes used in a Java 3D scene graph; used for texture images, background images, and raster components of `Shape3D` nodes.
- `LineAttributes` – defines all rendering states that can be set as a component object of a `Shape3D` node, such as line pattern.
- `Material` – creates objects that are a component of an `Appearance` object; `Material` objects define the appearance of an object under illumination.
- `MediaContainer` – creates objects that are components of a `Sound` node (these objects define audio properties associated with a `Sound` node).
- `PointAttributes` – creates objects that define all attributes that apply to point primitives, such as point size and antialiasing.
- `PolygonAttributes` – generates objects that define the rendering properties of polygon primitives, such as rasterization mode.
- `RenderingAttributes` – defines common rendering attributes for all primitive types.
- `TexCoordGeneration` – contains all parameters needed for automatic texture coordinate generation; is included as part of an `Appearance` component object.
- `Texture` – an abstract class that defines the texture properties used when texture mapping; as an abstract class, all texture objects must be created as either a `Texture2D` object or a `Texture3D` object (both are subclasses of `Texture`).
- `TextureAttributes` – used to define texture-mapping attributes, such as texture mode, blend color, and perspective correction.
- `TextureUnitState` – defines all texture mapping state for a single texture unit; is an appearance object that contains an array of texture unit state objects to define the state for multiple texture mapping units.

- `TransparencyAttributes` – describes all attributes affecting object transparency.

8.6 Scene Graph Viewing Object Classes

The Java 3D API includes five classes that are used to view scene graphs:

- `View` – contains all parameters needed in rendering a 3D scene from one viewpoint (note that all Java 3D viewing parameters are contained directly within the `View` object or from within objects referenced by it); a `View` object contains a list of `Canvas3D` objects that the view is rendered into, as well as contains a reference to a `PhysicalBody` and a `PhysicalEnvironment` object.
- `PhysicalBody` – contains a specification of the user's head; attributes of this object are defined in the head coordinate system.
- `PhysicalUniverse` – contains specification of the physical environment in which the view will be generated; is used to set up input devices (sensors) for head-tracking and other uses, and the audio output device.
- `Canvas3D` – provides a drawing canvas for 3D rendering (either on-screen or off-screen rendering); is an extension of Java's AWT `Canvas` class that can be subclassed to implement additional functionality.
- `Screen3D` – encompasses all information about a particular screen or display device, such as the height and width of a screen.

These classes, while not strictly part of the Java 3D scene graph, define important viewing parameters for Java 3D programs, plus they provide a way for users to interact with the program.

8.7 BranchGroup and TransformGroup

Recall that the `Group` class is the superclass that is used to specify the orientation and location of scene graph objects. Two of its subclasses, `BranchGroup` and `TransformGroup`, are key to using JClass Chart 3D.

BranchGroup

The `BranchGroup` node is used to construct branches of a Java 3D scene graph by acting as the root for the subgraph (called branch graph). A branch graph contains the various nodes that make up a scene graph. To create a branch graph, a developer constructs a `BranchGroup` object, then constructs the nodes that it will contain. The nodes subsequently are added to the `BranchGroup`. `BranchGroups` are generally attached to a `Locale`, which acts as an anchor for objects in a scene branch graph (only `BranchGroup` objects can be attached to a `Locale`). A `Locale`, in turn, is attached to a `VirtualUniverse` object, which is the top-level object in every Java 3D scene graph.

BranchGroups can be selectively attached and detached from the scene graph. Once a branch graph is inserted into a Locale, each object in that branch graph becomes live. Once objects are live, they are subject to being rendered. Also, the parameters of live objects cannot be modified unless the corresponding capability has been specifically set before the object became live. By attaching and detaching branch graphs, a Java 3D developer can control when specific portions of a graph scene are rendered.

Capability bits (or “capabilities of the object”) are a list of parameters that determine which properties of an object are changeable after that object is made live. Capability bits must be set before the object is compiled or made live.

Each SceneGraphObject has a suite of capability bits, which varies by class.

A `RestrictedAccessException` is thrown when an attempt is made to set the capability bits of an object that is part of a live or a compiled scene graph.

Java 3D programs usually comprise two `BranchGroup` objects: the view branch graph and the content branch graph. The view branch graph outlines the viewing parameters, such as the viewing location and direction. The content branch graph specifies the contents of the virtual universe: appearance, behavior, geometry, lights, location, and sound. Together, the two branches stipulate much of the work the renderer has to do.

`BranchGroup` objects can be compiled. Compiling a `BranchGroup` converts the internal representation of the `BranchGroup` object and all of its ancestors to a more efficient form for the rendering engine. Compiling `BranchGroup` objects is recommended as the last step before making it live.

TransformGroup Class

The `TransformGroup` class is a subclass of the `Group` class. `TransformGroup` objects hold geometric transformations such as translation and rotation. A `TransformGroup` node

specifies the position (relative to the Locale), orientation, and scale of the geometric objects in the virtual universe.

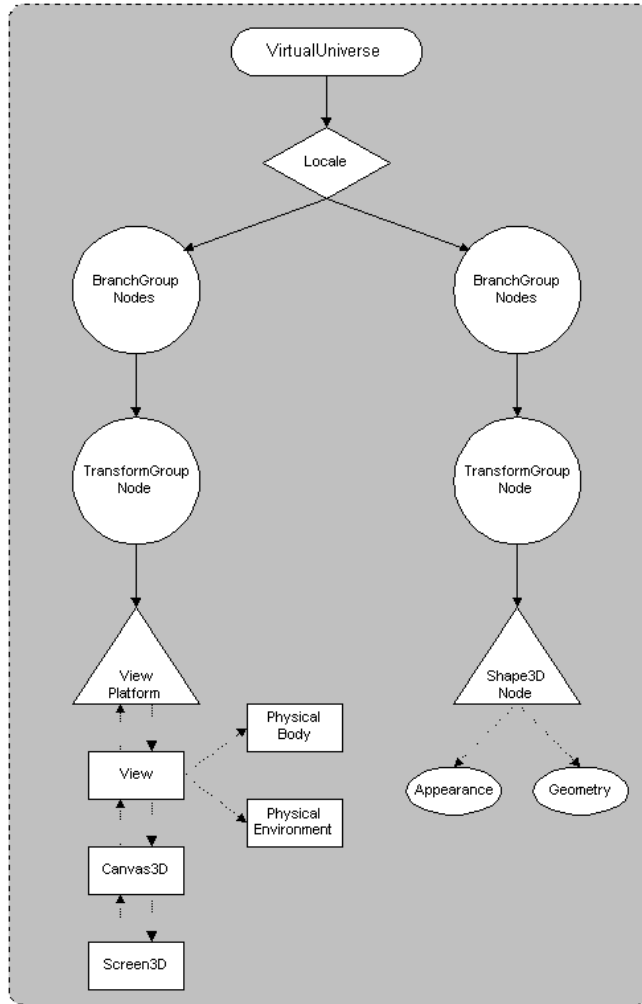


Figure 30 Basic Virtual Universe hierarchy diagram.

In the figure above, the Shape3D node references, for example, Appearance and Geometry node components. The Geometry object describes the geometric shape of a 3D object, while the Appearance object describes the appearance of the geometry, such as color, transparency, and other rendering attributes.

Again, in the figure above, `TransformGroup` specifies the position (relative to the `Locale`), orientation, and scale of `ViewPlatform`. `ViewPlatform` defines the end-user's view within the virtual universe.

8.8 Rendering

The Java 3D renderer traverses a Java 3D scene graph and displays its visible geometry in an on-screen window. In addition to drawing visible geometry, the Java 3D renderer also processes user input and performs behaviors.

The Java 3D API supports three rendering modes. The modes differ in the amount of control that the developer has over the rendering process, and that Java 3D has over optimizing rendering:

- immediate mode
- retained mode
- compiled-retained mode

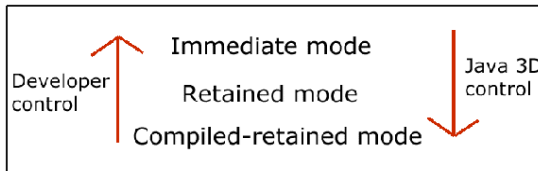


Figure 31 Rendering modes.

Immediate mode

The immediate mode offers the developer complete rendering control, which means that Java 3D has little opportunity to optimize rendering.

An application must provide a Java 3D `draw()` method with a complete set of points, lines, or triangles.

Retained mode

This mode balances the amount of rendering control between the developer and Java 3D. Retained mode requires that the developer construct a scene graph and specify the parts of the scene graph that may change during rendering. Scene graphs rendered in this mode allow the developer to add, delete, or modify nodes. The scene information is stored in a scene graph structure, so Java 3D is able to perform rendering optimizations in an effort to increase performance.

Compiled-retained mode

Similar to retained mode, compiled-retained mode requires that the developer construct a scene graph and specify the parts of the scene graph that may change in the rendering process.

A developer can compile individual Java 3D objects, and can specify portions of the scene graph for Java 3D to compile. While compiled entities are similar to noncompiled counterparts, compiled scene graphs and objects are stored in an internal format that is optimized for rendering.

Please note that once scene graphs and objects have been compiled, however, developers will have limited access to the internal structure.

8.9 Behaviors

To support interactivity, Java 3D allows developers the opportunity to create customized behaviors for objects in a virtual universe, embedding the logic into the scene graph so that an object can change in response to specific input or a stimulus.

Behavior node

A Behavior node can be added or removed from a scene graph. Every Behavior node contains a scheduling region that defines a spatial volume for enabling node scheduling.

A Behavior node contains an `initialization()` method that initializes the internal state of the behavior and specifies one or more wakeup conditions. The `initialization()` method is called when the `BranchGroup` object containing the behavior is added to the virtual universe.

As well, all Behavior nodes contain a `processStimulation()` method, which is used to receive and process stimulation. A behavior's `processStimulation()` method is invoked:

- when the node is active; and
- when one of its wakeup criteria is honored

8.10 Java 3D-Enabled Charting Features

8.10.1 Texture Mapping

For more control over the output, the Java 3D implementation of `JClass Chart 3D` provides you with methods which allow you to apply an image to the walls of the chart plot cube. These easy methods enable you to add texture and graphics to objects, allowing an enhanced artistic representation.

The first step in applying a Texture Map is to ensure that the image you would like to use is a Java 3D Texture object. The `JCTexture2D` class will provide you with many static factory methods to convert your image into a valid `Texture2D` object. Please note that because OpenGL imposes limits on the size of a `Texture2D` object, the factory methods in `JCTexture2D` will help you create a valid `Texture2D` object using an arbitrarily sized image.

Note: If the dimension of your source image is not a power of 2 (for instance, 512 or 1024), `JCTexture2D.createTexture(String filename, Color fill)` will center the image specified by `filename` in a larger image that is dimensionally correct. Any empty space around the original image will be filled with the color specified by the specified `Color` argument `fill`.

For more control over the alignment of the image within the `Texture` object use the following factory method:

```
JCTexture.createTexture(String filename, boolean square,
                       int hAlignment, int vAlignment, Color fill)
```

where `hAlignment` is one of `SwingConstants.LEFT`, `SwingConstants.CENTER`, or `SwingConstants.RIGHT`, `vAlignment` is one of `SwingConstants.TOP`, `SwingConstants.CENTER`, or `SwingConstants.BOTTOM`, and the Boolean `square` argument is used to force the final `Texture` object to be square if set to `true`. Please note that the `filename` argument must point to a valid image file which can be loaded by the Java AWT Toolkit, or by Java Advanced Imaging (JAI) if it is installed. Please refer to the respective APIs to find out which formats are valid.

For even more control over the construction of a `Texture2D` object, you can use any of the factory methods found in the `TextureLoader` class, or manipulate them yourself. Please refer to the Java 3D API for the `Texture2D` class for further details on constructing a `Texture2D` object, and for further details on the `Texture2D` object itself.

To apply your `Texture2D` object to the cube walls, use one of the `JCPlotCube` methods, such as `setWallTexture(int face, Texture2D texture, double angle)` or `setTexture(Texture2D texture)`.

8.10.2 Lighting

In order to create a virtual universe that has a realistic appearance, one must consider the lighting in the environment. The way that the light influences the appearance of objects is essential to creating the view you want to achieve. Java 3D gives `JClass Chart 3D` the power of selecting among four types of lighting to use in your output:

- ambient light
- directional light
- point light
- spot light

Note: Lighting can become a resource-intensive feature for browsers. It is recommended that you do not overuse the amount of light sources you have in your 3D virtual universe.

Ambient Light

Ambient light is uniform light, and thus produces uniform shade. Ambient light is intended as fill light in the scene where other sources do not light.

```
// AmbientLight Code Example
chart3d = new JCChart3dJava3d();
    :
    :
    :
JCChart3dAreaX area = (JCChart3dAreaX)(chart3d.getChart3dArea());
AmbientLight light = new AmbientLight();
light.setColor(new Color3f(java.awt.Color.yellow));
light.setEnable(true);
area.addLighting(light);
```

Directional Light

Directional light is representational of the sun in our universe. In the virtual universe, however, the light has no source, only parallel rays that all approach from the same direction. A directional light source is very useful when creating an environment that requires proper lighting, without much consideration to the actual source. Directional light can either be on or off. Because directional light is not ambient light, thus it does not degrade in any way, that is, it will produce a uniform shade.

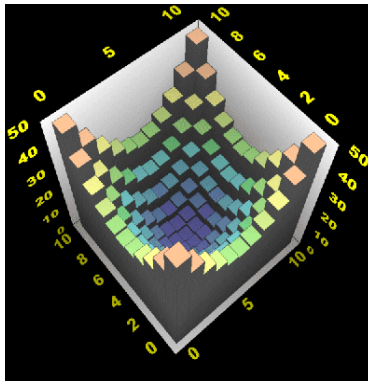


Figure 32 The effect of directional light on a cube.

Although you can manipulate the color and intensity of directional light, this type of lighting does not allow much other control. By setting the `On` field to `true`, the directional light will have been turned on.

```
// DirectionalLight Code Example
chart3d = new JCChart3dJava3d();
    .
    .
    .
JCChart3dAreaX area = (JCChart3dAreaX)(chart3d.getChart3dArea());
DirectionalLight light = new DirectionalLight();
light.setColor(new Color3f(java.awt.Color.yellow));
light.setDirection(new Vector3f(0.0f, -1.0f, 0.0f));
light.setEnable(true);
area.addLighting(light);
```

Point Light

A point light is a light source representative of a light bulb, where the light is emitted from one location in a radial pattern. One can select the source location of this type of light, as well as customizing the color and intensity.

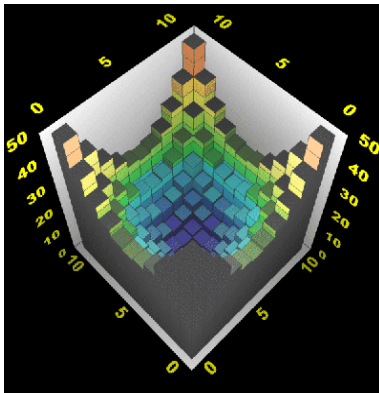


Figure 33 The effect of point light on a cube.

Point light also has the added capacities of attenuation and ambient intensity. Attenuation is the reproduction of “light attenuation”, the process by which light tapers off as it progresses in the distance. By using this feature, the scene will appear more realistic.

```
// PointLight Code Example
```

```
chart3d = new JCChart3dJava3d();  
    :  
    :  
    :  
JCChart3dAreaX area = (JCChart3dAreaX)(chart3d.getChart3dArea());  
PointLight light = new PointLight();  
light.setColor(new Color3f(java.awt.Color.yellow));  
light.setPosition(new Point3f(0.0f, 2.0f, 0.0f));  
light.setAttenuation(new Point3f(1.0f, 0.0f, 0.0f));  
light.setEnabled(true);  
area.addLighting(light);
```

Spot Light

Similar to a spot light in the real world, the spot light node is used to create light that travels in one specified direction in what appears to be a cone. This “light cone” forces the light to concentrate upon specific locations, leaving the rest of the environment in the darkness.

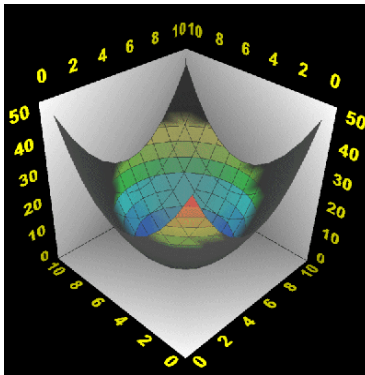


Figure 34 The effect of spot light on a cube.

Spot light is the most complex light source, in that there are several fields that must be specified in order to create ideal lighting. These include location, concentration, and attenuation, along with direction, beamWidth and spreadAngle.

```
// SpotLight Code Example

chart3d = new JCChart3dJava3d();
    :
    :
JCChart3dAreaX area = (JCChart3dAreaX)(chart3d.getChart3dArea());
SpotLight light = new SpotLight();
light.setColor(new Color3f(java.awt.Color.yellow));
light.setPosition(new Point3f(0.0f, 2.0f, 0.0f));
light.setAttenuation(new Point3f(1.0f, 0.0f, 0.0f));
light.setDirection(new Vector3f(0.0f, -1.0f, 0.0f));
light.setSpreadAngle((float)(Math.PI/2.0));
light.setConcentration(1.0f);
light.setEnable(true);
area.addLighting(light);
```

Since the spot light does not emit light in a radial range, it is necessary to direct the light to the appropriate location. Do this by customizing the direction field, knowing that the location field specifies the tip of the light, and the direction specifies its final destination.

The next two fields, concentration and spreadAngle, deal with the “light cone” that is emitted. In other words, it will indicate how large the light source will be when it hits its final destination, at full intensity.

8.10.3 Depth Cue

JClass Chart 3D has the capability, when using the Java 3D implementation, to add a fog-like appearance to the charts you produce. Hence, you can create visual effects to simulate haze, mist, smoke, or pollution, blurring the appearance of all objects to which the effect is applied.

The Fog node can be used to enhance the appearance of a JClass Chart 3D object, making it appear more realistic. Based on distance from the viewer, Java 3D will blend the fog color with objects in the scene – objects that are the furthest from the viewpoint will be the most blended. Typically the Fog node is used to make objects in the distance blur, although the opposite effect is possible.

It is suggested that one uses the same color for both the fog color and JCChart3dArea background color, forcing objects that are completely obscured by the fog to blend into the background.

To calculate the color of the fog applied to an object, the following equation is used:

$$\text{foggedColor} = \text{fogFactor} * \text{originalColor} + (1 - \text{fogFactor}) * \text{fogColor}$$

where the `fogFactor` is some function of the distance the viewer is from the object (Z-depth). The exact relationship depends upon the type of fog used in the virtual universe – either `LinearFog` or `ExponentialFog`.

Fog

The Fog leaf node defines a set of fog parameters common to all types of fog. These parameters include the fog color and a region of influence in which this Fog node is active. A Fog node also contains a list of Group nodes that specifies the hierarchical scope of this Fog. If the scope list is empty, then the Fog node has universe scope: all nodes within the region of influence are affected by this Fog node. If the scope list is not empty, then only those Leaf nodes under the Group nodes in the scope list are affected by this Fog node (subject to the influencing bounds).

If the regions of influence of multiple Fog nodes overlap, Java 3D will choose a single set of fog parameters for those objects that lie in the intersection. This is done in an implementation-dependent manner, but in general, the Fog node that is closest to the object is chosen.

Fog is an abstract class with two subclasses: `LinearFog` and `ExponentialFog`.

LinearFog

`LinearFog` is a Leaf node that defines the parameters of fog distance for a linear fog. `LinearFog` extends the Fog node by adding a pair of distance values, in Z, at which the fog should start obscuring the scene and should maximally obscure the scene. The front and

back fog distances are defined in the local coordinate system of the node, but the actual fog equation will ideally take place in eye coordinates.

```
// LinearFog Code Example
```

```
chart3d = new JCChart3dJava3d();  
:  
:  
:  
JCChart3dAreaX area = (JCChart3dAreaX)(chart3d.getChart3dArea());  
LinearFog fog = new LinearFog();  
fog.setColor(new Color3f(java.awt.Color.black));  
fog.setFrontDistance(4.5f);  
fog.setBackDistance(7.0f);  
area.addFog(fog);
```

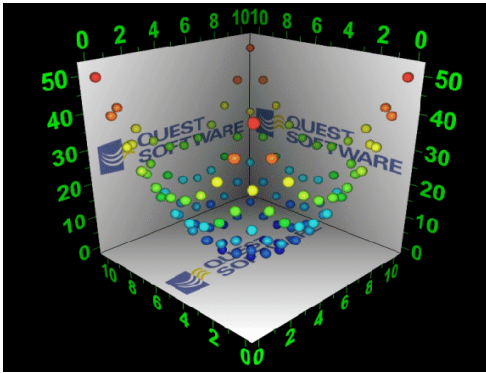


Figure 35 The effect of LinearFog on a cube.

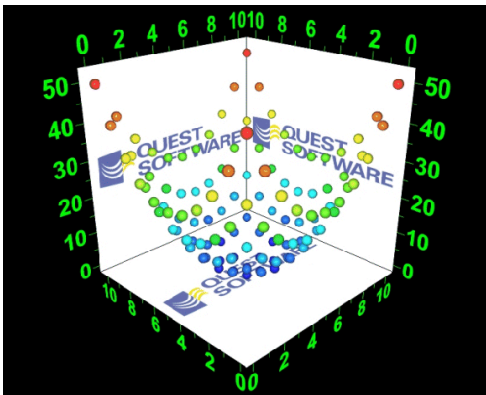


Figure 36 The effect of the Fog turned off.

When using `LinearFog`, the `fogFactor` is directly proportional to the Z-depth (distance of the viewer to the object). The `fogFactor` can be calculated using the following expression:

$$\text{fogFactor} = (\text{backDistance} - z) / (\text{backDistance} - \text{frontDistance})$$

where `z` is the distance from the viewpoint, `backDistance` is the Z-depth at which all objects will be completely obscured by the fog, and `frontDistance` is the Z-depth at which fog begins to take effect.

ExponentialFog

The `ExponentialFog` leaf node extends the `Fog` node by adding fog density. This density is a value created through an exponent, based on distance, of the fogging. This creates the effect of a nearby object having no fog, and the more distant objects having an exponential amount of fog.

In addition to specifying the fog density, `ExponentialFog` lets you specify the fog color, which is represented by R, G, and B color values, where a color of (0,0,0) represents black and (1,1,1) represents white.

`ExponentialFog` increases the `fogFactor` exponentially as the Z-depth of the object increases, following the preceding expression:

$$\text{fogFactor} = e^{-(\text{density} * z)}$$

where `z` is the distance from the viewpoint and `density` is the density of the fog.

```
// ExponentialFog Code Example
```

```
chart3d = new JCChart3dJava3d();
    .
    .
    .
JCChart3dAreaX area = (JCChart3dAreaX)(chart3d.getChart3dArea());
ExponentialFog fog = new ExponentialFog();
fog.setColor(new Color3f(java.awt.Color.black));
fog.setDensity(1.0f);
area.addFog(fog);
```


Part

II

*Reference
Appendices*

Appendix A

Interface Listing

Interface Summary

This appendix summarizes the commonly used JClass Chart 3D 3D interfaces, in alphabetical order.

A.1 Interface Summary

Name	Description
Chart3dDataListener	A template for event listener interfaces for chart data events.
Chart3dDataModel	The core data model interface for JClass Chart 3D.
Chart3dGridDataModel	The core grid data model interface for JClass Chart 3D.
Chart3dPointDataModel	The core point data model interface for JClass Chart 3D.
Editable3dDataModel	Indicates to interested classes that this datamodel is editable.
HoleValueChart3dDataModel	An interface used to specify hole values for a JClass Chart 3D data model.
JCChart3dListener	Event listener interface for chart events.
JCContourMapping	Allows the user to change the default contour level to contour style mapping.
LabelledChart3dGridDataModel	An interface used to specify X- labels and Y- labels for a JClass Chart3dGrid data model.
LabelledChart3dPointDataModel	An interface used to specify series labels for a JClass Chart3dPoint data model.

Appendix B

Object Property Listing

[Chart3D](#) ■ [Chart3d.Event](#) ■ [Chart3d.j2d](#) ■ [Chart3d.j2d](#)

This appendix summarizes the JClass Chart 3D 3D properties for all commonly used classes, in alphabetical order.

B.1 Chart3D

B.1.1 Chart3dData

Name	Description
DataOK	The DataOK property asks: Is the data passed in through the data source in a state in which it can be drawn?
DataSource	The DataSource property represents the data source for the internal data object.
HoleValue	The HoleValue property holds a special data value which determines where holes are drawn.
Name	The Name property, which is optional, holds the name of this data source.

B.1.2 Chart3dDataView

Name	Description
ChartType	The ChartType property holds the chart type of this dataView.
Contour	The Contour property references the JCContour object, which handles contouring and zoning.
Elevation	The Elevation property controls the JCElevation object, which determines meshing, shading, and transparency.

Name	Description
GridColors	The GridColors property allows certain facets or bars to have different colors.
Name	The Name property represents the name of this dataView.
ZoneData	The ZoneData property represents the internal zone data object.
ZoneDataSource	The ZoneDataSource property represents the zone data source for this dataView.

B.1.3 Chart3dGridData

Name	Description
ChartStyle	The ChartStyle property contains the chart style used if the chart type is a scatter plot; contoured and zoned are false.
NumX	The NumX property represents the number of X- grid values used. This is a read-only property.
NumY	The NumY property represents the number of Y-grid values used. This is a read-only property.
xGrid	The xGrid property represents the array of X- grid values. This is a read-only property.
xLabels	The xLabels property represents the X- data labels for this gridData object.
yGrid	The yGrid property represents the array of Y-grid values. This is a read-only property.
yLabels	The yLabels property represents the Y-data labels for this gridData object.
ZValues	The ZValues property represents the grid of Z values. This is a read-only property.

B.1.4 Chart3dPointData

Name	Description
NumSeries	The NumSeries property determines the number of series. This is a read-only property.

Name	Description
Series	The <code>Series</code> property contains the list of series for this point data object. This is a read-only property.

B.1.5 Chart3dPointSeries

Name	Description
ChartStyle	The <code>ChartStyle</code> property contains the chart style for this series.
Label	The <code>Label</code> property controls the data label for this series and is used in the legend.
NumPoints	The <code>NumPoints</code> property represents the number of points for this series. This is a read-only property.
Points	The <code>Points</code> property represents the array of points for this series. This is a read-only property.

B.1.6 JCAxis

Name	Description
AnnoFont	The <code>AnnoFont</code> property represents the annotation font and size for this axis.
AnnoFontCubeSize	The <code>AnnoFontCubeSize</code> property represents the annotation font cube size for this axis (this size is measured in thousandths of the unit cube size and must be between 0 and 1000).
AnnotationMethod	The <code>AnnotationMethod</code> property determines the annotation method.
AxisId	The <code>AxisId</code> property determines the axis id number. Usually one of <code>AXIS_X</code> , <code>AXIS_Y</code> , or <code>AXIS_Z</code> . You can only set this property on creation.
GridLines	The <code>GridLines</code> property determines the gridlines on a per plane basis for this axis.
LabelGenerator	The <code>LabelGenerator</code> property holds a reference to an object that implements the <code>JCLabelGenerator</code> interface. This interface is used to externally generate labels if the <code>AnnotationMethod</code> property is set to <code>JCAxis.VALUE</code> . Default value is <code>null</code> .

Name	Description
Max	The Max property controls the axis maximum value.
MaxIsDefault	The MaxIsDefault property determines whether Chart3d is responsible for calculating the maximum axis values. If true, Chart3d calculates the axis max. If false, Chart3d uses the provided axis max.
Min	The Min property controls the axis minimum value.
MinIsDefault	The MinIsDefault property determines whether Chart3d is responsible for calculating the minimum axis value. If true, Chart3d will calculate the axis min. If false, Chart3d will use the provided axis min.
Origin	The Origin property is used for Z axis only (for bars and scatter plot drop lines).
Showing	The Showing property asks: Is the axis showing?
Title	The Title property controls the axis title.
TitleFont	The TitleFont property controls the title font and size for this axis.
TitleFontCubeSize	The TitleFontCubeSize property controls the title font cube size for this axis (this size is measured in thousandths of the unit cube size and must be between 0 and 1000).
ValueLabels	The ValueLabels property is an indexed property containing a list of all annotation specified by the user for an axis.

B.1.7 JCBar

Name	Description
xFormat	The xFormat property represents the X- bar format.
xSpacing	The xSpacing property represents the X- bar spacing.
yFormat	The yFormat property represents the Y-bar format.
ySpacing	The ySpacing property represents the Y-bar spacing.

B.1.8 JCChart3d

Name	Description
About	The About property displays contact information for Quest Software in the bean box.
AllowUserChanges	The AllowUserChanges property determines whether the user viewing the chart can modify chart values. Used to allow edits to values and changes to parameters via the Customizer.
Batched	The Batched property controls whether chart updates are accumulated.
CancelKey	The CancelKey property specifies the key used to cancel the current action.
Chart3dArea	The Chart3dArea property controls the component that manages the area where the chart is drawn.
CustomizerName	The CustomizerName property specifies the name of the Customizer (used for instantiation).
DataView	The DataView property contains a list of dataViews for this chart.
Footer	The Footer property controls the footer for this chart.
FooterLayoutHints	The FooterLayoutHints property is used to give the layout manager information about the position and size of the footer.
Header	The Header property controls the header for this chart.
HeaderLayoutHints	The HeaderLayoutHints property is used to give the layout manager information about the position and size of the header.
Legend	The Legend property controls the legend for this chart.
LegendLayoutHints	The LegendLayoutHints property is used to give the layout manager information about the position and size of the legend.
LegendManager	The LegendManager property controls the default implementation of the legend populator and renderer.
Pick3dListener	The Pick3dListener property represents the current list of JCPick3dListener's for this chart.
ResetKey	The ResetKey property specifies the key used to reset the drawing viewport to its default value.
WarningDialog	The WarningDialog property controls whether a dialog will appear when the chart has warning messages.

B.1.9 JCChart3dArea

Name	Description
ActionHandler	The ActionHandler property is the handler that's handling any action in progress.
Axes	The Axes property contains a list of the X, Y, and Z axis objects.
Bar	The Bar property represents the object that controls bar chart only properties.
InAction	The InAction property asks: Are we currently in an action?
PlotCube	The PlotCube property controls the object that controls properties of the plot cube.
PreferredSize	The PreferredSize property asks: What is our preferred size? If null, a default size is used.
Scatter	The Scatter property controls the object that controls scatter plot-only properties.
Surface	The Surface property controls the object that controls surface-only properties.
View3d	The View3d property represents an object that controls the X, Y, Z rotation.
Viewport	The Viewport property represents an object that allows the user to control the drawing viewport.
xAxis	The xAxis represents the axis in the X- direction.
yAxis	The yAxis represents the axis in the Y-direction.
zAxis	The zAxis represents the axis in the Z direction.

B.1.10 JCChart3dLegend

Name	Description
ContinuousLayout	The ContinuousLayout property asks: Is layout really continuous?
DistRange	The DistRange property controls the constraints to place upon the data ranges that appear in the legend.

Name	Description
GroupGap	The GroupGap property represents the space between groups in a legend. (Columns when the legend is vertical; rows when the legend is horizontal.)
HorizItemGap	The HorizItemGap property represents the space between items.
InsideItemGap	The InsideItemGap property represents the space between symbol and String inside item.
LabelGenerator	The LabelGenerator property holds a reference to the label generator class.
LayoutStyle	The LayoutStyle property represents the style to use when laying out ranges.
MarginGap	The MarginGap property represents the space between outside and inside of legend.
UserLabels	The UserLabels property represents the list of user-specified legend labels.

B.1.11 JCChart3dLegendManager

Name	Description
FieldGap	The FieldGap property defines the gap between fields in JCMultiColumnStrings.
OutlineColor	The OutlineColor property determines the user-specified outline color for legend items.

B.1.12 JCChart3dStyle

Name	Description
LineStyle	The LineStyle property controls the line style to be used for this chart style.
SymbolStyle	The SymbolStyle property controls the symbol style to be used for this chart style.

B.1.13 JContour

Name	Description
Contoured	The Contoured property asks: Draw the contour lines?
ContourLevels	The ContourLevels property represents a list of contour levels.
ContourMapping	The ContourMapping property represents a mapping of contour levels to contour styles.
ContourStyles	The ContourStyles property represents a list of contour styles which determine how zones and contour lines are drawn.
Zoned	The Zoned property asks: Draw contour zones?
ZoneMethod	The ZoneMethod property represents a zoning filling method.

B.1.14 JContourLevels

Name	Description
IsDefault	The Default property asks: Are linear contour levels generated automatically (based on numLevels)?
Levels	The Levels property represents a strictly increasing array of contour levels.
NumLevels	The NumLevels property represents the number of contour levels.

B.1.15 JContourStyle

Name	Description
FillStyle	The FillStyle property controls the fill style to be used for this contour style.
LineStyle	The LineStyle property controls the line style to be used for this contour style.
SymbolStyle	The SymbolStyle property controls the symbol style to be used for this contour style.

B.1.16 JCData3dContourIndex

Name	Description
ContourStyleIndex	The ContourStyleIndex property controls the index of the contour style selected.
LowerContourRangeValue	The LowerContourRangeValue property represents the lower value of the range for the selected contour.
UpperContourRangeValue	The UpperContourRangeValue property represents the upper value of the range for the selected contour.

B.1.17 JCData3dGridIndex

Name	Description
x	The X- property controls the X-index of selected data point.
y	The Y-property controls the Y-index of the selected data point.

B.1.18 JCData3dIndex

Name	Description
DataView	The DataView property represents the data view object for the data point that this index references.
Distance	The Distance property controls the distance of a selected point from a pixel position.
Obj	The Obj property represents the component picked. It could be one of the Chart3dArea, Legend, Header or Footer components.

B.1.19 JCData3dPointIndex

Name	Description
Point	The Point property controls the point index of the selected point.
Series	The Series property controls the series index of the selected point.

B.1.20 JCElevation

Name	Description
MeshBottomColor	The MeshBottomColor property controls the mesh bottom color, in certain cases.
Meshed	The Meshed property asks: Are mesh lines drawn?
MeshTopColor	The MeshTopColor property controls the mesh top color, which is only drawn in certain cases.
Shaded	The Shaded property asks: Are facets shaded?
ShadedBottomColor	The ShadedBottomColor property controls the shaded bottom color, which is only drawn in certain cases. Currently not available for surfaces in the Java 3D API version of JClass Chart 3D.
ShadedTopColor	The ShadedTopColor property controls the shaded top color, which is only drawn in certain cases. Currently not available for surfaces in the Java 3D API version of JClass Chart 3D.
Transparent	The Transparent property asks: Are surfaces/bars transparent? or equivalently: Are hidden lines drawn?

B.1.21 JCGridColor

Name	Description
Color	The Color property determines the color to give selected bars.
DataIndex	The DataIndex property represents the grid index of the selected bar (or row/column or bars).

B.1.22 JCGridLines

Name	Description
LineStyle	The LineStyle property controls the gridline color, width, and pattern.
PlaneMask	The PlaneMask property controls the plane mask.

B.1.23 JLineStyle

Name	Description
Cap	The Cap property dictates the cap style to use at the ends of a line.
Color	The Color property determines the color used to draw the line.
Join	The Join property dictates the join style to use when joining two lines.
Pattern	The Pattern property dictates the pattern used to draw a line.
Width	The Width property controls line width.

B.1.24 JCPlotCube

Name	Description
Background	The Background property determines the plotCube's background color. If null, the chart3dArea's background color is used.
Ceiling	The Ceiling property determines the plotCube's ceiling projection.
Floor	The Floor property determines the plotCube's floor projection.
Foreground	The Foreground property determines the plotCube's foreground color. If null, the chart3dArea's foreground color is used.
xScale	The xScale property determines the scale in the X direction.
yScale	The yScale property determines the scale in the Y direction.
zScale	The zScale property determines the scale in the z direction.

B.1.25 JCProjection

Name	Description
Contoured	The Contoured property asks: Are contour lines drawn for this projection?

Name	Description
Zoned	The Zoned property asks: Are contour zones drawn for this projection?

B.1.26 JCSscatter

Name	Description
DropLines	The DropLines property asks: Are drop lines drawn? (Not currently available in the Java 3D API version of JClass Chart 3D.)

B.1.27 JCSurface

Name	Description
Solid	The Solid property asks: Should solid “skirts” under the surface be drawn?
xMeshFilter	The xMeshFilter property represents the filter value for X mesh lines. In the Java 3D API version of JClass Chart 3D, you can show either none or all of the meshed lines.
xMeshShowing	The xMeshShowing property asks: Are X mesh lines showing? In the Java 3D API version of JClass Chart 3D, you can show either none or all of the meshed lines.
yMeshFilter	The yMeshFilter represents the filter value for Y mesh lines. In the Java 3D API version of JClass Chart 3D, you can show either none or all of the meshed lines.
yMeshShowing	The yMeshShowing property asks: Are Y mesh lines showing? In the Java 3D API version of JClass Chart 3D, you can show either none or all of the meshed lines.

B.1.28 JCSymbolStyle

Name	Description
Color	The Color property determines the color used to paint the symbols.
Shape	The Shape property determines the shape of symbol that will be drawn.

Name	Description
Size	The <code>Size</code> property determines the size of the symbols. Note that a value of zero size means the symbol will not be drawn.

B.1.29 JCValueLabel

Name	Description
Label	The <code>Label</code> property specifies the text displayed inside the value label.
TickOnly	The <code>TickOnly</code> property specifies that only a minor tick is drawn for this label if <code>true</code> .
Value	The <code>Value</code> property controls the position of a label in data space along a particular axis.

B.1.30 JCView3d

Name	Description
Perspective	The <code>Perspective</code> property determines the plot cube perspective value.
xRotation	The <code>xRotation</code> property represents the X-rotation angle.
yRotation	The <code>yRotation</code> property represents the Y-rotation angle.
zRotation	The <code>zRotation</code> property represents the Z rotation angle.

B.1.31 JCViewport

Name	Description
HorizontalShift	The <code>HorizontalShift</code> property controls the horizontal shift as a multiple of the original viewport size.
Normalized	The <code>Normalized</code> property asks: Is the viewport normalized?
PreviewMethod	The <code>PreviewMethod</code> property controls the preview method.
Scale	The <code>Scale</code> property represents the zoom factor.
VerticalShift	The <code>VerticalShift</code> property controls the vertical shift as a multiple of the original viewport size.

B.2 Chart3d.Event

B.2.1 Chart3dDataEvent

Name	Description
Index	The Index property controls the index object which gives information about which grid or point index affected.
Type	The Type property contains the type of change that has happened to the chart data.

B.3 Chart3d.j2d

B.3.1 JCChart3dJava2d

Name	Description
JCChart3dJava2d	Default constructor, required by Java Beans.

B.4 Chart3d.j3d

B.4.1 JCChart3dJava3d

Name	Description
JCChart3dJava3d	Default constructor, required by Java Beans.

Appendix C

Additional Common JClass Chart 3D 3D Methods

[Chart3D](#) ■ [Chart3d.Event](#)

This appendix summarizes the JClass Chart 3D 3D extra methods for all commonly used classes, in alphabetical order.

C.1 Chart3D

C.1.1 Chart3dDataView

Name	Description
<code>coordToDataCoord</code>	Same as <code>map()</code> . Converts pixel coordinates to data space coordinates. Parameters: <i>x</i> – x value in screen pixels. <i>y</i> – y value in screen pixels. Returns: Point3d instance.
<code>dataCoordToCoord</code>	Same as <code>unmap()</code> . Converts data coordinates to pixel coordinates. Parameter: <i>point</i> – The point in 3d data space to be transformed. Returns: AWT Point object representing the location in screen pixels (relative to the Chart 3D component).

Name	Description
dataIndexToCoord	<p>Similar to <code>unpick()</code> for a specific this specific data view. Converts a <code>JCData3dIndex</code> instance (containing a data view and a point index for either grid data or point data) to pixel values relative to the Chart 3D component.</p> <p>Parameter: <i>index</i> – Object representing the index of the point to unpick. This is either a <code>JCData3dGridIndex</code> representing the (x, y) index of grid data point or a <code>JCData3dPointIndex</code> representing the (series, point) index of a point in a point data set.</p> <p>Returns: AWT Point object representing the location is screen pixels relative to the Chart 3D component.</p>
coordToDataIndex	<p>Similar to <code>pick()</code> for a specific data view. Converts pixel values relative to the Chart 3D component to a <code>JCData3dIndex</code> instance representing the index of the picked point. This is either a <code>JCData3dGridIndex</code> representing the (x, y) index of grid data point or a <code>JCData3dPointIndex</code> representing the (series, point) index of a point in a point data set.</p> <p>Parameters: <i>x</i> – The X value of screen position. <i>y</i> – The Y value of screen position.</p> <p>Returns: The <code>JCData3dIndex</code> object representing the index of the picked point.</p>
dragZValue	<p>Finds a new z value for a given point based on a given pixel position. Given a start point A (for grid data a point on the grid; for point data one of the points in the list of series) and a point P on the screen, project the line AP (in 3D-space) onto the line through A parallel to the z axis and find the z value that corresponds to P on the projected line.</p> <p>Parameters: <i>data</i> – The data for which this operation is to take place. It is either an instance of <code>Chart3dGridData</code> or <code>Chart3dPointData</code>. <i>index</i> – The data index of the point. For grid data, this must be an instance of <code>JCData3dGridIndex</code>, which corresponds to an X- and Y-grid position specification. For point data, this must be an instance of <code>JCData3dPoint</code> index which corresponds to the series and point number of the point. <i>x</i> – The X- value of screen position. <i>y</i> – The Y-value of screen position.</p> <p>Returns: The new computed z value.</p>

Name	Description
gridValue	<p>Given grid data and an (x,y) point on the visible xy plane within the grid, do bilinear interpolation using the four closest grid points and return the corresponding z value.</p> <p>Parameters:</p> <p><i>data</i> – The internal grid data object. This can be retrieved from a Chart3dDataView object via getElevationData() or getZoneData().</p> <p><i>x</i> – The X- data-space value</p> <p><i>y</i> – The Y-data-space value</p> <p>Returns:</p> <p>The interpolated z value. Returns the data's hole value if an error occurs.</p>

C.1.2 Chart3dGridData

Name	Description
getX	Returns the X- value at the specified index.
getXClosest	<p>Returns the index that contains the X- value closest to the specified value.</p> <p>Parameter:</p> <p><i>x</i> – The value for which the closest index should be found.</p>
getY	Returns the Y-value at the specified index.
getYClosest	<p>Returns the index that contains the Y-value closest to the specified value.</p> <p>Parameter:</p> <p><i>y</i> – The value for which the closest index should be found.</p>

C.1.3 Chart3dPointData

Name	Description
getPoint	<p>Returns the point indexed by pointNum in the series indexed by seriesNum.</p> <p>Parameters:</p> <p><i>seriesNum</i> – The series index of the point wanted.</p> <p><i>pointNum</i> – The point index of the point wanted.</p> <p>Returns:</p> <p>The point indexed by series and point.</p>

C.1.4 Chart3dPointSeries

Name	Description
getPoint	Returns the point in the points array indexed by point. Parameter: <i>point</i> – The index of the point to be returned. Returns: The point indexed by point.

C.1.5 JCAxis

Name	Description
getValueLabel	Retrieves the value label for the specified value from the list of user-specified value labels. Parameter: <i>value</i> – Data value corresponding to the value label. Returns: JCValueLabel instance.

C.1.6 JCChart3d

Name	Description
addChart3dListener	Adds listener to changes in JClass Chart 3D. Called after zoom, translate, scale, rotate, or edit (interactive only). Parameter: <i>l</i> – The listener to be added.
getDrawingArea	Gets the drawing area represented by this chart. Returns: Rectangle object containing drawing area.
getDrawingAreaHeight	Gets the height of the drawing area represented by this chart. Specified by: getDrawingAreaHeight in interface com.klg.jclass.util.legend.LegendComponentLayoutUser. Returns: The height of the drawing area.

Name	Description
getDrawingAreaWidth	<p>Gets the width of the drawing area represented by this chart.</p> <p>Specified by: getDrawingAreaWidth in interface com.klg.jclass.util.legend.LegendComponentLayoutUser.</p> <p>Returns: The width of the drawing area.</p>
getLayoutHints	<p>Sets and gets layout hints for chart children. Hints are rectangle objects.</p> <p>A value of Integer.MAX_VALUE in the rectangle's members indicates to calculate default values during layout. Other values indicate to the layout to use that value. For example, a rectangle with members x=5, y=10, width=MAX_VALUE, and height=200, would indicate to the layout mechanism that the chart child should be placed at (5,10), have a height of 200, and use the default width. Layout hints are only used by the DefaultChartLayout layout manager.</p> <p>Parameter: <i>child</i> – Chart child – either the chart3dArea, legend, header, or footer. <i>layoutHints</i> – Rectangle object containing the desired layout hints.</p>
getUI	<p>Returns and sets the UI for JCCart3d.</p> <p>Overrides: setUI in class javax.swing.JComponent.</p> <p>Parameter: <i>newUI</i> – The new user interface object.</p>
getUIClassId	<p>Returns the UIClass ID for JCCart3d.</p> <p>Overrides: getUIClassID in class javax.swing.JComponent.</p>
isProjection	<p>Is the surface represented by the first dataView a 3d view or a 2d projection?</p> <p>Returns: A Boolean indicating whether the first dataView is a 2D projection or not.</p>

Name	Description
pick	<p>Given a screen position in pixels, returns a <code>JCData3dIndex</code> object that represents the index of the closest point in the elevation data set of the specified <code>ChartData3dView</code> instance. If no data view is supplied, all data views are considered when finding the closest point (only one <code>dataView</code> is currently supported). If the data in a data view is being updated when <code>pick()</code> is called, the result may be incorrect.</p> <p>Parameters: <i>p</i> – Pick point in pixels relative to the <code>JCChart3d</code> object <i>dataView</i> – Data view on which to perform pick; if null, all data views are used (only one <code>dataView</code> is currently supported).</p> <p>Returns: The <code>JCData3dIndex</code> object representing the index of the picked point. This is either a <code>JCData3dGridIndex</code> representing the (x, y) index of grid data point, a <code>JCData3dPointIndex</code> representing the (series, point) index of a point in a point data set, or a <code>JCData3dContourIndex</code> representing a contour range.</p>
printAll	<p>Prints this component and all of its subcomponents. Overridden from <code>java.awt.Component</code>, but should be used in the same way.</p> <p>Overrides: <code>printAll</code> in class <code>javax.swing.JComponent</code>.</p> <p>Parameter: <i>g</i> – The graphics object used to paint.</p>
recalc	<p>Recalculates the entire chart if it has been marked for recalculation.</p>
removeChart3dListener	<p>Removes listener to changes in <code>JClass Chart 3D</code> from list of listeners.</p> <p>Parameter: <i>l</i> – The listener to be removed.</p>
reset	<p>Performs a reset on the chart. Returns to the <code>chart3d</code> its default dataport.</p>

Name	Description
snapshot	<p>Takes a snapshot of the current chart and places it in an image of the specified type. The image types are as specified in the <code>BufferedImage</code> class. <code>BufferedImage.TYPE_INT_ARGB</code> is a good default for representing many possible colors. If using fewer than 256 colors, <code>BufferedImage.TYPE_BYTE_INDEXED</code> may prove to generate faster and smaller images.</p> <p>Parameter: <i>imagetype</i> – The type of image to write to, as defined in the <code>java.awt.image.BufferedImage</code> class.</p> <p>Returns: Image object containing snapshot of chart.</p>
unpick	<p>Returns the position in screen pixels of a particular point in a particular data set (grid data or point data).</p> <p>Parameters: <i>dataView</i> – The data view containing the specified series. <i>index</i> – The data index of the point. This is either a <code>JCData3dGridIndex</code> representing the (x, y) index of grid data point or a <code>JCData3dPointIndex</code> representing the (series, point) index of a point in a point data set.</p> <p>Returns: AWT <code>Point</code> object representing position in screen pixels relative to the <code>JCChart3d</code> object or null if the point does not exist.</p>
update	<p>Forces the chart to re-layout and recalculate.</p>
updateUI	<p>Updates the UI for <code>JCChart3d</code>.</p> <p>Overrides: <code>updateUI</code> in class <code>javax.swing.JComponent</code>.</p>

C.1.7 JCChart3dArea

Name	Description
getAxis	<p>Sets and returns the axis based on the given <code>axisId</code>.</p> <p>Parameter: <i>axisId</i> – The axis ID (either <code>AXIS_X</code>, <code>AXIS_Y</code>, or <code>AXIS_Z</code>).</p>
getDrawingArea	<p>Gets the bounding rectangle of the component's drawing area (its area minus the shadows and insets).</p> <p>See Also: <code>JComponent.setBorder(javax.swing.border.Border)</code>.</p>

Name	Description
getMinimumSize	Returns the minimum size for the chart area. Overrides: getMinimumSize in class javax.swing.JComponent. Returns: A Dimension object containing the minimum size.
recalc	If necessary, forces recalculation of the chart area.
reset	Returns the chart back to the default viewport settings.

C.1.8 JCContour

Name	Description
contourIndex	Returns the contour style index that corresponds to this level. This mapping is based on an even distribution of contour styles through the number of levels. Specified by: contourIndex in interface JCContourMapping.

C.1.9 JCContourLevels

Name	Description
getLevelFromValue	Calculates the contour level for this value. Note that we return a value between 0 and numLevels (inclusive - there should be one more contourStyle than contour level). Parameter: <i>value</i> – The data value from which a contour level is calculated.

C.1.10 JCPlotCube

Name	Description
hasCeilingProjection	Does this PlotCube have a ceiling projection?
hasFloorProjection	Does this PlotCube have a floor projection?
hasProjections	Does this PlotCube have any projections?

C.2 Chart3d.Event

C.2.1 Chart3dGridDataEvent

Name	Description
getX	Method which returns the X-index of the affected data. Returns -100, if all X- values are affected. Returns: index the X-index affected.
getY	Method which returns the Y-index of the affected data. Returns -100, if all Y-values are affected. Returns: index the Y-index affected.

C.2.2 Chart3dPointDataEvent

Name	Description
getPoint	Retrieves the point index associated with the event. Returns: int the index of the point affected. Returns -100 if all points are affected.
getSeries	Retrieves the series index associated with the event. Returns: int the index of the series affected. Returns -100 if all series are affected.

Index

3D scatter plots 66
4D charts 93
 creating 93
4D graphs 93, 94
 creating 93
 legend 95

A

access an element of a collection 18
action
 adding 102
 cancel 103
 custom 102
 customize 103
 edit 103
 pick 103
 removing 102
 reset 103
 rotate 103
 rotateEye 103
 rotateX 103
 rotateY 103
 rotateZ 103
 scale 104
 switchRotateAny 104
 switchRotateEye 104
 switchRotateX 104
 switchRotateY 104
 switchRotateZ 104
 translate 104
 zoom 104
addGridColor() 57
AllowUserChanges property 25
Alpha 118
ambient
 intensity 127
 light 125
annoFont property 28
annoFontCubeSize property 28
annotationMethod property 27, 40, 41
API 4
Appearance 118
assumptions 2
attenuation 127
AuralAttributes 118
AWT 104

axis
 annotation 39
 overview 39
 ValueLabels 41
 bounds 29
 controls 28
 custom label 42
 font 28
 labelling 39
 labels
 data
 label, method 40
 values method 40
 min and max 29
 scaling 38
 show 28
 size 28

B

Background subclass 115
bar chart 15, 31, 55
 4D 94
 color 32
 coloring 57
 fifteen basic types of surfaces and bars 49
 shading 32
bar spacing 56
bar Z origin 55
Base3dDataSource 73
Base3dGridDataSource 73
Base3dPointDataSource 73
basic graph types 52
batching
 property updates 23
 resource updates 23
Bean properties
 run-time 28
 setting 28
 setting interactively 28
 setting interactively at run-time 28
Behavior
 node 123
 subclass 115
BoundingLeaf subclass 116
BranchGroup

- compiling 120
- subclass 115

browsers and Java 3D 111

C

- calling methods 18
- cancel action 103
- Canvas3D 119
- cell zoning 61
- changeChart 105
- changing data 71
- chart
 - bar 31, 55
 - bar chart 32
 - bar, axis scaling 38
 - color 23
 - color defaults 24
 - colors 23
 - components 11
 - histograms 55
 - labelling 84
 - scatter 30
 - scatter plot 31
 - scatter plot of point data 31
 - scatter plot, axis scaling 38
 - scatter plots of grid data 31
 - specify background color 24
 - specify foreground color 24
 - surface 31
 - surface, axis scaling 38
 - updating chart 88
- chart 3D
 - basics 11
 - outputting 19
- chart 3D customizer 25
- chart customizer
 - enabling 25
- chart data model hierarchy 70
- chart data source
 - definition 16
- chart mesh
 - colors 61
 - filtering 61
 - hidden lines 62
- chart styles 67
 - default 67
- chart surface
 - colors 62
 - solid 63
- chart types 14, 49
 - 4D 93
 - BAR 49
 - bar 15
 - contoured and zoned 54
 - contours 52
 - meshed 52
 - meshed and contoured 53
 - meshed and shaded 53
 - meshed and zoned 53
 - meshed, contoured, zoned 55
 - meshed, shaded, contoured 54
 - meshed, shaded, contoured, zoned 55
 - meshed, shaded, zoned 54
 - SCATTER 49
 - scatter plot 15
 - shaded 52
 - shaded and contoured 53
 - shaded and zoned 54
 - shaded, contoured, zoned 55
 - SURFACE 49
 - surface 15
 - zoned 52, 54
- Chart3d listener 105
- Chart3dArea 22
- chart3dDataChange 104
- Chart3dDataEvent 88
- Chart3dDataListener 88, 135
- Chart3dDataManager 65, 90
- Chart3dDataManager interface 49
- Chart3dDataModel 135
- Chart3DDataModel interface 71
- Chart3dDataSupport 90
- Chart3DDataView
 - containment hierarchy 22
- Chart3dDataView 65
- Chart3dGridData 49, 67
- Chart3DGridDataModel 70
- Chart3dGridDataModel 22, 93, 135
- Chart3dGridDataModel interface 72
- Chart3dGridDataModel method
 - getXGrid() 72
 - getYGrid() 72
 - getZValues() 72
- chart3dJava2d JavaBean 17, 28
- chart3dJava3d JavaBean 28
- Chart3DPointData 65
- Chart3dPointData 66
- Chart3DPointDataModel 70
- Chart3dPointDataModel 22, 65, 135
- Chart3dPointDataModel interface 71
- Chart3dPointSeries 66, 67
- Chart3dPointSeries label property 31
- Chart3dStyles
 - default 24
- chartable data source 69
- ChartDataViewSeries
 - property summary 150
- chartStyle 67, 70
- ChartStyle property 31
- chartType 49

- checkboxlist
 - startup 12
- class hierarchy 20
- Clip subclass 116
- ColoringAttributes 118
- colors
 - bar charts 57
 - chart mesh 61
 - chart surface 62
 - graph mesh 61
 - graph surface 62
 - property 57
 - surface chart 32
- com.sun.j3d 112, 113
- comments on product 7
- common methods
 - Chart3D 151
 - Chart3d.Event 159
 - Chart3dDataView 151
 - Chart3dDataView, coordToDataCoord 151
 - Chart3dDataView, coordToDataIndex 152
 - Chart3dDataView, dataCoordToCoord 151
 - Chart3dDataView, dataIndexToCoord 152
 - Chart3dDataView, dragZValue 152
 - Chart3dDataView, gridValue 153
 - Chart3dGridData 153
 - Chart3dGridData, getX 153
 - Chart3dGridData, getXClosest 153
 - Chart3dGridData, getY 153
 - Chart3dGridData, getYClosest 153
 - Chart3dGridDataEvent 159
 - Chart3dGridDataEvent, getX 159
 - Chart3dGridDataEvent, getY 159
 - Chart3dPointData 153
 - Chart3dPointData, getPoint 153
 - Chart3dPointDataEvent 159
 - Chart3dPointDataEvent, getPoint 159
 - Chart3dPointDataEvent, getSeries 159
 - Chart3dPointSeries 154
 - Chart3dPointSeries, getPoint 154
- JCAxis 154
- JCAxis, getValueLabel 154
- JCChart3d 154
- JCChart3d, addChart3dListener 154
- JCChart3d, getDrawingArea 154
- JCChart3d, getDrawingAreaHeight 154
- JCChart3d, getDrawingAreaWidth 155
- JCChart3d, getLayoutHints 155
- JCChart3d, getUI 155
- JCChart3d, getUIClassId 155
- JCChart3d, isProjection 155
- JCChart3d, pick 156
- JCChart3d, printAll 156
- JCChart3d, recalc 156
- JCChart3d, removeChart3dListener 156
- JCChart3d, reset 156
- JCChart3d, snapshot 157
- JCChart3d, unpick 157
- JCChart3d, update 157
- JCChart3d, updateUI 157
- JCChart3dArea 157
- JCChart3dArea, getAxis 157
- JCChart3dArea, getDrawingArea 157
- JCChart3dArea, getMinimumSize 158
- JCChart3dArea, recalc 158
- JCChart3dArea, reset 158
- JCContour 158
- JCContour, contourIndex 158
- JCContourLevels 158
- JCContourLevels, getLevelFromValue 158
- JCPlotCube 158
- JCPlotCube, hasCeilingProjection 158
- JCPlotCube, hasFloorProjection 158
- JCPlotCube, hasProjections 158
- compiled-retained mode 123
- compiling a BranchGroup 120
- component parameter 19
- content branch graph 120
- contents 35
- contour 30, 67
 - and meshed charts 53
 - and meshed graphs 53
 - and shaded charts 53
 - and shaded graphs 53
 - and zoned charts 54
 - and zoned graphs 54
- bars 51
- charts 52
- contoured 50, 51, 58, 59
- display 58
- graphs 52
- levels, customizing 95
- lines 51
 - legend 30
- meshed, shaded charts 54
- meshed, shaded, zoned charts 55
- meshed, shaded, zoned graphs 55
- meshed, zoned charts 55
- meshed, zoned graphs 55
- projection 59
- shaded, zoned charts 55
- shaded, zoned graphs 55
- style
 - fill, color 97
 - fill, pattern 97
 - line, color 97
 - line, pattern 97
 - line, width 97
- styles
 - customizing 96
 - default 96

- ContourStyles
 - default 24
 - property 51
- createCubicSampledDataModel 82
- createDataCopy() 81
- createJava2dChart() 13
- createJava3dChart() 13
- createLinearSampledDataModel 82
- createShadedDataModel 81
- createSmoothedDataModel 81
- custom
 - actions 102
 - axes labels 42
 - legends 34
- customize action 103
- customizer 25
 - chart 3D 25

D

- data
 - binding 16
 - SQL 81
 - using JDBCDataSource 81
 - bounds 29
 - dragZValue parameter 106
 - formatting 75
 - interfaces, summary 91
 - label, clustering 40
 - listener 104
 - loading, from XML source 78
 - loading, Swing TableModel 77
 - min and max 29
 - model 70
 - types 13
- data source 22, 69, 70
 - changing data 71
 - creating 82
 - grid data 70
 - internal data 71
 - loading data from a file 74
 - formatted file 75
 - irregular grid 76
 - irregular grid data 74
 - point data 77
 - regular grid data 74
 - standard file format 75
 - making an updating chart 88
 - point data 70
 - pre-built 73
 - responsibility 70
 - simplest chart data source possible 82
 - support classes 88
 - Chart3dDataEvent 88

- Chart3dDataListener 88
- Chart3dDataManager 90
- Chart3dDataSupport 90
 - updating data source 91
- data.gridValue method 107
- databases 69
- dataCopy() 81
- dataIndex property 57
- debugging
 - using customizer 25
- default user interactions 99
- depth cue 128
- DepthComponent 118
- Dimension object 35
- Direct3D 109
- directional light 125
- DistributionRange property 32
- dragZValue parameter 106
 - data 106
 - index 107
 - x 107
 - y 107
- drawLegendItem() 37
- drawLegendItemSymbol() 37
- drawType 35
- DrawZones with DrawShaded 55
- drop line 66
 - grid data 67
 - style, controlling 66
- dropLines property 66

E

- Edit action 103
- Editable3dDataModel 135
- EditableChart3dDataModel 49, 65
- EditableChartDataModel 85
- Editing
 - property 26
- elevation data source 22
- elevationData 65
 - property 49, 66, 70
- elevationDataSource 65
 - property 49, 65, 70
- encode method 19
- encoding parameter 19
- ExponentialFog 131

F

- FAQs 7
- feature overview 1
- fill
 - color 97

- pattern 97
- findGridColor() 57
- fireChart3dDataEvent 90
- fog 129
 - ExponentialFog 131
 - fogFactor 131
 - LinearFog 129
 - node 128
 - subclass 115, 116
- fogFactor 129, 131
- footer
 - adding 47
 - title 47
- formatted file 75

G

- Geometry 118
- get method 27
- getLegendItems() 36
- getOutlineColor() 37
- getting object properties 16
- GIF 19, 25
- graph mesh
 - colors 61
 - filtering 61
 - hidden lines 62
- graph surface
 - colors 62
 - solid 63
- graph types
 - 4D 93, 94
 - contoured 52
 - contoured and zones 54
 - meshed 52
 - meshed and contoured 53
 - meshed and shaded 53
 - meshed and zoned 53
 - meshed, contoured, zoned 55
 - meshed, shaded, contoured 54
 - meshed, shaded, contoured, zoned 55
 - meshed, shaded, zoned 54
 - shaded 52
 - shaded and contoured 53
 - shaded and zoned 54
 - shaded, contoured, zoned 55
 - zoned 52, 54
- GRID 76
- grid data 13, 49, 67, 68, 70
 - data source 70
 - differences with point data 72
- gridline 45
 - color 46
 - pattern 46
 - styles 45

- width 47
- gridValue method 107
 - data 107
 - x 107
 - y 107
- Group class 115
- Group subclass
 - BranchGroup 115
 - OrderedGroup 115
 - SharedGroup 115
 - Switch 115
 - TransformGroup 115
- GroupGap property 33

H

- hardware acceleration 111
- header
 - adding 47
 - title 47
- histograms 55, 56
- hole values 87
- HoleValueChart3dDataModel 49, 65, 135
- HoleValueChartDataModel 87
- HorizontalItemGap property 33
- horizontalShiftJCVViewport property 101
- HTML 78

I

- IGRID 76
- ImageComponent 118
- immediate mode 122
- index, dragZValue parameter 107
- indexed properties 18
- inheritance hierarchy 20
- initialization() 123
- InsideItemGap property 33
- instantiating a chart 12
- interactivity
 - setting object properties at run-time 17
- interface summary 135
 - Chart3dDataListener 135
 - Chart3dDataModel 135
 - Chart3dGridDataModel 135
 - Chart3dPointDataModel 135
 - Editable3dDataModel 135
 - HoleValueChart3dDataModel 135
 - JCChart3dListener 135
 - JCContourMapping 135
 - LabelledChart3dGridDataModel 135
 - LabelledChart3dPointDataModel 135
- internal data 71
- internationalization support 98

- Internet Explorer 111
- introducing JClass Chart 3D 1
- irregular grid 76
 - data 74
- isTitleItem() 36
- itemInfo 35

J

- Java 3D 109
 - behavior 123
 - browsers 111
 - charting features 123
 - lighting 124
 - texture mapping 123
- Java 3D API 111
 - mode, compiled-retained 122
 - mode, immediate 122
 - mode, retained 122
- Java IDE
 - setting object properties at design-time 17
- JavaBean 28
 - chart3dJava2d 17, 28
 - chart3dJava3d 28
- javax.prefs 112
- javax.media.j3d 112
- javax.media.j3d.javax.vecmath 112
- JCActionTable class 101
 - adding individual action 102
 - custom actions 102
 - removing individual actions 102
- JCAxis 22, 27
 - Min and Max properties 29
- JCAxis.ANNOTATION_VALUE 39
- JCAxis.ANNOTATION_VALUE_LABELS 40
- JCAxis.ANNOTATION_DATA_LABELS 39
- JCChart3D object hierarchy 22
- JCChart3d STEPPED legend 34
- JCChart3DArea 22
- JCChart3dEvent 104
- JCChart3dJava3d
 - eliminating references 18
- JCChart3dLegend 30, 33
- JCChart3dLegendLabelGenerator 34
- JCChart3dLegendLabelGenerator interface 33
- JCChart3dListener 135
- JCChartStyle 67
- JCContour 51
 - contoured 51
 - zoned 51
- JCContour class 59
- JCContourLevels 22, 51
- JCContourLevels property
 - isDefault 95
 - levels 95
 - max 95
 - min 95
 - numLevels 95
- JCContourMapping 59, 135
- JCContourStyle 32
- JCData3dGridIndex 57
- JCData3dIndex class 105
- JCData3dUtil 81
- JCDefault3dGridDataSource 73
- JCDefault3dPointDataSource 73
- JCEditable3dGridDataSource 73
- JCEditable3dPointDataSource 74
- JCElevation 50
 - meshed 50
 - shaded 50
- JCFile3dDataSource 74, 75
- JCGridLines 45
- JCLabelGenerator interface 42
- JClass Chart 3D
 - Java 2 JavaBean 28
 - Java 3D JavaBean 28
- JClass Chart 3D customizer 25
- JClass technical support 6
 - contacting 6
- JCLegend 22
- JCLegend Toolkit 34
- JCLegendItem 34, 35
- JCLegendItem property listing 35
- JCLegendPopulator 34, 36
- JCLegendRenderer 34, 36
- JCMultiFieldString 33
- JComponent 22
- JCPick3dListener interface 105
- JCPlotCube 38, 59
- JCProjection 59
- JCSurface 61
- JCSwing3dDataSource 74, 77
- JCTexture2D 124
- JCViewport 101
- JCViewport class
 - rotation 101
- JCXMLE3dDataSource 74
- JDBC3dDataSource 74
- JLabel 47
- JPEG 19, 25

L

- label
 - adding 47
 - clustering 44
 - customize 29
 - legend 29
 - overriding 33
 - selection 44

- LabelGenerator property 33
- LabelledChart3dGridDataModel 49, 84, 135
- LabelledChart3dPointDataModel 65, 84, 135
- LabelledChartDataModel 84
- labelling your chart 84
- Labels property 33
- layoutLegend() method 35
- LayoutStyle property 32
- Leaf class 115
- Leaf subclass
 - Background 115
 - Behavior 115
 - BoundingLeaf 116
 - Clip 116
 - Fog 115, 116
 - Light 115, 116
 - Link 116
 - Morph 116
 - Shape3D 115, 116
 - Sound 115, 116
 - Soundscape 116
 - ViewPlatform 117
- legend 29, 30
 - 4D graphs 95
 - contour lines 30
 - custom 34
 - custom, population 36
 - custom, rendering 36
 - display 30
 - layout 34
 - orientation 30
 - positioning 30
 - text 30
 - using 29
- levels 22
- license 4
- licensing 4
- light 124
 - ambient 125
 - depth cue 128
 - directional 125
 - ExponentialFog 131
 - fog 129
 - fogFactor 131
 - LinearFog 129
 - point 126
 - spot 127
 - subclass 115, 116
- line
 - color 68, 97
 - pattern 68, 97
 - width 68, 97
- LinearFog 129
- LineAttributes 118
- LineStyle 31
- Link subclass 116

- list 33
- listener 104
 - Chart3d 105
 - data listener 104
 - mechanism 104
 - pick 105
- loading data 16
 - from a file 74
 - formatted file 75
 - irregular grid 76
 - irregular grid data 74
 - point data 77
 - regular grid data 74
 - standard file format 75
 - from a Swing TableModel 77
 - XML 78
- Locale 119
- Locale object
 - scene graph programming model 110

M

- makeLabel() 42
- mapping 105
 - user input 101
- MarginGap property 33
- Material 118
- Max 22, 29
- MediaContainer 118
- mesh 50
 - and contoured charts 53
 - and contoured graphs 53
 - and shaded charts 53
 - and shaded graphs 53
 - and zoned charts 53
 - and zoned graphs 53
 - bars 50
 - charts 52
 - colors 61
 - contoured, zoned charts 55
 - contoured, zoned graphs 55
 - controls 61
 - filtering 61
 - graphs 52
 - hidden lines 62
 - shaded, contoured graphs 54
 - shaded, contoured, zoned charts 55
 - shaded, contoured, zoned graphs 55
 - shaded, contours charts 54
 - shaded, zoned charts 54
 - shaded, zoned graphs 54
- meshBottomColor property 50
- meshTopColor property 50
- method

- calling 18
- encode 19
- get 27
- set 27
- methods
 - common 3D methods, Chart3D 151
 - common 3D methods, Chart3d.Event 159
- Min 22, 29
- minorTick property 41
- modifying data 85
- Morph subclass 116

N

- Name property 31
- Netscape Navigator 111
- node
 - Behavior 123
 - class 115
 - scene graph programming model 110
- NodeComponent Class 117
- nomenclature 70
- normalizedJCViewport property 101

O

- object collection
 - accessing an element 18
 - working with 18
- object containment 21
 - hierarchy 21
 - listing 21
- object property listing 137
- opacity 25
- OpenGL 109, 124
- OrderedGroup subclass 115
- Orientation property 30
- output parameter 19
- outputtingJClass Chart 3D 19
- overriding
 - labels 33
 - public methods 36

P

- paintChart 105
- parameter
 - component 19
 - encoding 19
 - output 19
- performance-enhancing features 111
- perspective 37
- PhysicalBody 119

- PhysicalUniverse 119
- pick 105, 106
 - action 103
 - listener 105
- pickRectangle 35
- plot cube 12
- PNG 19, 25
- POINT 76
- point data 13, 66, 70
 - difference with grid data 72
 - file 77
 - source 70
- point light 126
- PointAttributes 118
- PolygonAttributes 118
- pre-built DataSources 73
- previewMethodJCViewport property 101
- processStimulation() 123
- product feedback 7
- programming
 - advanced 93
 - bars 49
 - basics 18
 - Java 3D API 109
 - Java 3D API, system set-up 110
 - scatter plots 65
 - scene graph model 109
 - surfaces 49
 - user interaction 99
- programming scatter plots 65
- properties 27
 - batching updates 23
 - listing 137
 - setting Bean properties at run-time 28
 - setting Bean properties interactively 28
- property
 - Anchor 31
 - annoFont 28
 - annoFontCubeSize 28
 - AnnotationMethod 41
 - annotationMethod 27, 39, 40
 - Background 24
 - Chart3D 137
 - Chart3d.Event 150
 - Chart3d.j2d 150
 - Chart3d.j3d 150
 - Chart3dData 137
 - Chart3dData, DataOK 137
 - Chart3dData, DataSource 137
 - Chart3dData, HoleValue 137
 - Chart3dData, Name 137
 - Chart3dDataEvent 150
 - Chart3dDataView 137
 - Chart3dDataView, ChartType 137
 - Chart3dDataView, Contour 137
 - Chart3dDataView, Elevation 137

Chart3dDataView, GridColors 138
 Chart3dDataView, Name 138
 Chart3dDataView, ZoneData 138
 Chart3dDataView, ZoneDataSource 138
 Chart3dGridData 138
 Chart3dGridData, ChartStyle 138
 Chart3dGridData, NumSeries 138
 Chart3dGridData, NumX 138
 Chart3dGridData, NumY 138
 Chart3dGridData, Series 139
 Chart3dGridData, xGrid 138
 Chart3dGridData, xLabels 138
 Chart3dGridData, yGrid 138
 Chart3dGridData, yLabels 138
 Chart3dGridData, ZValues 138
 Chart3dPointData 138
 Chart3dPointData, ChartStyle 139
 Chart3dPointData, Label 139
 Chart3dPointData, NumPoints 139
 Chart3dPointData, Points 139
 Chart3dPointSeries 139
 Chart3dPointSeries label 31
 ChartDataViewSeries 150
 ChartDataViewSeries, Index 150
 ChartDataViewSeries, Type 150
 ChartStyle 31
 color 24, 57
 contourStyles 51
 dataIndex 57
 DistributionRange 32
 dropLines 66
 editing 26
 elevationData 49, 66, 70
 elevationDataSource 49, 65, 70
 Foreground 24
 GroupGap 33
 HorizontalItemGap 33
 horizontalShift 101
 InsideItemGap 33
 IsShowing 48
 JCAxis 139
 JCAxis, AnnoFont 139
 JCAxis, AnnoFontCubeSize 139
 JCAxis, AnnotationMethod 139
 JCAxis, AxisId 139
 JCAxis, GridLines 139
 JCAxis, LabelGenerator 139
 JCAxis, Max 140
 JCAxis, MaxIsDefault 140
 JCAxis, Min 140
 JCAxis, MinIsDefault 140
 JCAxis, Origin 140
 JCAxis, Showing 140
 JCAxis, Title 140
 JCAxis, TitleFont 140
 JCAxis, TitleFontCubeSize 140
 JCAxis, ValueLabels 140
 JCBar 140
 JCBar, xFormat 140
 JCBar, xSpacing 140
 JCBar, yFormat 140
 JCBar, ySpacing 140
 JCCChart3d 141
 JCCChart3d, About 141
 JCCChart3d, AllowUserChanges 141
 JCCChart3d, Batched 141
 JCCChart3d, CancelKey 141
 JCCChart3d, Chart3dArea 141
 JCCChart3d, CustomizerName 141
 JCCChart3d, DataView 141
 JCCChart3d, Footer 141
 JCCChart3d, FooterLayoutHints 141
 JCCChart3d, Header 141
 JCCChart3d, HeaderLayoutHints 141
 JCCChart3d, Legend 141
 JCCChart3d, LegendLayoutHints 141
 JCCChart3d, LegendManager 141
 JCCChart3d, Pick3dListener 141
 JCCChart3d, ResetKey 141
 JCCChart3d, WarningDialog 141
 JCCChart3dArea 142
 JCCChart3dArea, ActionHandler 142
 JCCChart3dArea, Axes 142
 JCCChart3dArea, Bar 142
 JCCChart3dArea, InAction 142
 JCCChart3dArea, PlotCube 142
 JCCChart3dArea, PreferredSize 142
 JCCChart3dArea, Scatter 142
 JCCChart3dArea, Surface 142
 JCCChart3dArea, View3d 142
 JCCChart3dArea, Viewport 142
 JCCChart3dArea, xAxis 142
 JCCChart3dArea, yAxis 142
 JCCChart3dArea, zAxis 142
 JCCChart3dJava2d 150
 JCCChart3dJava2dJCCChart3dJava3d 150
 JCCChart3dJava3d 150
 JCCChart3dJava3d, JCCChart3dJava2d 150
 JCCChart3dLegend 142
 JCCChart3dLegend, ContinuousLayout 142
 JCCChart3dLegend, DistRange 142
 JCCChart3dLegend, GroupGap 143
 JCCChart3dLegend, HorizItemGap 143
 JCCChart3dLegend, InsideItemGap 143
 JCCChart3dLegend, LabelGenerator 143
 JCCChart3dLegend, LayoutStyle 143
 JCCChart3dLegend, MarginGap 143
 JCCChart3dLegend, UserLabels 143
 JCCChart3dLegendManager 143
 JCCChart3dLegendManager, FieldGap 143
 JCCChart3dLegendManager, OutlineColor 143
 JCCChart3dStyle 143

- JCChart3dStyle, LineStyle 143
- JCChart3dStyle, SymbolStyle 143
- JCContour 144
- JCContour, Contoured 144
- JCContour, ContourLevels 144
- JCContour, ContourMapping 144
- JCContour, ContourStyles 144
- JCContour, Zoned 144
- JCContour, ZoneMethod 144
- JCContourLevels 144
- JCContourLevels, IsDefault 144
- JCContourLevels, Levels 144
- JCContourLevels, NumLevels 144
- JCContourStyle 144
- JCContourStyle, FillStyle 144
- JCContourStyle, LineStyle 144
- JCContourStyle, SymbolStyle 144
- JCData3dContourIndex 145
- JCData3dContourIndex, ContourStyleIndex 145
- JCData3dContourIndex, LowerContourRangeValue 145
- JCData3dContourIndex, UpperContourRangeValue 145
- JCData3dGridIndex 145
- JCData3dGridIndex, x 145
- JCData3dGridIndex, y 145
- JCData3dIndex 145
- JCData3dIndex, DataView 145
- JCData3dIndex, Distance 145
- JCData3dIndex, Obj 145
- JCData3dPointIndex 145
- JCData3dPointIndex, Point 145
- JCData3dPointIndex, Series 145
- JCElevation 146
- JCElevation, MeshBottomColor 146
- JCElevation, Meshed 146
- JCElevation, MeshTopColor 146
- JCElevation, Shaded 146
- JCElevation, ShadedBottomColor 146
- JCElevation, ShadedTopColor 146
- JCElevation, Transparent 146
- JCGridColor 146
- JCGridColor, Color 146
- JCGridColor, DataIndex 146
- JCGridLines 146
- JCGridLines, LineStyle 146
- JCGridLines, PlaneMask 146
- JCLineStyle 147
- JCLineStyle, Cap 147
- JCLineStyle, Color 147
- JCLineStyle, Join 147
- JCLineStyle, Pattern 147
- JCLineStyle, Width 147
- JCPlotCube 147
- JCPlotCube, Background 147
- JCPlotCube, Ceiling 147
- JCPlotCube, Floor 147
- JCPlotCube, Foreground 147
- JCPlotCube, xScale 147
- JCPlotCube, yScale 147
- JCPlotCube, zScale 147
- JCProjection 147
- JCProjection, Contoured 147
- JCProjection, Zoned 148
- JCScatter 148
- JCScatter, DropLines 148
- JCSurface 148
- JCSurface, Solid 148
- JCSurface, xMeshFilter 148
- JCSurface, xMeshShowing 148
- JCSurface, yMeshFilter 148
- JCSurface, yMeshShowing 148
- JCSymbolStyle 148
- JCSymbolStyle, Color 148
- JCSymbolStyle, Shape 148
- JCSymbolStyle, Size 149
- JCValueLabel 149
- JCValueLabel, Label 149
- JCValueLabel, TickOnly 149
- JCValueLabel, Value 149
- JCView3d 149
- JCView3d, Perspective 149
- JCView3d, xRotation 149
- JCView3d, yRotation 149
- JCView3d, zRotation 149
- JCViewport 149
- JCViewport, HorizontalShift 149
- JCViewport, Normalized 149
- JCViewport, PreviewMethod 149
- JCViewport, Scale 149
- JCViewport, VerticalShift 149
- LabelGenerator 33
- Labels 33
- LayoutStyle 32
- levels 22
- MarginGap 33
- Max 22, 29
- meshBottomColor 50
- meshTopColor 50
- Min 22, 29
- minorTick 41
- Name 31
- normalized 101
- Orientation 30
- previewMethod 101
- scale 101
- shadedBottomColor 50
- shadedTopColor 50
- show 28
- Solid 63
- surfaceTopColor 50
- Text 48

- Title 29
- UseDefault 22, 29
- Value 41
- VerticalItemGap 33
- verticalShift 101
- viewing 26
- Visible 30
- zoneData 70

Q

- Quest Software technical support
 - contacting 6

R

- regular grid data 74
- related documents 4
- removeGridColor() 57
- rendering 122
- RenderingAttributes 118
- reset action 103
- responsibility for data 70
- RestrictedAccessException 120
- retained mode 122
- return to default 100
- RGB color
 - specifications 24
- rotate action 103
- rotateEye action 103
- RotateX action 103
- RotateY action 103
- RotateZ action 103
- rotation 100
 - interactive 99
 - JCViewport 101

S

- scale
 - action 104
 - interactive 99
 - JCViewport property 101
 - of axis 38
 - scaling 100
- SCATTER 65
- scatter plot 66
 - 2D 66
 - 3D 66
 - 3D with drop lines 66
 - basic types 66
 - chart 15, 30, 31
 - drop lines 66

- flat charts 66
 - of grid data 31
 - of point data 31
 - programming 65
- scene graph 110
 - programming model 109
 - Locale 110
 - nodes 110
 - view 119
- SceneGraphObject 120
- SceneGraphObject class 114
- Screen3D 119
- set method 27
- setContoured 59
- setElevationDataSource 71
- setFillGraphics() 37
- setLabelGenerator() 42
- setMeshBottomColor 61
- setMeshTopColor 61
- setting Bean properties 28
- setting object properties 16
 - interactively at run-time 17
 - with a Java IDE at design-time 17
 - with Java code 17
- setZoned 59
- setZoneDataSource 71
- shade 50, 57
 - and contoured charts 53
 - and contoured graphs 53
 - and meshed chart 53
 - and meshed graphs 53
 - and zoned charts 54
 - and zoned graphs 54
 - bar chart 32
 - bars 50
 - charts 52
 - contours, zones charts 55
 - contours, zones graphs 55
 - graphs 52
 - meshed, contoured charts 54
 - meshed, contoured graphs 54
 - meshed, contoured, zoned charts 55
 - meshed, contoured, zoned graphs 55
 - meshed, zoned charts 54
 - meshed, zoned graphs 54
 - shaded 93
 - surface chart 32
- shadedBottomColor 58
- shadedBottomColor property 50
- shadedTopColor 58
- shadedTopColor property 50
- Shape3D
 - node 117
 - subclass 115, 116
- SharedGroup subclass 115

- show property 28
- Solid property 63
- Sound subclass 115, 116
- Soundscape subclass 116
- spot light 127
- SQL result set 81
- standard file format 75
- startup checklist 12
- Sun's XML site 78
- support 6, 7
 - contacting 6
 - FAQs 7
 - internationalization 98
- surface 49
 - and bar charts 31
 - chart 15
 - color 32
 - shading 32
 - charts
 - 4D 93
 - creating 4D 93
 - colors 62
 - graphs
 - 4D 93
 - creating 4D 93
 - solid 63
- surfaceTopColor property 50
- Swing event 104
- Swing TableModel
 - loading data 77
- Switch subclass 115
- SwitchRotateAny action 104
- SwitchRotateEye action 104
- SwitchRotateX action 104
- SwitchRotateY action 104
- SwitchRotateZ action 104
- symbol 35
 - color 68
 - shape 68
 - size 68
 - style, controlling 66
- symbolDim 35
- SymbolStyle 31
- system set-up
 - hardware acceleration 111
 - programming with Java 3D API 110

T

- TableModel 77
- technical support 6, 7
 - contacting 6
 - FAQs 7
- terminology 11

- TexCoordGeneration 118
- textDim 35
- Texture 118
- texture mapping 123
- Texture2D 124
- TextureAttributes 118
- TextureLoader 124
- TextureUnitState 118
- Title property 29
- TransformGroup 120
 - subclass 115
- translate action 104
- translation 100
 - interactive 99
 - rotating 99
 - scale 99
 - translation 99
 - zoom 99
- transparency 25
- TransparencyAttributes 119
- Trigger property 25
- types of charts 49
- typographical conventions 3

U

- unit cube 12
- unmapping 105
- unpicking 106
- UseDefault 22, 29
- user input mapping 101
- user interaction
 - default 99
 - features 99
 - programming 99
 - rotating 99
 - scaling 99
 - translation 99
 - zooming 99

V

- Value property 41
- ValueLabels 41
- ValueLabels axis annotation 41
- values method 40
- Vector object 36
- VerticalItemGap property 33
- verticalShiftJCV viewport property 101
- view 119
 - branch graph 120
 - property 26
 - scene graphs 119
- ViewPlatform 122

- subclass 117
- virtual universe 110
- VirtualUniverse object 110
- Visible property 30

X

- x, dragZValue parameter 107
- x, gridValue method 107
- xLabels 70
- xMeshFilter 61
- XML 78
 - constructor 79
 - example data file 80
 - HTML 78
 - primer 78
 - using in JClass 78
 - XSLT 78
- XML sources 69
- XSLT 78

Y

- y, dragZValue parameter 107
- y, gridValue method 107
- yLabels 70
- yMeshFilter 61

Z

- Z-axis restrictions 29
- zone
 - and contoured charts 54
 - and meshed charts 53
 - and meshed graphs 53
 - and shaded charts 54
 - and shaded graphs 54
 - bars 51
 - cell 61
 - charts 52, 54
 - contoured, shaded graphs 54
 - contours, shaded graphs 55
 - data source 22
 - display 58
 - graph 52
 - graphs 54
 - legend display 30
 - meshed, contoured charts 55
 - meshed, contoured graphs 55
 - meshed, shaded charts 54
 - meshed, shaded, contoured charts 55
 - meshed, shaded, contoured graphs 55
 - method 60

- projection 59
 - shaded, contoured charts 55
 - shaded, meshed graphs 54
 - zoned 50, 51, 57, 58, 59, 67, 93, 94, 95
- zoneData property 70
- zoom 100
 - action 104
 - interactive 99

