JClass Chart™

Programmer's Guide & Reference Manual

Version 3.6

JDK 1.0.2, JDK 1.1, JDK 1.1+Swing, and JDK 1.2

The Best Java Charting Solution

KL GROUP

Software Development Productivity

260 King Street East
Toronto, Ontario, Canada M5A 1K3
(416) 594-1026
www.klg.com

February 1999 RefNo: PRGDE-JCCHT/J/361-02/99

Table of Contents

Preface

Introducing JClass Chart

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

JClass Chart may be used in conjunction with KL Group's JClass BWT and JClass LiveTable. JClass BWT provides additional Java components that complement or replace standard AWT components.

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

- Component users, setting JClass Chart properties programmatically
- OO developers, instantiating and extending JClass Chart objects
- Java Bean developers, setting JClass Chart properties using a third-party Integrated Development Environment (IDE)
- Web page designers, setting JClass Chart properties exposed through HTML parameters.

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

Feature Overview

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

- Chart type (Plot, Scatter Plot, Area, Stacking Area, Bar, Stacking Bar, Pie, Hi-Lo, Hi-Lo-Open-Close and Candle)
- Header and footer positioning, border style, text, font, and color.
- Number of data views, each having its own data, chart type, axes, and chart styles
- Flexible data loading from Applets, files, URLs, input streams, and databases.
- Chart styles: line color, fill color, point size, point style, point color
- Legend positioning, orientation, border style, anchor, font, and color

- Chart positioning, border style, color, width, height, and 3-D effect (bar, stacking bar and pie charts only)
- Axis labelling using Point labels, Series labels, Value labels, or Time labels
- Number of X- or Y-axes, each having its own minimum and maximum, axis numbering method, numbering and ticking increment, grid increment, font, origin, axis direction and precision
- Control of user interaction with component including picking, mapping, Chart Customizer, rotation, scaling, translation
- Chart labels that can appear anywhere on the chart, including automatic dwell labels for each point on the chart
- Rich text (fonts, colors, placement, URLs) for any chart element using JCStrings

Assumptions

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

Typographical Conventions Used in this Manual

`Typewriter Font`	■ Java language source code and examples of file contents. ■ JClass Chart and Java classes, objects, methods, properties, constants and events. ■ HTML documents, tags, and attributes. ■ Commands that you enter on the screen.
Italic Text	■ Pathnames, filenames, URLs, programs and method parameters. ■ New terms as they are introduced, and to emphasize important words. ■ Figure and table titles. ■ The names of other documents referenced in this manual, such as *Java in a Nutshell.*
Bold	■ Keyboard key names and menu references.

Overview of Manual

Part I – “Using JClass Chart” describes programming with JClass Chart.

Chapter 1, “Getting Started” provides help with common configuration problems, including CLASSPATH and IDE setup.

Chapter 2, “JClass Chart Basics”, provides a programmer’s overview of JClass Chart. It covers class hierarchy, object containment, terminology, programming basics, and specific issues to be aware of before using JClass Chart.

Chapter 3, “SimpleChart Bean Tutorial“, introduces basic Bean concepts, and guides you through developing a chart application in an IDE or BeanBox.

Chapter 4, “Bean Reference”, is a guide to the different JClass Chart Beans. It illustrates all of the properties available, including the different data loading methods.

Chapter 5, “MultiChart“, is a user’s guide for MulitChart, an advanced charting Bean.

Chapter 6, “Chart Programming Tutorial”, is a tutorial designed to introduce you to JClass Chart programming, It includes examples of common chart programming tasks.

Chapter 7, “Axis Controls”, covers JClass Chart properties used when first setting up your chart, concentrating on axis properties.

Chapter 8, “Data Sources”, is a guide to using different pre-built data sources, and how to use the data source toolkit to create your own.

Chapter 9, “Text and Style Elements”, covers JClass Chart properties used to customize the appearance of a chart, including header/footer, legend and chart styles.

Chapter 10, “Advanced Chart Programming”, looks at programming more advanced aspects of the chart.

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

Appendix A, “JClass Chart Property Listing”, summarizes the properties contained in all of the JClass Chart objects.

Appendix B, “JCString Properties”, describes the types of JCString properties available for adding hypertext, images and text within JClass Chart programs.

Appendix C, “Colors and Fonts”, lists all of the colornames and RGB values that can be used in JClass Chart applications. It also lists all of the fonts and font style constants that can be used.

Appendix D, “HTML Property Reference”, is a listing of all of the HTML properties that can be used by Applets created from JClass Chart.

Related Documents

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

- "*Writing Java Programs*" at *http://www.javasoft.com/docs/programmer.html* and the "*Java Tutorial*" at *http://www.javasoft.com/docs/books/tutorial/index.html* from Sun Microsystems
- *Java in a Nutshell, 2nd Edition* from O'Reilly & Associates Inc.
- Resources for using JavaBeans at *http://www.javasoft.com/beans/resources.html*

However, these documents are not required to develop applications using JClass Chart and Java.

Technical Support

Many of the initial questions you may have are basic installation or configuration problems. Consult this product's *readme* file and Chapter 1, "Getting Started", for help with these types of problems.

KL Group's **Standard Support** plan is included with your purchase and entitles registered users with a valid JClass software license to the following support:

- 30 days of direct technical support via telephone, email or fax.
- FAQ Documents on our Web site.
- JClass Knowledge Base, a searchable collection of information including program samples and problem/resolution documents.
- JClass Forum Newsgroup, where you can communicate with other developers using JClass products around the world.
- Minor bug-fix update releases downloadable from our Web site.

Upgrading to KL Group's **Gold Support with Subscription** plan entitles you to the following *additional* support:

- Unlimited direct technical support for one full year.
- Web-based Express Case Submission form for quickly logging problems; a Customer Support Engineer will contact and assist you directly.
- All product upgrade releases; download from Web site or shipped to you on CD-ROM.

For information on obtaining **Gold Support** for your JClass product, please visit our online store or your JClass reseller. You can also email `sales@klg.com`.

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:

- The type and version of the operating system you are using
- Your development environment and its version

- A full description of the problem including the steps required to duplicate it.

Telephone:	800-663-4723 (toll free in North America) or 416-594-1026 Available Monday – Friday, 9:00 a.m. to 8:00 p.m. Eastern time
Fax:	416-594-1919
Standard Support Email:	jclass_support@klg.com
Express Case Submission Form (Gold Support only)	http://www.klg.com/cgi-bin/webcase.cgi

Other Support Resources

JClass Technical Support (links to Knowledge Base):	http://www.klg.com/cs/tech/jclass/
JClass FAQs:	http://www.klg.com/cs/tech/jclass/faq/
Using JClass in IDEs:	http://www.klg.com/jclass/ides.html

Product Feedback and Annoucements

We are interested in hearing about how you use JClass Chart, 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:

KL Group Inc.
260 King Street East
Toronto, Ontario, M5A 1K3 Canada

Phone: (416) 594-1026
Fax: (416) 594-1919
Email: dev_jclass@klg.com
Internet: news://news.klg.com/klg.forum.jclass

While we appreciate your feedback, we cannot guarantee a response. Please do not use the `dev_jclass` email address for technical support questions.

Occasionally, we send JClass-related product announcments to our customers using an email list. To add yourself to this mailing list, send email with the word "`subscribe`" in the body of the message to jclass_announce-request@klg.com. Visit the KL Group web site at http://www.klg.com for more details.

Part I

Using JClass Chart

1

Getting Started

1.1 Introduction

This chapter covers common configuration issues so you can start using JClass Chart as quickly as possible. Because of the wide variety of Java platforms and development environments, JClass Chart may not be configured correctly for your environment after installation.

Please see the *readme-chart.txt* file included with this release for details on installing JClass Chart and for information on supported Java environments.

1.2 Matching JClass and JDK Versions

Separate versions of JClass Chart are available for specific versions of the Java Platform. The version you use should match the JDK version needed by your application/applet. For example, if you are creating an applet to run in Microsoft Internet Explorer 4.0 (JDK 1.1 platform), use the JClass Chart version for JDK 1.1. Use the following table to determine which version of JClass Chart to use for your application:

Version	Java Platform	Description
JClass Chart 3.6**T**	JDK 1.0.2	■ "Transitional Beans" that provide JDK 1.1-level event APIs for easy migration to JDK 1.1.
JClass Chart 3.6	JDK 1.1	■ Standard "AWT-style" JavaBeans.
JClass Chart 3.6**S**	JDK 1.1 + Swing	■ JavaBeans for JDK 1.1 applications using Swing 1.0.3 components.

Version	Java Platform	Description
JClass Chart 3.6**J**	JDK 1.2	■ JavaBeans for JDK 1.2 applications. ■ Also for JDK 1.1 applications using Swing 1.1.

Each version has the same API and virtually the same features to make it easy for existing applications to migrate to new versions of the Java platform. For clarity, distribution filenames and JAR/ZIP archives contain the full version number in the name, for example, *jcchart360.jar* and *jcchart360S.jar.*

This documentation covers all versions of JClass Chart, noting any differences between versions where they occur.

Determining the JDK and JClass Chart Version

To determine the version of the JDK you are using, enter the following at a command prompt:

```
java -version
```

To determine the version of JClass Chart you are using on your system, run the version program provided:

```
java jclass.chart.JCVersion
```

This program will only run if the CLASSPATH has been set correctly as described in the following section.

1.3 Setting the CLASSPATH Environment Variable

The Java Virtual Machine (JVM) and other applications use the CLASSPATH environment variable to locate user-defined classes. You should ensure that the CLASSPATH points to the location of the JClass Chart classes (and classes you develop). The installation program does this automatically for Windows users; Unix users need to add JClass Chart to the CLASSPATH manually.

Two entries should be part of the CLASSPATH – one specifying the JClass product classes (a JAR or ZIP file located in the product's *\lib* directory), and one specifying the installation directory (necessary to run JClass Chart example and demo programs). You should not need to unzip the JAR/ZIP archive to develop with JClass Chart.

For example, if you installed JClass Chart on a Windows machine in *C:\JClass36*, the CLASSPATH would include the following ([*xxx*] is the product version number):

```
C:\JClass36\lib\jcchart[xxx].jar;C:\JClass36\
```

To determine the current CLASSPATH, enter the following at a command prompt:

Windows – `echo %CLASSPATH%`
Unix – `echo $CLASSPATH`

Some CLASSPATH specification tips:

- Each entry is separated by a semicolon (Windows) or a colon (Unix).
- An entry is typically a root directory to search through for *.class* files (if a class is part of a package, each level in the package is treated as a subdirectory from here), for example, `C:\JClass36`.
- Entries can also specify a JAR or ZIP file containing archived classes, for example, `C:\JClass36\lib\jcchart[xxx].jar`.
- Add a period (.) to the CLASSPATH to include the current directory.
- Setting the CLASSPATH in a startup file causes it to be used when running web browsers and other applications for your entire session.

1.3.1 Setting the CLASSPATH in Windows

The Windows-based setup program automatically adds JClass Chart to the CLASSPATH during installation. The following instructions are provided in case you need to configure the CLASSPATH manually for some reason.

Windows 95 and Windows 98

Add the following statement to your *autoexec.bat* file to include JClass Chart in the CLASSPATH ([*xxx*] is the product version number):

```
set CLASSPATH=%CLASSPATH%;C:\JClass36\lib\jcchart[xxx].jar;
  C:\JClass36;
```

JDK 1.0.2 users: Replace *jcchart[xxx].jar* above with *jcchart[xxx]-classes.zip*.

Restart Windows to make the change take effect.

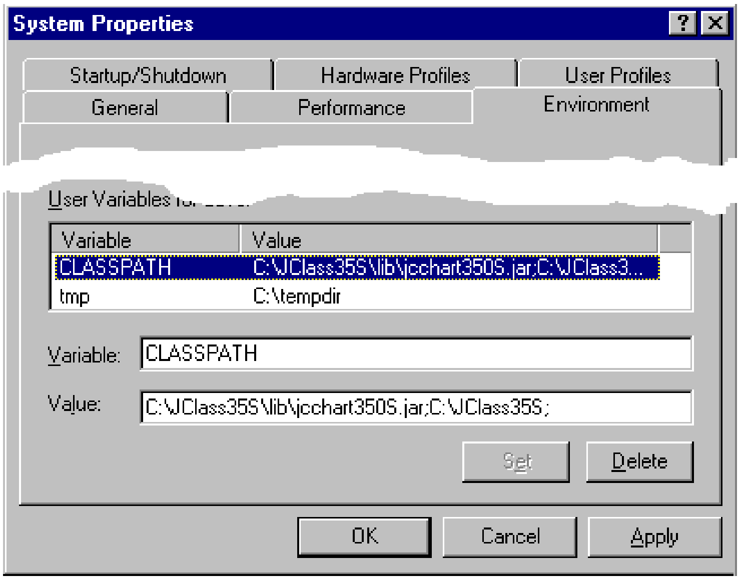
Windows NT (3.51 and higher)

The best way to set environment variables is using the **Control Panel**. Start **Control Panel** and select **System**. Locate the CLASSPATH environment variable (if it doesn't exist, create it). Add the following value to the variable to include JClass Chart in the CLASSPATH ([*xxx*] is the product version number):

```
[existing-classes];C:\JClass36\lib\jcchart[xxx].jar;C:\JClass36;
```

JDK 1.0.2 users: Replace *jcchart[xxx].jar* above with *jcchart[xxx]-classes.zip*.

The following illustrates setting the CLASSPATH on Windows NT; your actual setting may vary or have additional directories/JAR files.



Variable	Value
CLASSPATH	C:\JClass35S\lib\jcchart350S.jar;C:\JClass3...
tmp	C:\tempdir

Variable: CLASSPATH

Value: C:\JClass35S\lib\jcchart350S.jar;C:\JClass35S;

1.3.2 Setting the CLASSPATH in Unix

You must manually configure the CLASSPATH environment variable before you can start using JClass Chart. The CLASSPATH must point to the location of the JClass Chart classes and installation directory (for example */usr/local*).

Add a *setenv* command to your startup file (such as *.cshrc*) to set CLASSPATH to point to the JClass Chart classes, for example ([*xxx*] is the JClass Chart version number):

```
setenv CLASSPATH .:/usr/local/JClass36/lib/jcchart[xxx].jar:
    /usr/local/JClass36
```

JDK 1.0.2 users: Replace *jcchart[xxx].jar* above with *jcchart[xxx]-classes.zip*.

1.3.3 Testing the Installation

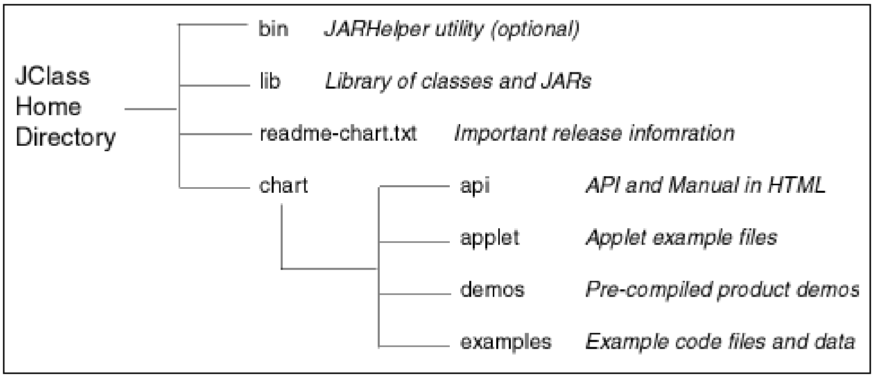
After setting the CLASSPATH environment variable you should verify that it has been configured correctly. The easiest way to test whether you can start programming with JClass Chart is to execute the `JCVersion` class. Enter the following at a command prompt:

```
java jclass.chart.JCVersion
```

If the version number does not match the version just installed, there is probably an older version of JClass Chart listed earlier in the CLASSPATH.

1.4 Installed Files Overview

JClass products install into a single root directory. The directory hierarchy is designed to make it easy to work with multiple JClass products in one location. The following diagram provides an overview of the directory hierarchy created for JClass Chart.


JClass Home Directory
bin
JARHelper utility (optional)
lib
Library of classes and JARs
readme-chart.txt
Important release infomration
chart
api
API and Manual in HTML
applet
Applet example files
demos
Pre-compiled product demos
examples
Example code files and data

Class Library Archives

The *\lib* directory contains the JClass Chart class library archives in JAR or ZIP format. JClass Chart developers can add these files to an IDE, or simply work with them through the JDK. You usually do not need to unzip the archives when programming with JClass Chart.

Your release of JClass Chart may include the following archives ([*xxx*] is the product version number):

jcchart[*xxx*].jar	Standard JClass Chart components.
jcchart[*xxx*]jb.jar	The standard components plus Beans that data bind to Borland JBuilder data source components.
jcchart[*xxx*]vc.jar	The standard components plus Beans that data bind to Visual Café data source components.
jcchart[*xxx*]ds.jar	JClass Chart components that data bind with JClass DataSource data Beans. JClass DataSource is available separately or as part of a JClass product suite from KL Group.
jcchart[*xxx*]-classes.zip	All JClass Chart components for development environments that cannot use JAR files.

See the *readme-chart.txt* file for details on the archives that ship with each version of JClass Chart.

Sample Code

The *jclass\chart\examples\* and *jclass\chart\demos\* directories contain sample Java programs that use JClass Chart. The programs can be executed as either applets or applications. To run as applications, use the Java interpreter, specifying the application class's full package path, for example:

```
java jclass.chart.examples.plot1
```

To run as applets, either open *index.html* in a compatible browser (you may need to unset the CLASSPATH environment variable first) or use the JDK *appletviewer* program.

JDK 1.2 Note: To run JClass Chart sample programs using *appletviewer*, you may need to extract the product JAR file into your *JCLASS_HOME* directory. This is because *appletviewer* in JDK 1.2 does not use the CLASSPATH environment variable. You may also need to use the `-nosecurity` switch, for example:

```
appletviewer -nosecurity index.html
```

Product Documentation

The *jclass\chart\api\* directory contains JClass Chart programming and reference documentation in HTML format. Open *index.html* in a frames-capable web browser to read the documentation.

Version Notes, Compatibility, Known Problems

The *readme-chart.txt* file contains details on version-specific files installed with the JClass Chart version for each JDK platform, compatibility with JDK and browser environments, and changes and known problems with this release.

1.5 Adding JClass Chart to Your IDE

JClass Chart works well with any JavaBeans-compliant Integrated Development Environment (IDE), including Symantec Visual Café, Inprise Borland JBuilder, IBM VisualAge for Java, Sybase PowerJ, and SuperCede for Java.

Once added to the development environment's component palette you can use JClass Chart the same way you use standard AWT or Swing components – adding them to forms, setting initial property values, specifying event-handling, and so on.

All environments provide a way to add components contained in a JAR or ZIP file to their component palette. The exact steps are unique to each environment so the best source for details is the documentation for your development environment. The JClass Chart JAR and ZIP files are located in the *\lib* subdirectory of where you installed JClass Chart.

1.5.1 Using Visual Café with JClass Chart

We recommend installing JClass Chart *after* installing Visual Café; this way, JClass Chart can be added to the Component Library automatically. The setup program copies the JClass Chart JAR file to Visual Café's *\bin\components* directory. (If you install Visual Café after installing JClass Chart, you can add the JAR to the Component Library manually as described in the Visual Café help.)

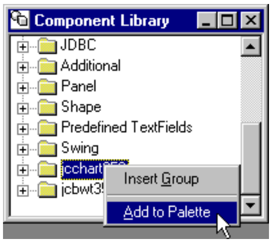
Replacing the Bundled JClass Components

It is important to note that installing this release **does not** automatically replace the older JClass BWT, JClass Chart, and JClass LiveTable components that are included with Visual Café (located in *\KLGroup\klg.jar*). If the bundled JClass components have been added to the Component Library, a newer version will not be shown in the Component Library.

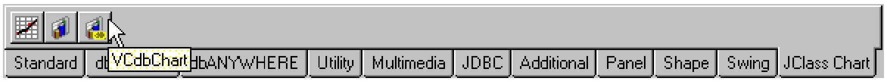
To force Visual Café to replace the old JClass components in the Component Library, you must explicitly add *jcchart[xxx]vc.jar* to the Component Library (**Insert | Component into Library...**).

Adding JClass Chart to the Component Palette

When JClass Chart is in the Component Library, you can add its components to the Component Palette to make them convenient to use. The following steps describe one easy way to add a new palette tab containing all of the JClass Chart components:



1. Display the Component Library window if it is not already visible (**View | Component Library**)
2. Right-click the "jcchart[*xxx*]vc" folder and select **Add to Palette** from the popup menu. Visual Café creates a new tab on the Component Palette and adds all of the JClass Chart components to it.
3. You can rename the tab to make it easier to read. To do this, right-click the Component Palette, select **Customize Palette...** from the popup menu, and change the name of the "jcchart[*xxx*]vc" folder to "JClass Chart".



Upgrading to a Newer Version of JClass Chart

Visual Café only allows one version of a component to be listed in the Component Library, so when you install a newer version of JClass Chart, it automatically replaces the older version in the Component Library (except for the version included with Visual Café; see Replacing the Bundled JClass Components for details).

When you reopen your project, it seamlessly uses the latest version of JClass Chart. There should generally be no problem using a newer version of JClass Chart with an existing application. However, if you do experience problems, you can revert back to the previous version by moving the new version's JAR file out of the *\bin\components* directory (previous versions' JARs are not deleted).

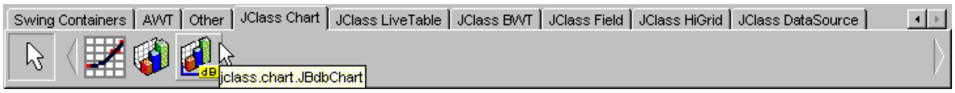
Note: You must add JClass Chart to the Component Palette again manually when you install a newer version.

Removing JClass Chart from the Component Library

Using the **Add/Remove Programs** dialog in the **Control Panel** does not remove JClass Chart from Visual Café. You must manually delete the JClass Chart JAR file from Visual Café's *\bin\components* directory and manually remove the JClass Chart tab from the Component Palette.

1.5.2 Using JBuilder with JClass Chart

We recommend installing JClass Chart *after* installing Borland JBuilder; this way, JClass Chart is added to the Component Palette automatically. If you install JBuilder after installing JClass Chart, you can add the JAR file to the Palette manually (**Tools | Configure Palette...**) as described in the JBuilder help.



Upgrading to a Newer Version of JClass Chart

When you install a newer version of JClass Chart, the Component Palette is automatically updated to use the new version. However, existing JClass Chart projects need to be reconfigured to use the new version, as outlined below:

1. With your project open, display the Project Properties dialog (**File | Project Properties...**).
2. Edit the Java Libraries list on the Paths tab to use the new version of the JClass Chart JAR file.
3. Save your project files.
4. Similarly, edit the default Java Libraries list (**Tools | Default Project Properties...**) to use the new version of the JClass Chart JAR file for new projects.

See the JBuilder help for complete details. There should generally be no problem using a newer version of JClass Chart with an existing application. However, if you do experience problems, you can revert back to the previous version in the Project Properties dialog.

Removing JClass Chart from JBuilder

Using the **Add/Remove Programs** dialog in the **Control Panel** does not remove JClass Chart from JBuilder. You must manually configure JBuilder to remove all references to JClass Chart:

- Remove JClass Chart from the default Java libraries list using the Default Project Properties... dialog (**Tools | Default Project Properties...**).

- To remove the JClass Chart tab from the Palette, right-click the palette, select **Properties...**, and click the **Remove** button.

1.6 Java and JavaBeans Basics

Java is both a compiled and an interpreted language. After writing a Java program using a text editor, save it as a source file with the extension *.java*. When this source file is run through the Java compiler, it compiles the file into a *.class* file. Unlike *.exe* files, these compiled *.class* files are not directly executable under any operating system, because they do not contain machine-language code that can be understood directly by the microprocessor. Instead, they are compiled into a byte-code format consisting of machine-language instructions designed for a virtual microprocessor. This virtual microprocessor is the Java Virtual Machine, which interprets the byte-code into a machine-language code that can be understood by your system's microprocessor. As long as the Java Virtual Machine software exists for a computing platform, any Java programs you create will be able to run on that platform.

If the Java compiler and CLASSPATH are properly configured, you can compile a Java program by running the Java compiler at the command prompt, for example:

```
javac MyJavaProgram.java
```

Java Applications and Applets

Java programs are usually one of two types: stand-alone applications and applets. Stand-alone applications can be run directly on a system containing the Java interpreter or Java runtime environment, while applets can be added to web pages for execution by a Java-compatible browser. JClass components can be used to create both types of Java programs.

JClass Chart and JavaBeans

JavaBeans$^{(TM)}$ is the software component model for Java. Introduced in JDK 1.1, the JavaBeans specification enables developers to create and use platform-independent, reusable software components on a wide variety of platforms and development environments. JClass Chart components are JavaBeans; they follow standard API naming conventions, the JavaBeans event model, and can easily be integrated with Java IDEs.

A good source of general information on Java and JavaBeans is the Frequently Asked Questions (FAQ) list that can be found at the JavaSoft Web site at *http://www.javasoft.com/products/jdk/faq.html* and *http://www.javasoft.com/beans/FAQ.html* respectively.

2

JClass Chart Basics

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

2.1 Chart Areas

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

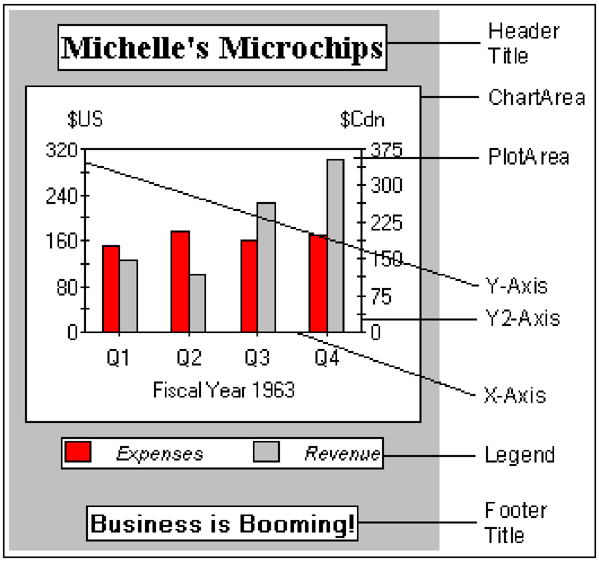


Figure 1 Elements contained in a typical chart

2.2 Chart Types

JClass Chart can display data as one of ten basic chart types: Plot, Scatter Plot, Area, Stacking Area, Bar, Stacking Bar, Pie, Hi-Lo, Hi-Lo-Open-Close, and Candle. It is also possible to simulate more specialized types of charts using one of these basic types.

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

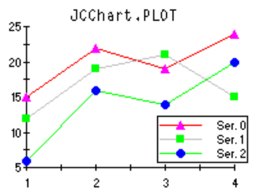
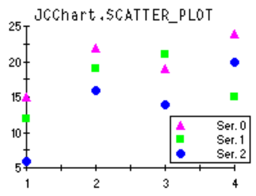
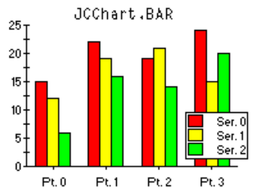
Chart Type	Array data	General data	Notes
	✔	✔	**Plot** Draws each series as connected points of data. ■ When using Array data, X-values shared across series ■ Series appearance determined by chart style line color, symbol shape, size and color properties
	✔	✔	**Scatter Plot** Draws each series as unconnected points of data. ■ When using Array data, X-values shared across series ■ Series appearance determined by chart style symbol shape, size and color properties
	✔		**Bar** Draws each series as a bar in a cluster. The number of clusters is the number of points in the data. Each cluster displays the *n*th point in each series. ■ X-axis generally annotated using Point-labels ■ Series appearance determined by chart style fill color and image properties ■ 3D effect available using depth, elevation and rotation properties

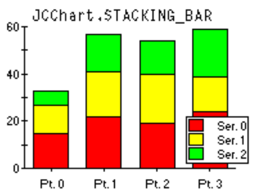
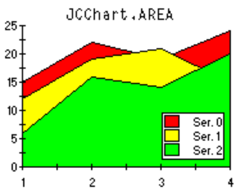
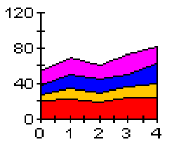
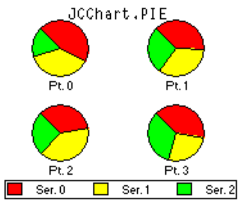
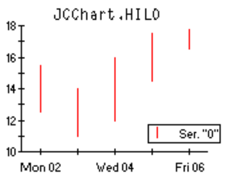
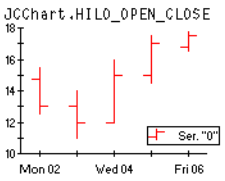
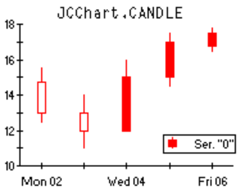
Chart Type	Array data	General data	Notes
	✔		**Stacking Bar** Draws each series as a portion of a stacked bar cluster, the number of clusters being the number of data points. Each cluster displays the *n*th point in each series. Negative Y-values are stacked below the X-axis. ■ X-axis generally annotated using Point-labels ■ Series appearance determined by chart style fill color property ■ 3D effect available using depth, elevation and rotation properties
	✔	✔	**Area** Draws each series as connected points of data, filled below the points. Each series is layered over the preceding series. ■ When using Array data, X-values shared across series ■ Series appearance determined by chart style fill color property
	✔	✔	**Stacking Area** Draws each series as connected points of data, filled below the points. Places each Y-series on top of the last to show the area relationships between each series and the total. ■ When using Array data, X-values shared across series ■ Series appearance determined by chart style fill color property
	✔		**Pie** Draws each series as a slice of a pie. The number of pies is the number of points in the data (values below a certain threshold can be grouped into an *other* slice). Each pie displays the *n*th point in each series. ■ Pies are annotated with Point-labels only ■ Series appearance determined by chart style fill color property ■ 3D effect available using depth, elevation and rotation properties
	✔		**Hi-Lo** Draws *two* series together as a "High-low" bar. The points in each series define one portion of the bar: 1st series — points are the "high" value 2nd series — points are the "low" value ■ Appearance determined by chart style line color property in **first** series of the two

Chart Type	Array data	General data	Notes
	✔		**Hi-Lo-Open-Close** Similar to Hi-Lo, but draws *four* series together as a "High-low-open-close" bar. The additional series' points make up the other components of the bar: 3rd series — points are the "open" value 4th series — points are the "close" value ■ Appearance determined by chart style line color and symbol size properties in **first** series of the four
	✔		**Candle** A special type of Hi-Lo-Open-Close chart, draws four series together as a "candle" bar. ■ *Simple* candle appearance determined by chart style line color, fill color, and symbol size properties in **first** series of the four ■ *Complex* candle appearance determined by different chart style properties from **each** series of the four

2.3 Loading Data

Data is loaded into a chart by attaching one or more chartable 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 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, enabling you to read data from an input stream, from a file, from a URL, databases and from HTML applet `<PARAM>` tags. Loading data from a databases is called 'data binding'. You can also create your own data sources. See Data Sources for more information on loading data, data binding, and creating your own data sources.

2.4 Setting and Getting Object Properties

There are four ways to set (and retrieve) JClass Chart properties:

1. By calling property `set` and `get` methods in a Java program
2. By specifying applet properties in an HTML file
3. By using a Java IDE at design-time (JavaBeans)
4. By using the Chart Customizer at run-time

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

Note: In most cases, you need to understand the chart's object containment hierarchy to access its properties. Use the containment diagram in the previous section to determine how to access the properties of an object.

2.4.1 Setting Properties with Java Code

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

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

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

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

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

For detailed information on the properties available for each object, consult the online HTML API reference documentation.

2.4.2 Setting Applet Properties in an HTML File

Another way to set chart properties, particularly appropriate for applets, is in an HTML file. Applets built with JClass Chart automatically parse applet `<PARAM>` tags and set the chart properties defined in the file. (A pre-built applet called *JCChartApplet.class* is provided with JClass Chart). Even standalone Java applications can save the values of chart properties to an HTML file, which can serve as a useful debugging tool.

Using HTML to set properties has the following benefits:

- Speed – see the effect of different property values quickly without recompiling.
- Flexibility – use a single applet class to create many different kinds of charts simply by varying HTML properties; end-users can modify HTML properties to suit their own needs.

Chart properties are coded in HTML as applet `<PARAM>` tags. The `NAME` element of the `<PARAM>` tag specifies the property name; the `VALUE` element specifies the property value to set.

The following example HTML file supplies the chart's data in the applet:

```
<HTML>
<HEAD><TITLE>plot1</TITLE></HEAD>
<BODY>
<CENTER><H2>plot1</H2></CENTER>
<APPLET CODE=jclass/chart/JCChartApplet.class CODEBASE="../../.."
    HEIGHT=300 WIDTH=400>
<PARAM NAME=data VALUE="
```

```
        ARRAY 2 4
        # X-values
        1.0 2.0 3.0 4.0
        # Y-values
        150.0 175.0 160.0 170.0
        # Y-values set 2
        125.0 100.0 225.0 300.0
    ">
    </APPLET>
    </BODY>
    </HTML>
```

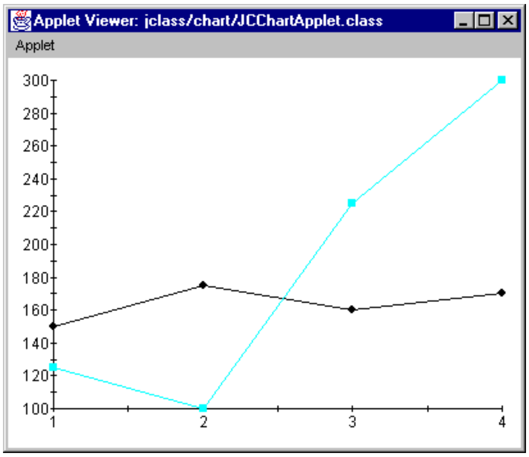


Figure 2   Chart applet displaying data specified in its HTML file

The easiest way to create a set of HTML properties is to use the JClass Chart Customizer to save the property values to an HTML file. For more details, see the "The Chart Customizer" section in this chapter. A full listing of the syntax of JClass Chart properties when used in HTML files can be found in Appendix D: HTML Property Reference. Many example HTML files are located in the *JCLASS_HOME/jclass/chart/applet/* directory.

2.4.3 Setting Properties with a Java IDE at Design-Time

JClass Chart can be used with a Java Integrated Development Environment (IDE), and its properties can be manipulated at design time. Consult the IDE documentation for details on how to load third-party Bean components into the IDE.

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 documentation for complete details.

2.4.4 Setting Properties Interactively at Run-Time

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

For details on enabling and using the Customizer, see The Chart Customizer later in this chapter.

2.5 Other Programming Basics

Working with Object Collections

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

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

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

Note that the index refers to the first element of a collection. Also, note that by default `JCChartArea` contains one element in `XAxis` and one in `YAxis`.

Calling Methods

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

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

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

2.6 JClass Chart Inheritance Hierarchy

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

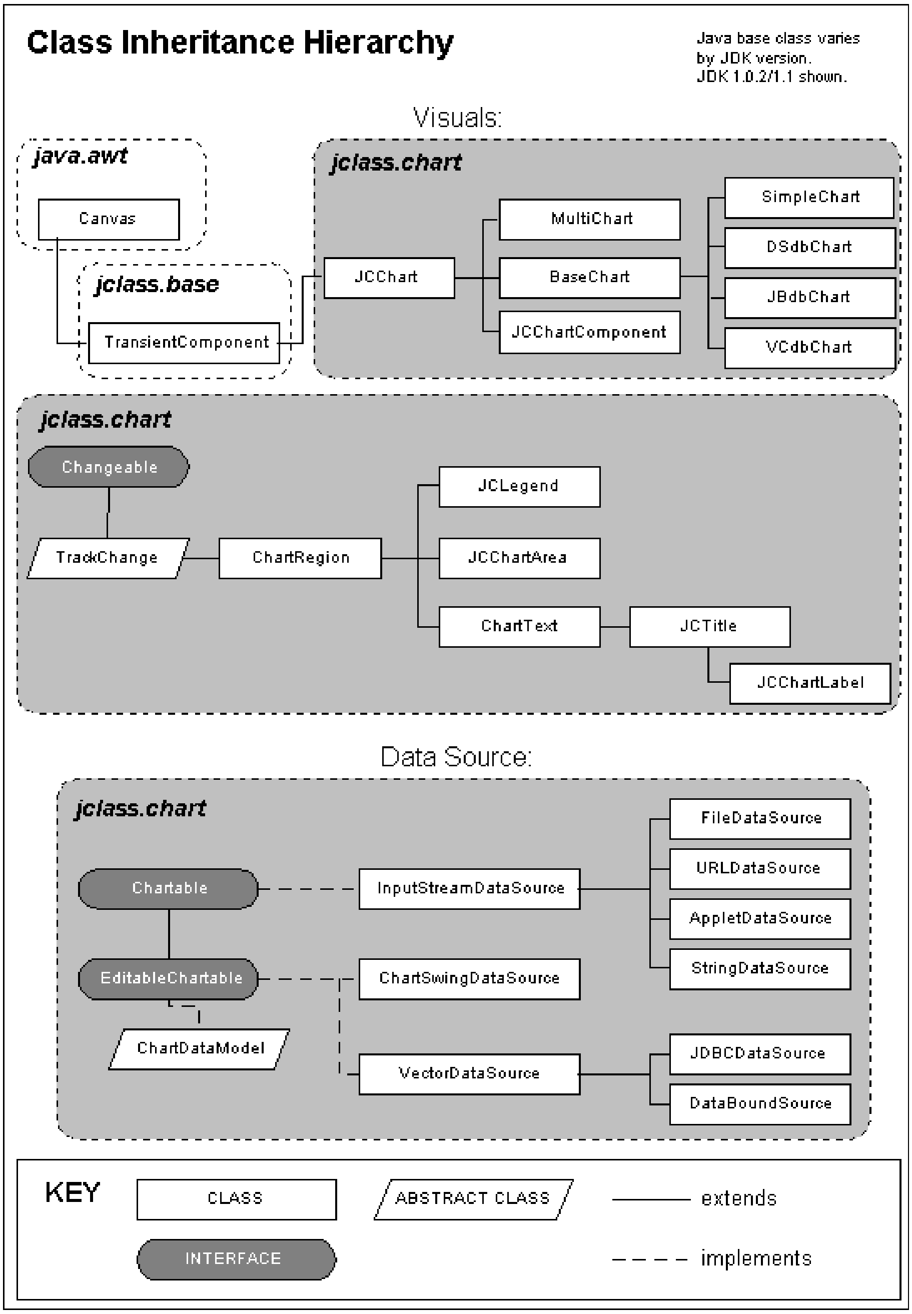


Figure 3 Class hierarchy of the jclass.chart package

JClass Chart has a different Java base class depending on the JDK version used. For example, JClass Chart for JDK 1.1 + Swing is subclassed from `com.sun.java.swing.JComponent`, so it inherits the capabilities provided by the Swing component set. The following diagram shows the Java base class for each Java version supported.

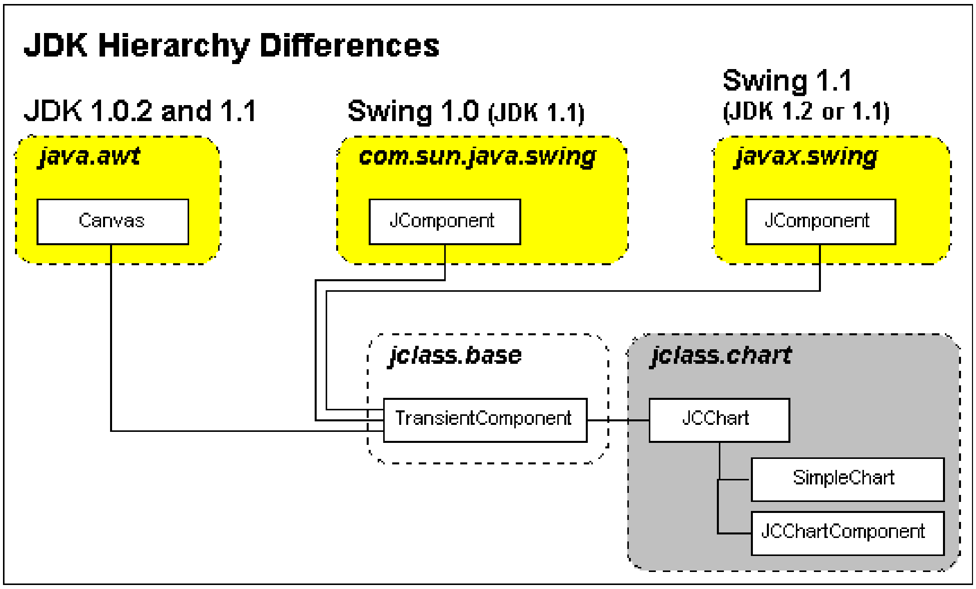
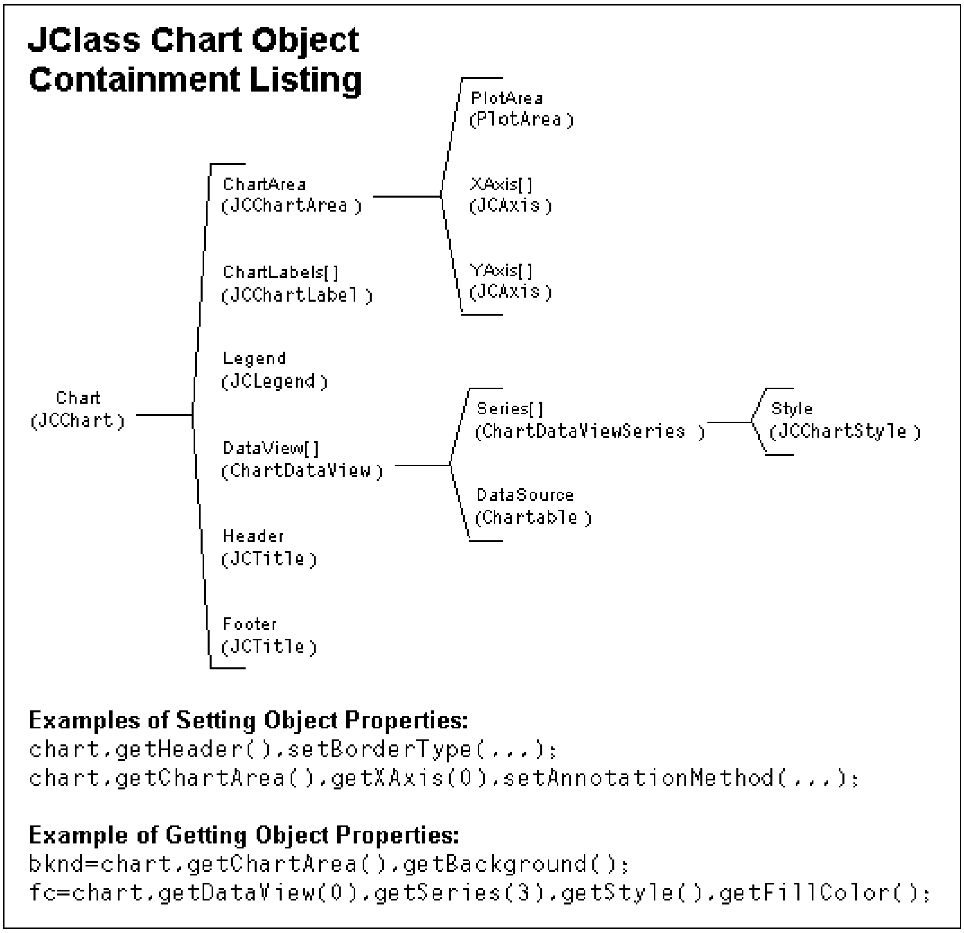


Figure 4 Java base class used by each version of JClass Chart

2.7 JClass Chart Object Containment

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



Examples of Setting Object Properties:

```
chart.getHeader().setBorderType(...);
chart.getChartArea().getXAxis(0).setAnnotationMethod(...);
```

Example of Getting Object Properties:

```
bknd=chart.getChartArea().getBackground();
fc=chart.getDataView(0).getSeries(3).getStyle().getFillColor();
```

Figure 5   Objects contained in a chart — traverse contained objects to access properties

`JCChart` (the top-level object) manages header and footer `JCTitle` objects, a legend (`JCLegend`), and the chart area (`JCChartArea`). The chart also contains a collection of data view (`ChartDataView`) objects and can contain a collection of chart label (`JCChartLabel`) objects.

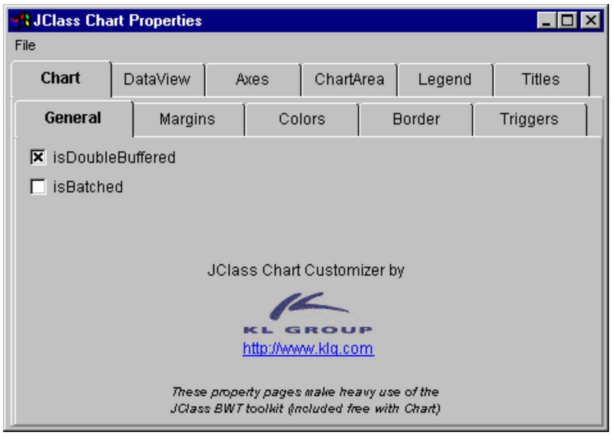
The chart area contains most of the chart's actual properties because it is responsible for charting the data. It also contains and manages a collection of X-axis (`JCAxis`) objects and Y-axis (`JCAxis`) objects (one of each by default).

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

But the chart does not own the data itself, 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's data source model, see Chapter 8, Data Sources.

2.8 The Chart Customizer

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



[✓ isDoubleBuffered]
isBatched

JClass Chart Customizer by

These property pages make heavy use of the JClass BWT toolkit (included free with Chart)

Figure 6 The JClass Chart 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.

2.8.1 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` and `Trigger` properties, for example:

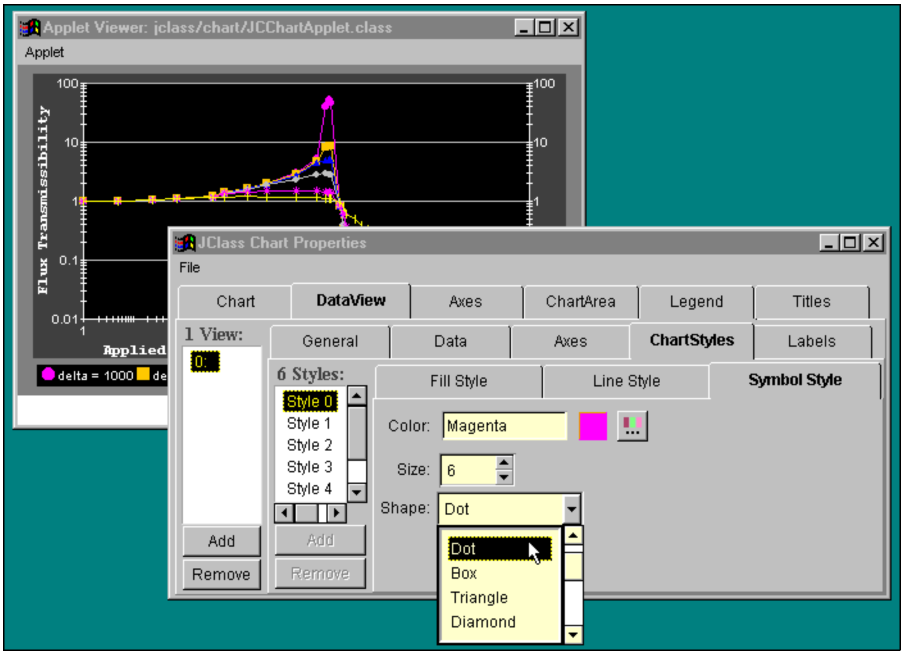
```
chart.setAllowUserChanges(true);
chart.setTrigger(0, new EventTrigger(Event.META_MASK,
                                        EventTrigger.CUSTOMIZE);
```

To display the Customizer once it has been enabled, move the mouse over the chart and click the *secondary* mouse button; that is, the button on your system that displays popup menus, for example:

- Windows – Right mouse button
- UNIX – Middle mouse button

2.8.2 Editing and Viewing Properties

1. Select the tab that corresponds to the element of the chart that you want to edit. Tabs contain one or more inner tabs that group related properties together. Select inner tabs to narrow down on 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.



Color: Magenta
Size: 6
Shape: Dot

Figure 7 Editing a sample chart with the Customizer

As you change property values, the changes are immediately applied to the chart and displayed. You can make further changes without leaving the Customizer. However, once you have changed a property the only way to "undo" it 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).

2.8.3 Saving Customized Charts

Changes made with the Customizer are lost when the program terminates. However, you can save the current values of all chart properties to an HTML file. You can then view the generated HTML file, edit the properties further if you wish, and use the HTML file to create a chart with those property settings.

To save a customized chart, select **SaveAs** from the **File** menu on the Customizer, enter a filename in the dialog, and click the Save button.

2.9 Distributing Applets and Applications on a Web Server

Once you have finished programming your Java applet or application, you will undoubtedly want to distribute it to your users. A common method of applet and application distribution is with your Web server. Here is a brief overview of how to deploy applets and applications, as well as reduce the size and customize the contents of the deployment archive[1].

2.9.1 Publishing an Applet on a Web Server

You can distribute your applet by putting the Web pages that contain it onto your Web server. Distributing your applet this way involves:

- creating directories for your JClass archive, HTML and class files
- copying the required JClass archive files to the Web server
- setting a CLASSPATH on the Web server
- copying the HTML and class files to the Web server
- ensuring that the HTML files properly reference the JClass archive and class files

Install the JClass Archives on the Server

First, you need to make sure that your CLASSPATH is not set. Although you will need to set it later when adding applets to the server, keep it undefined for now.

Create a JClass directory on your Web server (e.g. *\JClassLib*, just below the root document directory). This directory holds all of the archives that came with your JClass products.

1. Although the term "archive" has a somewhat ambiguous and flexible definition, for the purpose of this section, it refers to the JClass product JAR files.

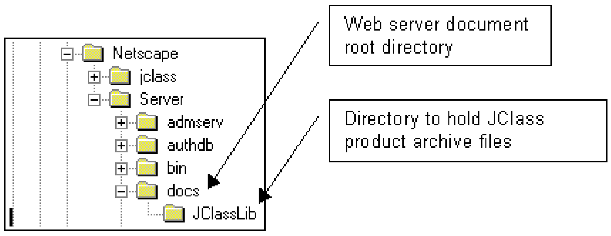

Netscape
jclass
Server
admserv
authdb
bin
docs
JClassLib
Web server document root directory
Directory to hold JClass product archive files

Figure 8 Example: suggested JClass archives folder name and location

Copy the JClass archive files to the newly created *\JClassLib* directory. The number and version of archives copied over, depend on which JClass products you own. These JAR files are found in the *\lib* directory of your JClass installation. Please refer to the Installed Files Overview section earlier in chapter 1 for more information about these files.

Preparing the directory for the applet

Create a directory for the applet classes and their HTML pages. It is important to keep the directory structure identical to the one found in the original location of the classes.

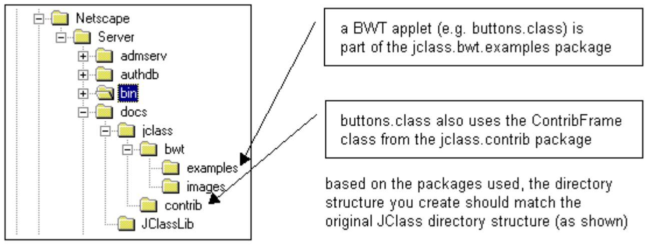

Netscape
Server
admserv
authdb
bin
docs
jclass
bwt
examples
images
contrib
JClassLib
a BWT applet (e.g. buttons.class) is part of the jclass.bwt.examples package
buttons.class also uses the ContribFrame class from the jclass.contrib package
based on the packages used, the directory structure you create should match the original JClass directory structure (as shown)

Figure 9 Example: proper applet class directory structure (using BWT)

Set a CLASSPATH on Your Web Server

If the applet reads local files from the Web server, the CLASSPATH needs to include the directory in which these files are located. As an example, if your applet uses images, the CLASSPATH needs to point to that images directory.

Install Your Applet Classes and HTML files on Your Web Server

Now that the directories have been created with the correct structure, you can copy over all of the required class and HTML files. The directories in which the class files are copied must be the same as the ones from where they are being copied. The HTML files can be placed together in a different location from the associated HTML files (as a suggestion, either the *\JClassLib* or *\JClass* will work fine), since they can point to class files in different locations.

Since your HTML files contain a JClass applet, and they might be located in a different directory from the associated class files, there are certain attributes that must be used to ensure that the file points to the proper JClass archive, class and location.

- `<ARCHIVE>`: The value given for this attribute is the path or URL of the JClass product archive (ZIP or JAR) that the applet requires to run.
- `<CODEBASE>`: You will need this if your applet is in a package or uses classes that are in other packages. The value of this attribute points to the 'top' of the directory structure that contains these classes and packages.
- `<CODE>`: The value of this attribute points to your applet class file.

Any printed or online HTML reference can provide more in–depth information about these attributes. Please refer to it if you need to.

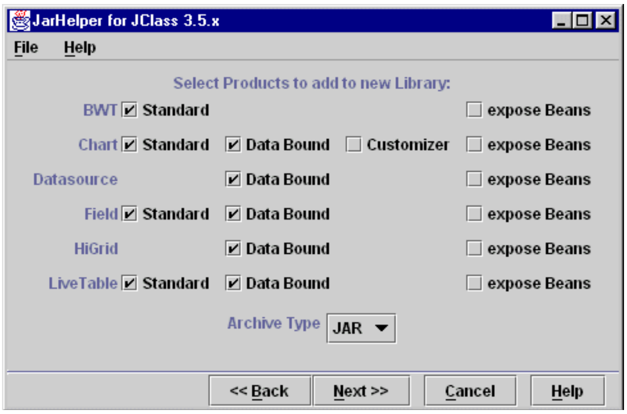
For troubleshooting information about the above procedures, please refer to the JClass Knowledge Base on KL's Web site support area, and perform an online search for *Publishing JClass products on a Web Server*.

2.9.2 Using JarHelper to Customize the Deployment Archive

Deploying your applet or application does not end with copying the required class and HTML files to your Web server; the size of the archive should also be a consideration. The size of the archive, and its related download time are important factors to consider when deploying your applet or application on a Web server.

JarHelper is a utility that allows you to customize and reduce the size of the deployment archive. Using JarHelper, you can combine different JClass product JARs. As well, you can also choose which of the components found within one or more JClass product JARs will be included in the deployment archive. JarHelper takes the selected components of the JClass JAR(s), and creates a new, smaller file, which results in faster download times.

For example, you can use JarHelper to exclude the Bean property editors that are only useful during development, to significantly reduce the size of the deployment archive.


JarHelper for JClass 3.5.x
File
Help
Select Products to add to new Library:
Archive Type
JAR
<< Back
Next >>
Cancel
Help

BWT	✓ Standard			☐ expose Beans
Chart	✓ Standard	✓ Data Bound	☐ Customizer	☐ expose Beans
Datasource		✓ Data Bound		☐ expose Beans
Field	✓ Standard	✓ Data Bound		☐ expose Beans
HiGrid		✓ Data Bound		☐ expose Beans
LiveTable	✓ Standard	✓ Data Bound		☐ expose Beans

Figure 10 JarHelper's JClass product and component selection screen

JClass JarHelper comes with the JClass Enterprise Suite, and is installed automatically with the rest of the bundle's products. It is also available for download from KL Group's Web site for licensees of any JClass product.

Please refer to the *readme-jarhelper.txt* file for JDK and Swing requirements and installation.

Running JarHelper

Windows 95 and NT: Using the **Start** menu, navigate to the JClass JarHelper program group and select the **JarHelper.bat (DOS)** icon. You can also run JarHelper from a command line; the batch file is located in *JCLASS_HOME\bin\JarHelper.bat.*

Unix: Execute the shell script located in the *$JCLASS_HOME/bin* directory from a command prompt.

Using JarHelper

For more information about using JarHelper to create new JARs, please consult its online documentation.

3

SimpleChart Bean Tutorial

3.1 Introduction to JavaBeans

JClass Chart components are JavaBean-compliant. The JavaBeans specification makes it very easy for a Java Integrated Development Environment (IDE) to "discover" the set of properties belonging to an object. The developer can then manipulate the properties of the object easily through the graphical interface of the IDE when constructing a program.

The three main characteristics of a Bean are:

- the set of properties it exposes
- the set of methods it allows other components to call
- and the set of events it fires

Properties control the appearance and behavior of the Bean. Bean methods can also be called from other components. Beans fire events to notify other components that an action has happened.

3.1.1 Properties

Under the new model, "properties" are the named method attributes of a class that can affect its appearance or behavior. Properties that are readable have a "get" method which enables the developer to read a property's value, and those properties which are writable have a "set" method which enable a property's value to be changed.

For example, the `JCAxis` object in JClass Chart has a property called `AnnotationMethod`. This property is used to control how an axis is labelled. To set the property value, the `setAnnotationMethod()` method is used. To get the property value, the `getAnnotationMethod()` method is used.

For complete details on how JClass Chart's object properties are organized, see "JClass Chart Object Containment" and "Setting and Getting Object Properties" in the JClass Chart Basics chapter.

Setting Bean Properties at Design-Time

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

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 documentation for complete details.

3.1.2 New Event Model

Events are a mechanism used to propagate state change notifications between a source object and one or more target listener objects. Events are typically used within windowing toolkits for delivering notifications of such things as mouse or keyboard actions, or other programmatically-defined actions.

In the JDK 1.0.2, the behavior of Java programs was typically modified by subclassing and overriding event handling classes. Subclassing overrode the default behavior of a method. This approach was clumsy because it required the programmer to have a thorough understanding of base class methods and internals before writing any code.

Bean-compliant JClass programs (like JClass Chart) provide the means for an application to be notified when an event occurs through event listeners. It works like this: if a component is acted upon by the user or from within the program, a `JCFooEvent` is fired. The `JCFooListener` (which has been registered by calling `addFooListener()` on the component) receives the instance and enacts the action to be taken. The programmer uses the `JCFooListener` to define what action or actions should take place when it receives the `JCFooEvent`.

3.2 SimpleChart Tutorial

This tutorial guides you through the development of an application that uses `SimpleChart` to chart the financial information of “Michelle’s Microchips”. It is a good starting point for learning basic JClass Chart features. To explore more advanced features of JClass Chart, however, we recommend that you use the MultiChart Bean.

The tutorial does not cover all of the properties available in SimpleChart. For a complete reference, see the Bean Reference. The screen captures have all been taken from Sun’s BeanBox and will differ slightly from your IDE’s appearance.

3.2.1 Steps in this Tutorial

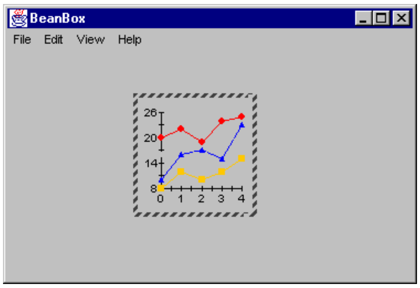
This tutorial has eight steps:

1. Create a new application in your IDE and add a container
2. Put a SimpleChart object into the container
3. Load the data for Michelle’s Microchips
4. Add a header, footer, and legend
5. Add point labels to the x-axis
6. Change the background color to white
7. Set the chart type to bar, and add 3D effects
8. Compile and run the application

Step 1: Create the ‘Michelle’ Application
Create a new application in your IDE and add a container to hold a `SimpleChart` object. In most IDE’s this will be a frame or a panel. See your IDE’s documentation for instructions on creating a basic application and adding a container.

Step 2: Put a SimpleChart Object into the Container
With the container displayed in design mode, click the `SimpleChart` icon and place a `SimpleChart` object into the container’s area. See your IDE’s documentation for details on placing objects into a container. The `SimpleChart` icon looks like this:

In your container object, you should now see a basic chart area with an x- and y-axis, like this:

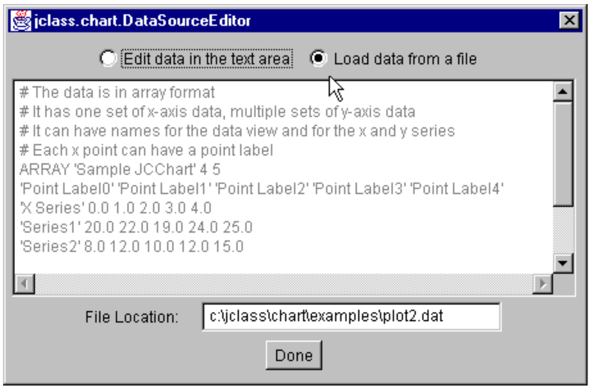


If you open your property list (the window that displays the Bean's properties) with the `SimpleChart` area selected, you should see the property editors that are available in `SimpleChart`.

Step 3: Load Data from a File

This tutorial uses data from a file named *plot2.dat,* contained in the *JCLASS_HOME/jclass/chart/examples/* directory. To load *plot2.dat* into `SimpleChart`, bring up the custom data source editor by clicking on the `data` property:

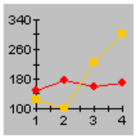
The data source editor provides two methods for loading data: editing data in the text area, or loading data from a file. For Michelle's Microchips, click the **Load data from a file** radio button. Then, enter the full path name of *plot2.dat* in the **File Location** field. After you click **Done**, you should see the data displayed in the chart area as follows:



Edit data in the text area [✓ Load data from a file]

```
# The data is in array format
# It has one set of x-axis data, multiple sets of y-axis data
# It can have names for the data view and for the x and y series
# Each x point can have a point label
ARRAY 'Sample JCChart' 4 5
'Point Label0' 'Point Label1' 'Point Label2' 'Point Label3' 'Point Label4'
'X Series' 0.0 1.0 2.0 3.0 4.0
'Series1' 20.0 22.0 19.0 24.0 25.0
'Series2' 8.0 12.0 10.0 12.0 15.0
```

File Location: c:\jclass\chart\examples\plot2.dat



What's in plot2.dat?

Plot2.dat has financial information for Michelle's Microchips, formatted for the file data source method of data loading. `SimpleChart` accepts only *.dat* files, or modifications to the default data in the editor. For more information on creating a file data source, see, File Data Source in the Data Sources chapter.

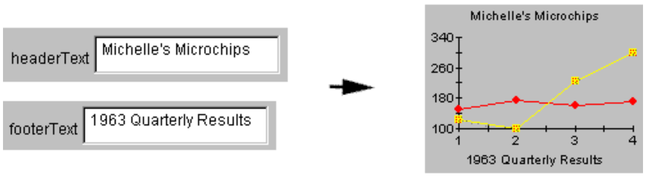
The content of *plot2.dat* is:

```
ARRAY '' 2 4
'Q1' 'Q2' 'Q3' 'Q4'
'' 1.0  2.0  3.0  4.0
'Expenses' 150.0 175.0 160.0 170.0
'Revenue' 125.0 100.0 225.0 300.0
```

JClass Chart also has other Beans which allow you chart data from a database easily. See the Bean Reference for more information.

Step 4: Add a Header, Footer, and Legend

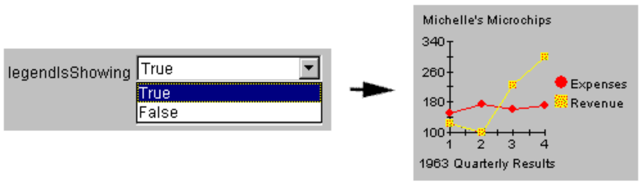
Enter "Michelle's Microchips" in the `headerText` property editor and "1963 Quarterly Results" in the `footerText` property editor:



headerText: Michelle's Microchips

footerText: 1963 Quarterly Results

To add the legend, set the `legendIsShowing` property to `true`. The legend items and text are taken from information in the data source. Notice how the plot area is resized to accommodate the legend. You may have to resize your chart area to accommodate the changes:



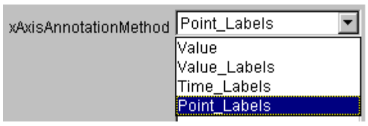
legendIsShowing: True (options: True, False)

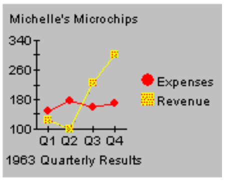
For more information on legend properties, see Legends, in the Bean Reference.

Step 5: Add Point Labels to the x-axis

By default, `SimpleChart` annotates the axes using with values. You can change the annotation to show point labels or time labels.

For Michelle's Microchips, change the x-axis annotation from values to point labels. Do this by setting the `xAxisAnnotationMethod` property to `Point_Labels`:


xAxisAnnotationMethod Point_Labels

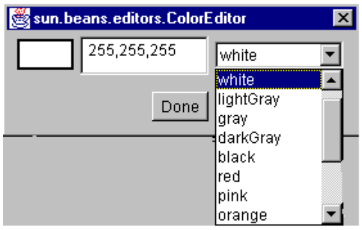


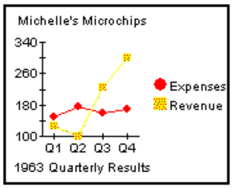
You should now see "Q1", "Q2", "Q3", and "Q4" on the x-axis. These labels are contained in the *plot2.dat* file, and come up automatically when `Point_Labels` is selected. For more information on axis annotation, see Axis Properties, in the Bean Reference.

Step 6: Change the Background Color

To change the background color to white, click the `background` property to bring up your color editor:

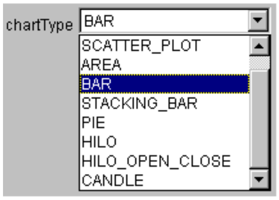
The custom color editor used by your IDE will differ from the BeanBox. Select pure white from the options on your color editor:


sun.beans.editors.ColorEditor
255,255,255 white

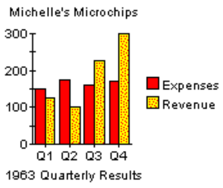


Step 7: Change to Bar Chart and add 3D Effects

You can select from nine chart types using the `chartType` property editor (see, Chart Types, for a complete list). For Michelle's Microchips, select the `BAR` type:


chartType BAR

SCATTER_PLOT
AREA
BAR
STACKING_BAR
PIE
HILO
HILO_OPEN_CLOSE
CANDLE

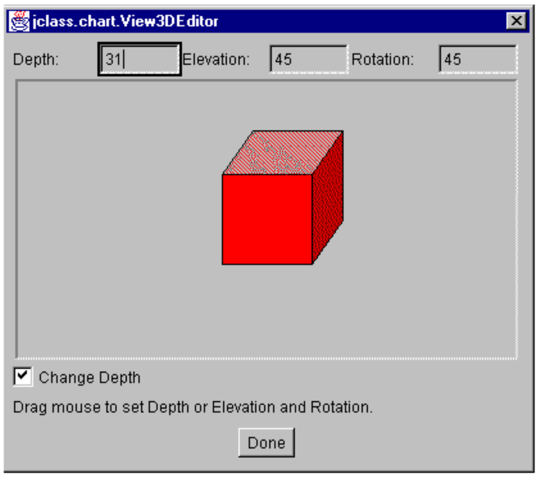


To add three-dimensional visuals to your chart, click the `view3D` property to bring up the **View3DEditor**:

view3D Click to edit...

There are two main settings in the **View3DEditor** (below): depth, and combined elevation and rotation. They are both set by dragging the box in the editor with a mouse.

First, drag the square with your mouse until you have an Elevation of 45 and a Rotation of 45. Second, check the **Change Depth** box, and drag the red square until it has a depth of 31. Click **Done** to set the changes:

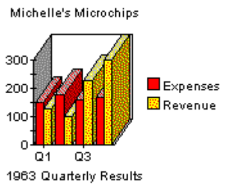

jclass.chart.View3DEditor

Depth: 31 Elevation: 45 Rotation: 45

[✓] Change Depth

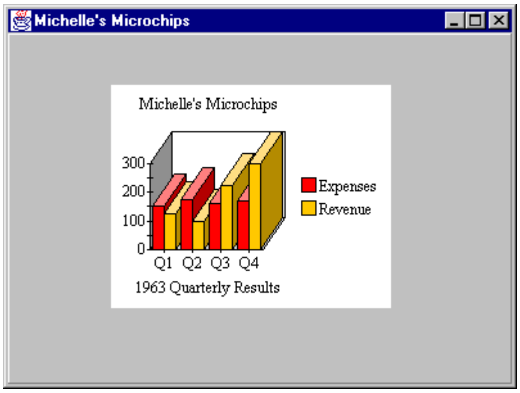
Drag mouse to set Depth or Elevation and Rotation.

Done



Step 8: Compile and Run the Application

For the last step, compile and run the application. See your IDE's documentation for details. And that's it! When you run the application, you should have a window with a chart, displaying Michelle's Microchips' financial information. The following example illustrates how the application appears when run:



4

Bean Reference

This chapter is a reference for JClass Chart Beans and their properties. For basic Bean concepts and a tutorial, see the SimpleChart Bean Tutorial.

4.1 Choosing the Right Bean

When creating new applications in an IDE, you can either use `MultiChart`, `SimpleChart`, or one of the data binding Beans. Unless you are binding to a database, we recommend using `MultiChart`, both for learning JClass Chart's features, and creating new applications.

The MultiChart Bean

`MultiChart` is JClass Chart's newest and most powerful Bean. It contains a richer set of features than previous Beans, highlighting the superiority of JClass Chart as a charting application tool. Among its features are the ability to handle multiple data sources and multiple axes. For more information, see the MultiChart chapter.

SimpleChart

`SimpleChart` was designed for quick chart development in any IDE environment. It exposes the most commonly used charting properties, and presents them in easy to use property editors. `SimpleChart` can load data from a file or a design-time editor.

`SimpleChart` and the data binding Beans share a common set of properties that are covered in this chapter. SimpleChart and the data binding Beans only differ in how they load data. Therefore, this chapter is divided into Standard Bean Properties and Data Loading Methods.

Data Binding Beans

If you want to load data from a database, you will have to use one of the data binding Beans. In order to chart data from a database, your application must be able to establish a connection, perform necessary queries on the data, and then put the data into a chartable format. This type of database connectivity is often called 'data binding'.

There are three data binding Beans. One for JBuilder, one for Visual Café, and one for JClass DataSource.

Once you have set up your data handling for a specific Bean, you can then use the Standard Bean Properties to customize your chart.

4.1.1 JClass Chart Beans

The following table shows all of the available Beans, and their uses:

JClass Chart Bean	Description
MultiChart	The newest and most powerful charting Bean. ■ Chart data from two data sources and plot them against multiple axes. ■ Data sources can be a file, or data entered at design-time. Also supports using Swing `TableModel` objects as data sources. ■ Compatible with all IDE's. See MultiChart for complete details.
SimpleChart	Chart data from a file, or data entered at design-time. Also supports a Swing `TableModel` object as a data source. Compatible with all IDE's.
DSdbChart	Bind a chart to JClass DataSource and chart data from a database. Compatible with all IDE's and the BeanBox (Requires JClass DataSource Component)
JBdbChart	Bind a chart to a JBuilder DataSet and chart data from a database. (Requires Borland JBuilder 2.0)
VCdbChart	Bind a chart with a Visual Café QueryNavigator and chart data from a database. (Requires Visual Café 2.5).
JCChartComponent	Included for backwards compatibility. It is recommended that you use the other Beans for new charting applications.

4.1.2 JClass Chart Beans and JCChart

All JClass Chart Beans are subclasses of the main chart object, `JCChart`. This means that the entire JClass Chart API is available to any developer using any of the Beans.

4.1.3 A Note for JClass Chart Lite Users

JClass Chart Lite is a special verison of Chart bundled free of charge with popular IDEs. Chart Lite limits some functionality, but imposes no run-time penalty. In other words, if you use Chart Lite and deploy an application or applet, there is no indication that Chart is "lite" – no special banner or dialog or message. Chart Lite does have some feature limitations:

- When dropped into an IDE at design time, a dialog will appear that explains you are using a Lite version. This happens for all Beans.
- Several features of MultiChart have been restricted or omitted. See the MultiChart chapter for more details on these limitations.

4.2 Standard Bean Properties

SimpleChart, and the data binding beans (VBdbChart, JBdbChart, and DSdbChart) have a set of standard properties that allow you to control the appearance and behavior of your charts.

They only differ in the way they retrieve data. This section covers the standard properties. See Data Loading Methods, later in this chapter, for information on data management properties for the different Beans.

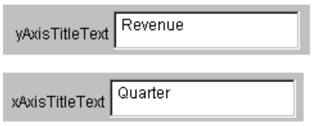
4.2.1 Axis Properties

JClass Chart Beans set up basic axis properties for you automatically, and adjust these properties to your data. You can also customize your axes with the axes property editors. You have control over the following axis properties:

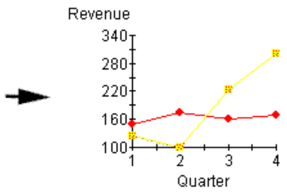
- Axis Titles
- Annotation Method
- Axis Number Intervals
- Axis Range
- Axis Grids
- Axis Hiding
- Logarithmic Notation
- Axes Orientation

Axis Titles

Enter x- and y-axis titles in the `xAxisTitleText` and `yAxisTitleText` property editors:

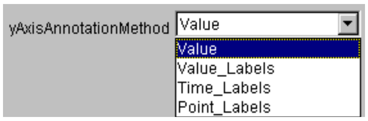


yAxisTitleText	Revenue
xAxisTitleText	Quarter

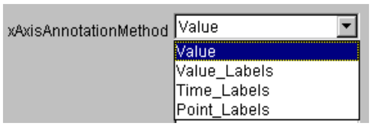


Annotation Method

Set the annotation method for the axes using the `xAnnotationMethod` and `yAnnotationMethod` editors. By default, `Value` annotation is used for both:

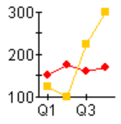


yAxisAnnotationMethod: Value

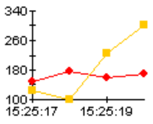


xAxisAnnotationMethod: Value

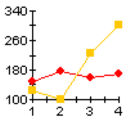
`Value_Labels` notation can only be added programmatically, or by using HTML parameters, and is therefore, not very useful for `Bean` programming. The following examples show the three applicable annotation methods as applied to the x-axis:



Point_Labels



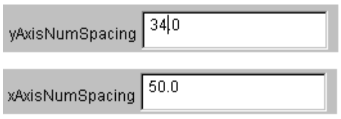
Time_Labels



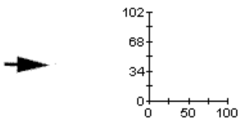
Value

Axis Number Intervals

To specify the number interval on the axes, enter the interval into the `yAxisNumSpacing` or `xAxisNumSpacing` property editors:

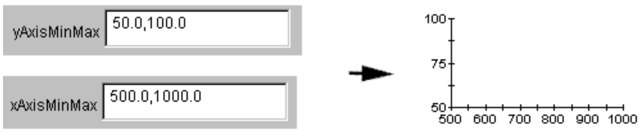


yAxisNumSpacing	34.0
xAxisNumSpacing	50.0



Axis Range

The axis number range is determined by the minimum and maximum values of the axes. By default, these values are set automatically, based on the available data. You can specify the range using the `xAxisMinMax` and `yAxisMinMax` property editors. Enter the minimum value on the left of the comma, and the maximum on the right:

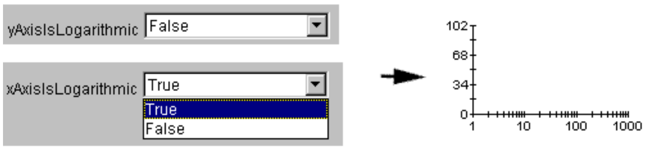


yAxisMinMax 50.0,100.0

xAxisMinMax 500.0,1000.0

Logarithmic Notation

You can specify that one or both of the axes are logarithmic, by setting the `xAxisIsLogarithmic` or `yAxisIsLogarithmic` properties to `true`:

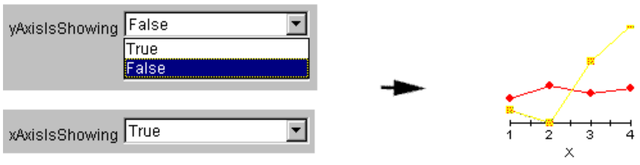


yAxisIsLogarithmic False

xAxisIsLogarithmic True

Hiding Axes

by default, both the x and y axes are displayed. You can hide them by setting the `xAxisIsShowing` or `yAxisIsShowing` properties to `false`. The following example hides the y-axis:

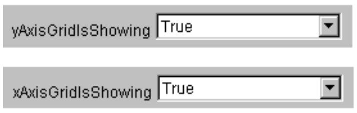


yAxisIsShowing False

xAxisIsShowing True

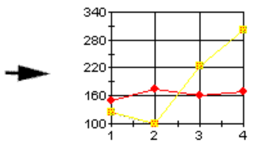
Showing Grids

Display grid lines for one or both axes by setting the `xAxisGridIsShowing` or `yAxisGridIsShowing` properties to `true`. By default, the grids are hidden. The following example sets both axes to display grid lines:



yAxisGridIsShowing True

xAxisGridIsShowing True

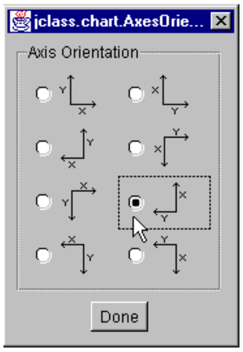


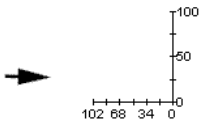
Axis Orientation

Axis orientation determines how the axes are positioned on the chart. By default, the axes are positioned with the y-axis left/vertical and the x-axis right/horizontal. Use the axis orientation custom editor to change how your axes are oriented. To launch the custom editor, click the `axisOrientation` property:

axisOrientation Click to edit...

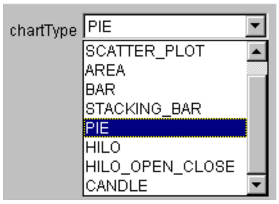
The axis orientation editor will illustrate the eight combinations. Select the desired orientation and click **Done**.



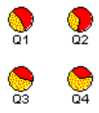


4.2.2 Chart Types

By default, JClass Chart Beans use the Plot chart type to display data. To change to another type, use the `chartType` property editor. The following example selects the PIE type:

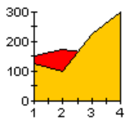


chartType PIE

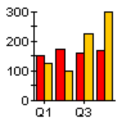


Data Interpretation

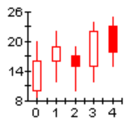
The following examples show how data is displayed by the different chart types:



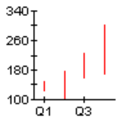
Area



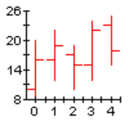
Bar



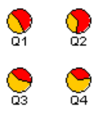
Candle



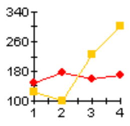
HiLo



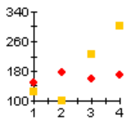
Hilo_Open_Close



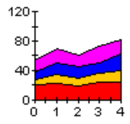
Pie



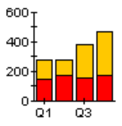
Plot



Scatter_Plot



Stacking_Area



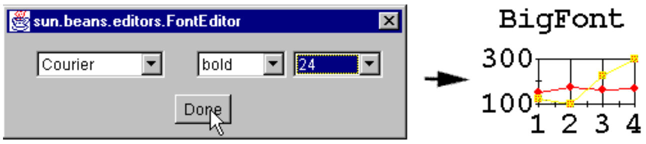
Stacking_Bar

4.2.3 Display Properties

Font

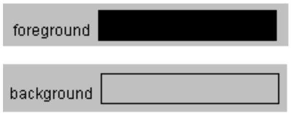
Set the size and style of text on your chart by clicking the `font` property:

The font you choose will apply to all text on the chart simultaneously. Note that the font editor that appears in your IDE may be different from the example below. The following example sets the font to **Courier, Bold, 24 point**, with the BeanBox font editor:

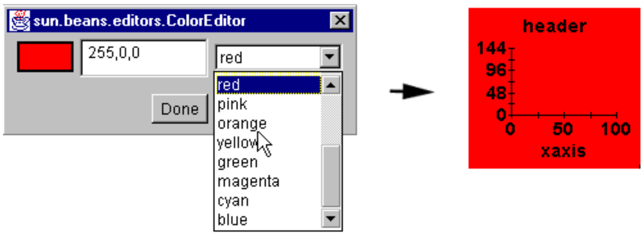


Foreground and Background Colors

Click the `foreground` and `background` properties to set the foreground and background colors of your chart. A color editor will appear. By default, the colors are black foreground and light-grey background:



Most IDE's have their own color editors that differ from the BeanBox. The following example sets the background color to `red`:



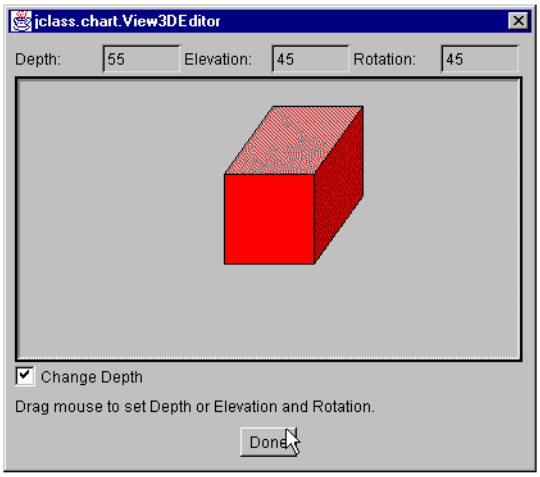
3D Effects

To add 3D effects to your chart, click the `View3D` property:

view3D Click to edit...

This will bring up the **View3DEditor**. There are two main settings in the **View3DEditor**: depth and combined elevation and rotation.

To add 3D effects, first drag the red square in the editor until it has the desired Elevation and Rotation. Then, check the **Change Depth** option box, and drag the red square until it has the Depth you want to see on your chart. The degree of depth, elevation and rotation is displayed in numbers at the top of the editor. Click **Done** to set the changes:

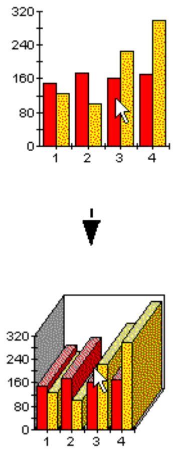


jclass.chart.View3DEditor

Depth: 55 Elevation: 45 Rotation: 45

[✓] Change Depth

Drag mouse to set Depth or Elevation and Rotation.



Margins

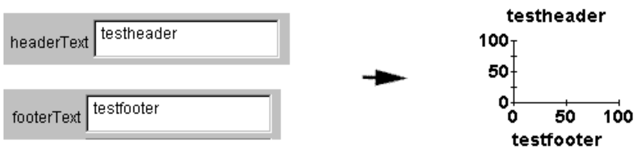
The `margins` property controls the distance from the data display to the chart area boundary:

margins 1,1,1,1

As you increase the margins, you shrink the data display area, including headers, footers, and the legend. The overall chart size remains the same.

4.2.4 Headers and Footers

Add a header, footer, or both with the `headerText` and `footerText` property editors. The following example sets both:



headerText testheader

footerText testfooter

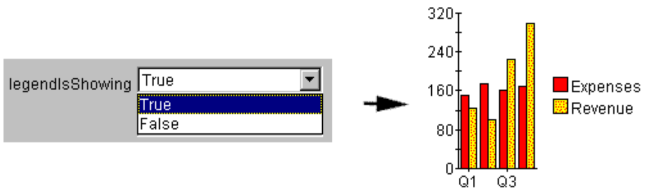
The font characteristics of the header and footer are determined by the chart's default, which can be changed with the `font` property. See Display Properties in this chapter, for more details.

4.2.5 Legends

You can add a legend, position it, and select its layout. The items that appear in the legend come from the information in the data source. In order to change what appears in the legend, you have to change what is in the data source. For information on how to set up legend items in the data source, see Standard Data Formats, in the Data Sources chapter.

Showing the Legend

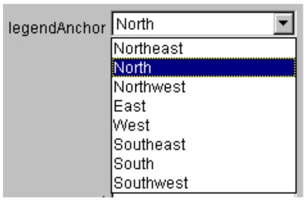
To show the legend, set the `legendIsShowing` property to `true`:

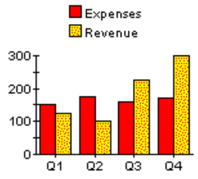


legendIsShowing True

Legend Placement

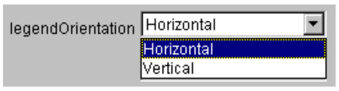
Specify where the legend will be anchored in the chart area by selecting a compass direction from the `legendAnchor` property options. By default, legends are anchored on the `East` (see above). The following example anchors the legend `North`:

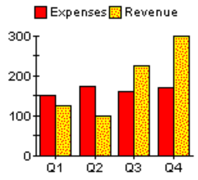




Legend Layout

Legend items can be laid out vertically or horizontally. By default the legend has a vertical layout (see above). To specify a horizontal layout, set the `legendOrientation` property to `Horizontal`:





4.3 Data Loading Methods

This section covers the data loading methods of `SimpleChart` and the data binding Beans. For `MultiChart` data loading details, see, the MultiChart chapter. Select the Bean that best matches your data needs, and follow the instructions on loading the data for that Bean:

JClass Chart Bean	Data Source & IDE Compatibility
SimpleChart	■ Formatted file or design-time editor ■ Also supports using a Swing `TableModel` object as the data source ■ All IDE's
DSdbChart	■ Data binding ■ All IDE's (Requires JClass DataSource Component)
JBdbChart	■ Data binding ■ Borland JBuilder 2.0
VCdbChart	■ Data binding ■ Visual Café 2.5

If you are using an IDE other than Borland JBuilder or Visual Café, and you want to connect to a database, then, you will have to use the JClass DataSource. JBuilder and Café users may still want to use the JClass DataSource for data binding instead of their IDE-specific solutions.

JClass DataSource

JClass DataSource is a full data binding solution. It is a robust hierarchical, multiple-platform data source that you can use to bind and query any JDBC compatible database. It can also bind to platform-specific data solutions in JBuilder and Visual Café.

The JClass DataSource product comes with the JClass Enterprise Suite, JClass HiGrid, or can be purchased separately. Visit *http://www.klg.com* for information and downloads.

4.3.1 SimpleChart: Loading Data from a File

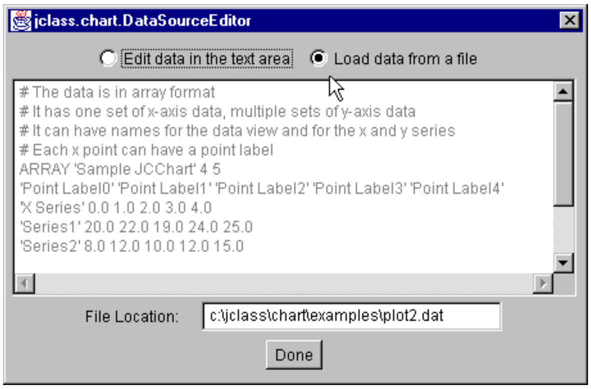
There are two ways of loading data with the `SimpleChart` Bean: from a *.dat* file, or by entering data directly into the custom editor. Both methods are managed by the **DataSourceEditor**. To bring up the **DataSourceEditor**, click on the `data` property:

data

The DataSource Editor will appear (see below).

Loading Data from a *.dat* File

To load data from a file, click **Load data from a file**, enter the name of the file in the **File Location** field, and click **Done**:


jclass.chart.DataSourceEditor

Edit data in the text area [✓ Load data from a file]

```
# The data is in array format
# It has one set of x-axis data, multiple sets of y-axis data
# It can have names for the data view and for the x and y series
# Each x point can have a point label
ARRAY 'Sample JCChart' 4 5
'Point Label0' 'Point Label1' 'Point Label2' 'Point Label3' 'Point Label4'
'X Series' 0.0 1.0 2.0 3.0 4.0
'Series1' 20.0 22.0 19.0 24.0 25.0
'Series2' 8.0 12.0 10.0 12.0 15.0
```

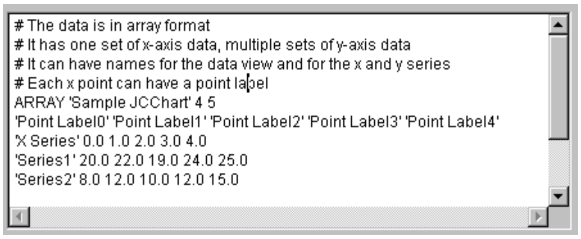
File Location: c:\jclass\chart\examples\plot2.dat

Specify the full path of the file. The file must be pre-formatted to the JClass Chart Standard (see Data Sources). Sample data files are located in the *JCLASS_HOME/jclass/chart/examples* directory.

Editing the Default Data

You can use the data provided in the editor, as is, or you can modify it. To use existing data, just check the **Edit data in the text area** radio button, and click

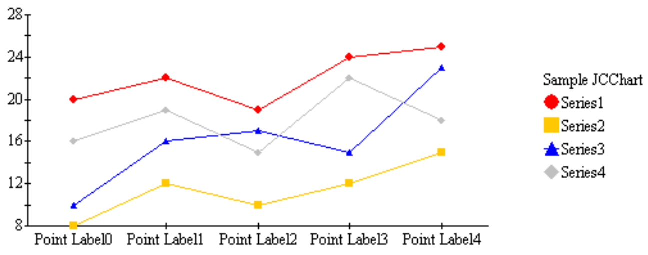
Done. Change data by deleting and inserting text in the area provided. Be careful to preserve the punctuation surrounding the original text:



```
# The data is in array format
# It has one set of x-axis data, multiple sets of y-axis data
# It can have names for the data view and for the x and y series
# Each x point can have a point label
ARRAY 'Sample JCChart' 4 5
'Point Label0' 'Point Label1' 'Point Label2' 'Point Label3' 'Point Label4'
'X Series' 0.0 1.0 2.0 3.0 4.0
'Series1' 20.0 22.0 19.0 24.0 25.0
'Series2' 8.0 12.0 10.0 12.0 15.0
```

The chart below shows how the default data appears as a plot. Notice where the different elements are positioned. Each point on the x-axis is labelled with the names specified in the default data. The name of each series of y-values appears in the legend. The name of the data view is positioned directly above the legend.

In order for the default data to display this way, you must first set the `xAxisAnnotation` property to `Point_Labels`, and the `legendIsShowing` property to `true`.



4.3.2 SimpleChart: Using Swing TableModel Data Objects

Your (Swing) application may have the data you want to chart contained in a Swing `TableModel`-type data object. You can use this object as your data source instead of using the JClass Chart built-in data sources.

Use the `SwingDataModel` property to specify an already-created Swing `TableModel` object to use as the chart's data source.

4.3.3 Data Binding in Borland JBuilder

Binding a chart to a database in JBuilder involves adding a database connection and query functionality with JBuilder Components and then using `JBdbChart` to connect to the dataset and chart the data. This section walks through these steps.

Database connection and querying are handled by JBuilder components. Our coverage of these components is only intended as a guide. For detailed information on JBuilder database connectivity, consult your JBuilder documentation.

Before proceeding, make sure you have:

- Borland JBuilder 2.0
- JDK 1.1
- `JBdbChart` Bean loaded in your JBuilder Palette. For details on how to load a Bean, see the Getting Started Chapter or your JBuilder documentation.
- Database set up properly.
- Basic SQL command knowledge

Step 1: Connect to a Database

Use JBuilder's `Database` Bean to add a database connection. The icon is found under the **Data Express** tab.

Add an instance to your frame. Then, use the `connection` property to specify the URL of the database that you want to use.

Step 2: Query the Data

To query the database, add an instance of JBuilder's `QueryDataSet` to your frame. This Bean is found under the **Data Express** tab.

Select columns that you may want to chart with the `query` property editor. Each column will represent a series of data, or point labels. For example, to select all of the columns from a table named MotorVehicle_Sales, you would type a statement similar to:

```
select * from MotorVehicle_Sales
```

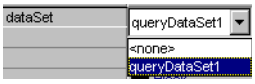
You can include all columns at this step, and then use JBdbChart to choose which ones to display later.

Step 3: Connect the Chart to the DataSet

With the database connection established and the query created, you can now use `JBdbChart` to connect to the JBuilder DataSet and chart the data. `JBdbChart`'s data binding properties are `dataSet`, and `DataBindingMetaData`.

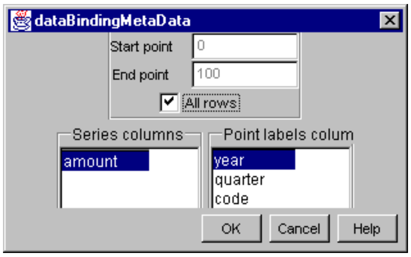
Insert a `JBdbChart` into your frame.

Select a query from the `dataSet` property's pull down menu. If the database connection and query are set up properly with JBuilder components, there should be one or more queries in the list.



You can now select the columns and range of data that will be displayed. Columns that contain numeric data are considered 'data series', and can be plotted on a chart. Columns that have non-numeric data can be used for point labels on the X-axis. Click the `dataBindingMetaData` property to bring up the custom editor:

This editor allows you to set the columns and the data range of the chart. Click on column names to select them (when they are highlighted, they are selected).



dataBindingMetaData

Start point 0

End point 100

[✓] All rows

Series columns: amount

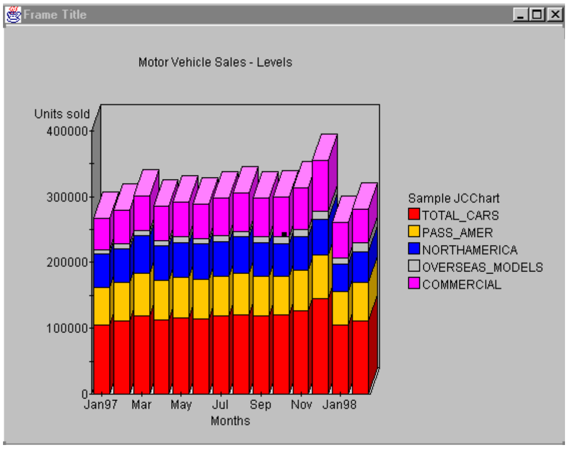
Point labels colum: year, quarter, code

Columns on the left hand side of the editor are numeric. Columns on the right are non-numeric. Only one X-axis column is allowed.

You can either set the range to all data by checking the **All rows** box, or you can specify a range using the **Start Point** and **End Point** fields.

In order to display the point labels on the X-axis, you have to set the `xAxisAnnotationMethod` property to `Point_Labels`. For more information, see, Axis Properties.

That's all there is to it. You should see your data in the design frame:.



With your connection established, you can then use the Standard Bean Properties to customize and enhance your chart. In the example above, a header, footer, axis title, legend, point labels and 3D effects have been added.

4.3.4 Data Binding in Visual Café

Binding a chart to a database in Café involves two basic steps: establishing a connection to a database table with Café components, and then using `VCdbChart` to connect the chart to the QueryNavigator and display the data.This section walks through the steps.

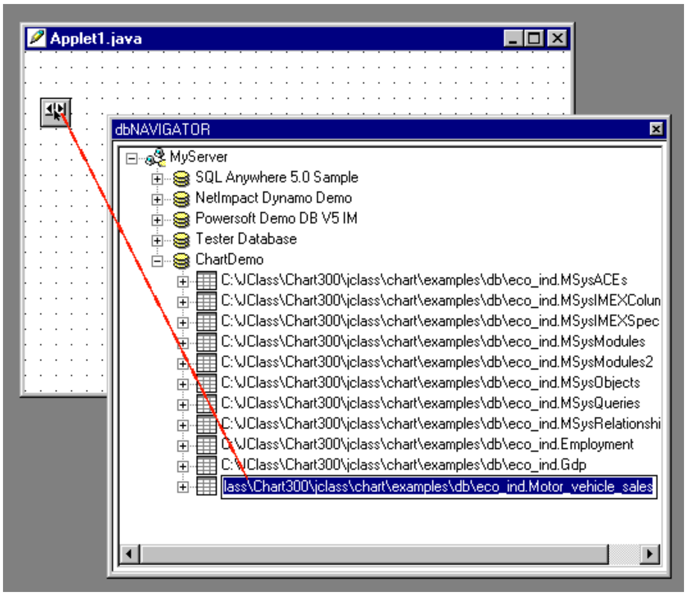
Database connection and querying are handled by Visual Café components. This manual's coverage of these components is only intended as an introductory guide. For detailed information on database connectivity, consult your Visual Café documentation.

Before proceeding, you must have:

- Visual Café 2.5
- JDK 1.1
- `VCdbChart` component in your Visual Café component library. For details on how to load a Bean, see the Getting Started Chapter or your Café documentation.
- Symantec dbAnywhere server running

Step 1: Connect to a DataBase Table

An easy way to set up data binding in Visual Cafe is to drag a table from the **dbNavigator** window to the design area and 'drop' it in. This will set up the `DatabaseManager`, `JdbcConnection`, and `QueryNavigator` all at once:


Applet1.java
dbNAVIGATOR
MyServer
SQL Anywhere 5.0 Sample
NetImpact Dynamo Demo
Powersoft Demo DB V5 IM
Tester Database
ChartDemo
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.MSysACEs
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.MSysIMEXColun
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.MSysIMEXSpec
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.MSysModules
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.MSysModules2
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.MSysObjects
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.MSysQueries
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.MSysRelationshi
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.Employment
C:\JClass\Chart300\jclass\chart\examples\db\eco_ind.Gdp
lass\Chart300\jclass\chart\examples\db\eco_ind.Motor_vehicle_sales

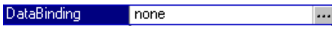
When you see the `QueryNavigator` icon in your design area, you can proceed to the next step: connecting the chart to the database.

Step 2: Connect the Chart to QueryNavigator

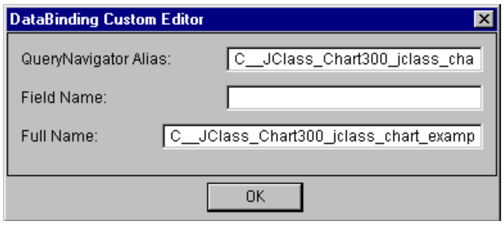
Once you have a connection to the database in Café, connect `VCdbChart` to the `QueryNavigator` and select the columns to be displayed. Two properties are used for connecting the data: `DataBinding`, and `DataBindingMetaData`.

First, add a `VCdbChart` to the design area. The icon is shown below

Click the `DataBinding` property to bring up the `Databinding` custom editor.



DataBinding none



DataBinding Custom Editor

QueryNavigator Alias: C__JClass_Chart300_jclass_cha

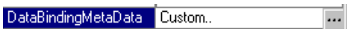
Field Name:

Full Name: C__JClass_Chart300_jclass_chart_examp

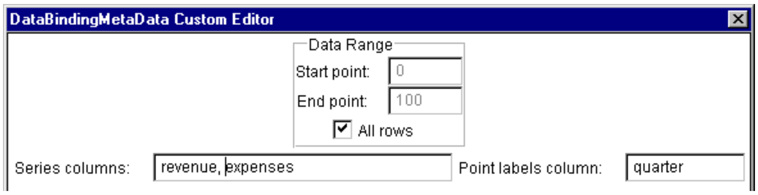
OK

In the **QueryNavigator Alias** field, enter the *Alias Name* of your `QueryNavigator`. The easiest way to do this is cut/paste from the `QueryNavigator's` property list. The **Full Name** field will automatically update itself when selected. Click **OK** when finished.

You can now select the columns and range of data that you want to display, using the `DataBindingMetaData` custom editor. To bring it up, click `DataBindingMetaData` property:



DataBindingMetaData Custom..



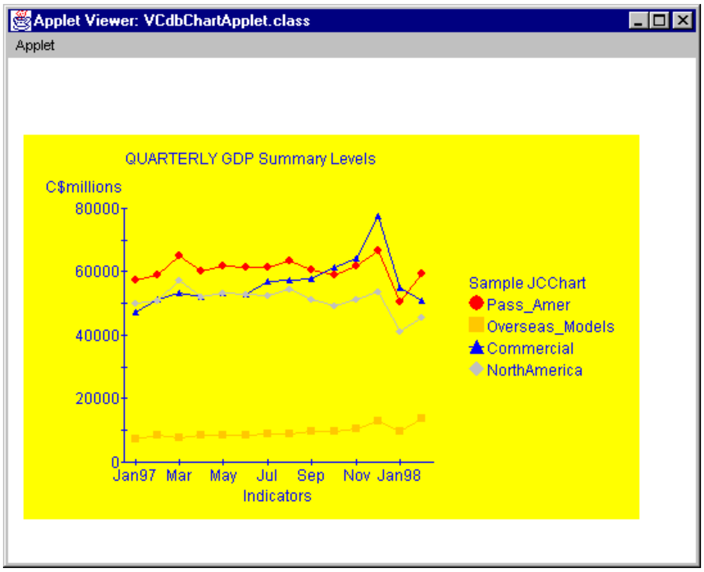
Start point: 0
End point: 100
☑ All rows
Series columns: revenue, expenses
Point labels column: quarter

In the **Series Columns** field, enter the columns to be displayed as Y- data series. Separate the column names with a comma.

Then enter the column that you have selected for point labels (optional) in the **Point labels column** field. In order to display the point labels on the X-axis, set the `xAxisAnnotationMethod` property to `Point_Labels` (for more information, see, Axis Properties).

You can either set the range to all data by checking the **All rows** box, or you can specify a range using the **Start Point** and **End Point** fields.

Now, test-run the application to see how the data is displayed in your chart, and then use the Standard Bean Properties to control the behavior and appearance of the chart. In the example below, a header, footer, axis title, legend, and point labels have been added.



4.3.5 Data Binding with JClass DataSource

The JClass DataSource manages all connection and query functionality for data binding. After establishing a connection and query with JClass DataSource, you then bind `DSdbChart` to JClass DataSource to chart the data.

The JClass DataSource package contains a number of Beans used for binding to databases, including `TreeDataBean`, and `DataBean`. This section will illustrate the process with the `DataBean` Bean. DSdbChart uses the same method to connect to either Bean. Consult your JClass DataSource documentation for details on their features and how to use them.

To use this solution, you require the following:

- Sun's BeanBox, or any IDE
- JDK 1.1
- JClass DataSource. (comes with JClass Enterprise, HiGrid, or can be purchased separately).
- `DSdbChart` loaded into the BeanBox or IDE. For details on how to load a Bean, see the Getting Started chapter, or your JClass DataSource documentation
- If you are using Windows, you will need to establish an ODBC database connection. Set this in the in **Control Panel > ODBC.**

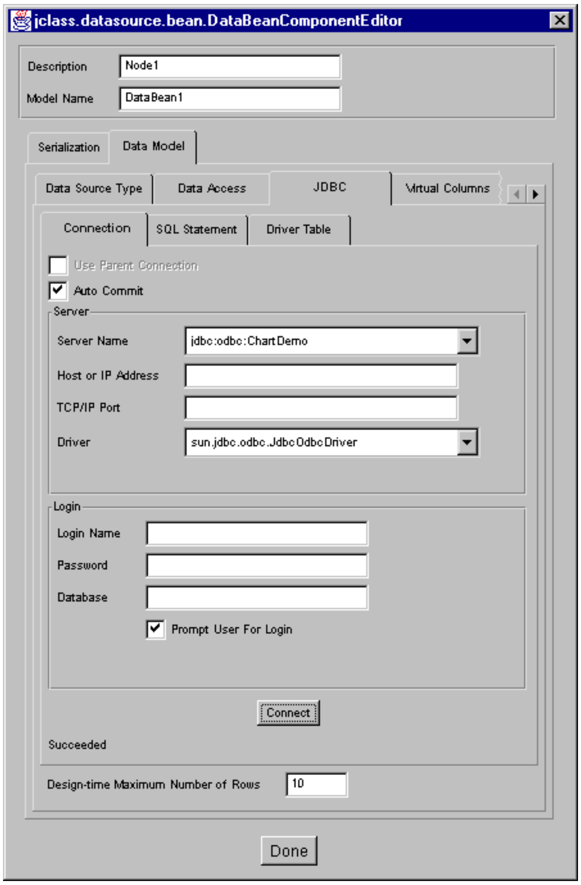
The following steps guide you through using `DSdbChart` to connect to JClass DataSource. They are: connect to a database, query the data, and connect `DSdbChart` to the JClass DataSource.

Step 1: Connect to a Database

Add a `DataBean` instance to your design area. The icon looks like this

Click the `dataBeanComponent` property to bring up the DataBeanComponentEditor.

dataBeanComponent Click to edit...



Description: Node1
Model Name: DataBean1
Use Parent Connection: ☐
Auto Commit: ✓
Server Name: jdbc:odbc:ChartDemo
Host or IP Address:
TCP/IP Port:
Driver: sun.jdbc.odbc.JdbcOdbcDriver
Login Name:
Password:
Database:
Prompt User For Login: ✓
Succeeded
Design-time Maximum Number of Rows: 10

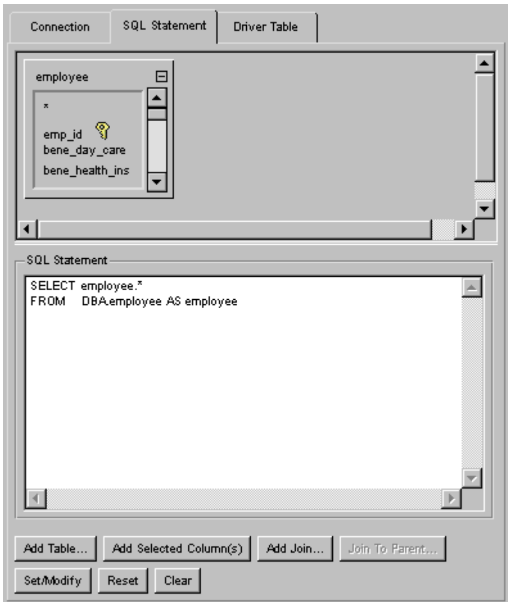
This editor manages all of the connection and query settings. The first thing you have to do is set up a serialization file under the **Serialization** tab. This file saves information and settings about the connection. You can then proceed to set up a connection and query.

To set up a database connection, go to the **DataModel > JDBC > Connection** tab, and specify the *Server Name* and *Driver* for the database you want to connect to. Test the connection. If there are error messages, consult your JClass DataSource documentation.

When your connection is successful, you can then proceed to set up a query.

Step 2: Query the Data

Click the **Data Model > JDBC > SQL Statement** tab to show the query options:



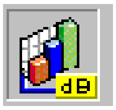
```
SELECT employee.*
FROM   DBA.employee AS employee
```

You can create your whole SQL query using mouse clicks. First, add a table, and then create a query by selecting columns. When all finished, click **Set/Modify**, and then **Done**.

Step 3: Connect a Chart to JClass DataSource

With your database connection established, you can then bind a chart to the data. This is done using the `dataBinding` and `DataBindingMetaData` property editors.

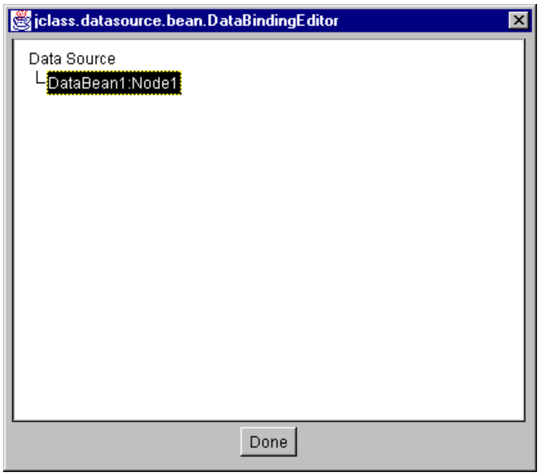
First, add `DSdbChart` to your design area. The icon looks like this:



Click the `dataBinding` property to bring up the DataBindingEditor.

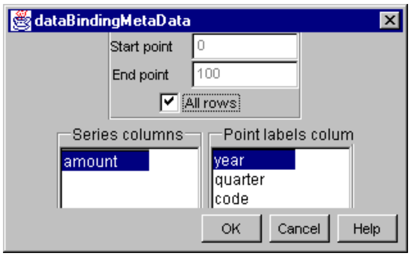
dataBinding DataBean1:Node1

If the connection in JClass DataSource is properly established, you should see one or more data sources to select from:



Select a source and click **Done**.

You can now select the columns and range of rows to be displayed in the chart. To do this, click the `DataBindingMetaData` property to bring up the DataBindingMetaData custom editor:



Columns on the left are numeric and are displayed as Y series. Columns on the right can be used as X-axis point labels. In order to display the point labels on the X-axis, set the `xAxisAnnotationMethod` property to `Point_Labels`. For more information, see, Axis Properties.

You can either set the range to all data by checking the **All rows** box, or you can specify a range using the **Start Point** and **End Point** fields.

When you click **Done**, you should see the data displayed in the design area of the Beanbox or IDE. Your data binding is complete.

5

MultiChart

5.1 Introduction to MultiChart

`MultiChart` is the next generation charting Bean from JClass Chart. It contains a richer set of features than previous Beans, highlighting the superiority of JClass Chart as a charting application tool.

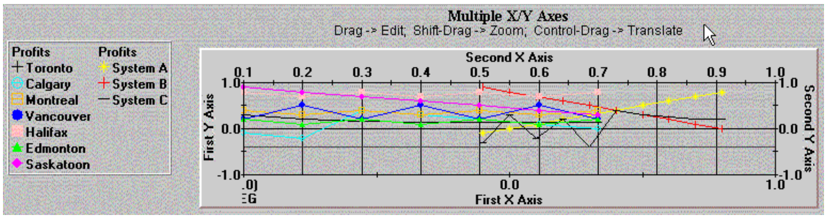
The `MultiChart` icon:

Highlights of the new MultiChart Bean

- Handles multiple data sources
- Plot data against multiple x and y axes.
- Fully customizable axes.
- Extensive control of font, colors, borders, and styles for each chart element.

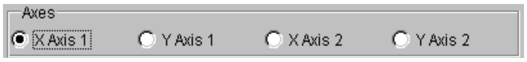
5.1.1 Multiple Axes

`MultiChart` can have two x and two y axes, as in the example below:



Setting Properties on Multiple Axes

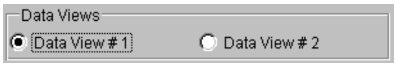
Axis properties can be set for each axis individually. At the top of each axis editor you will see four radio buttons:



When a radio button is selected, all that follows below will apply to that axis.

5.1.2 Multiple Data Views

`MultiChart` allows you to load data from two different sources at the same time.When loading data from two different sources, they are each assigned to a separate data view.



By default, both data views are showing, but you can hide or reveal data views depending on your application's needs. Both sets of data can be mapped to the same set of x and y axes, or, mapped to different axes.

5.1.3 Intelligent Defaults

`MultiChart` has a sophisticated set of dynamic default settings in the custom property editors. You can override these defaults to suit your needs. When you override a default value in a text editor, it becomes static, and will not automatically adjust anymore.

Returning to Default Values

If you want to return to default settings in the custom editors after overriding them, all you have to do is delete the contents of the changed field, and leave it blank. The next time you bring the editor you will see that the automatic values have returned.

5.2 Getting Started with MultiChart

`MultiChart` has a sophisticated set of dynamic default settings that adjust to your data and other settings. This means that you only have to make a minimum of settings to have a respectable chart. The following list describes the most common start-up tasks and the editors used for them:

- **Load Data.** To load data in the chart, use the DataSource editor. This editor allows you to load data from one or two sources. There is also a default set of data built-in that you can use to experiment with. Alternately, you can use a Swing `TableModel` data object as the chart's data source using the `SwingDataModel` property.
- **Select Chart Types.** For each data view, you can select a chart type and the axes that the data will be plotted against with the DataChart editor.
- **Set BackGround Color.** Use ChartAppearance to set the color of the chart background.
- **Set Axis Annotation.** By default, `MultiChart` uses values to annotate the axes. You can also use value labels, point labels, or time labels by setting the annotation type with the AxisAnnotation editor.
- **Add a Legend.** Add a legend by checking the Showing box in the LegendAppearance editor.
- **Add a Header and Footer.** To add a header, use HeaderText to add the text, and then check the Showing box in HeaderAppearance. The footer is the same, but uses the FooterText, and FooterAppearance editors
- **Add Extra Axes.** By default a standard X-Y axis set is displayed. If you require, you can display a second X or Y axis. Display them with the AxisMisc editor's **IsShowing** property. Then use the many axis editors, such as AxisPlacement, to set up and align the axes.

5.3 MultiChart Property Reference

The following property reference covers all of `MultiChart`'s features. Note that if you are a JClass Chart Lite user, not all features will be available for you. The following list summarizes the limited features:

Features Unavailable to JClass Chart Lite Users

When dropped into an IDE at design time, a dialog will appear that explains that you are using a lite version. The limited components will not have any run-time penalty, such as a banner or message. The following features are limited at design-time, however:

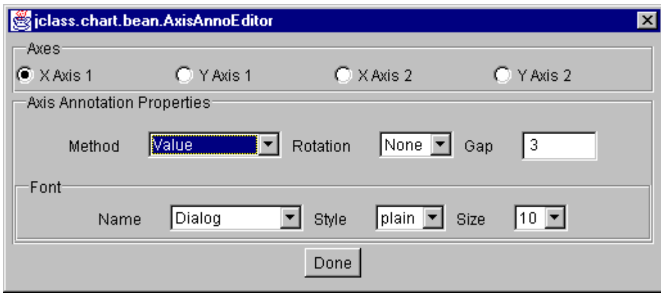
Feature/Editor	Limiting
All appearance editors	Border types limited to NONE, PLAIN, IN, FRAME_IN and ETCHED_IN
All axis editors	Multiple X-Y axes not available
AxisAnnotations	Time labelling disallowed
AxisPlacement	Automatic axis placement only.
AxisRelationships	All features disabled
AxisScale	Automatic tick-spacing and precision only
DataChart	Plotting data against second x or y axis disallowed
OriginPlacement	Only automatic placement allowed
TriggerList	Editor does not allow triggers to use control, alt, or meta modifiers.

5.3.1 Axis Controls

This group of editors sets up the axes. `MultiChart` has a sophisticated set of automatic default values, that adjust to your data. This makes chart development fast and easy. But, `MultiChart` is also extremely flexible, and every aspect of the axes can be adjusted.

AxisAnnotation

With the `AxisAnnotation` editor, you can set the annotation type for each axis, and control how they look. Axis annotations are numbers or text that appear along the axes. Options in the **Method** menu are: `Value`, `Time_Labels`, `Point_Labels`, and `Value_Labels`.



jclass.chart.bean.AxisAnnoEditor

Axes: [✓ X Axis 1] Y Axis 1 X Axis 2 Y Axis 2

Axis Annotation Properties

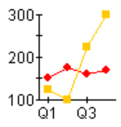
Method: Value; Rotation: None; Gap: 3

Font

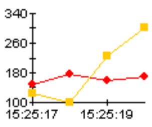
Name: Dialog; Style: plain; Size: 10

For each of the labelling methods, there is a corresponding editor that gives you more control over the behavior and appearance. For Value, use AxisScale, for `Point_Labels`, use AxisPointLabels, for `Time_Labels`, use, AxisTimeLabels, and for `Value_Labels`, use, AxisValueLabels.

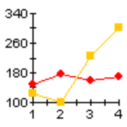
The following examples illustrate the different label types:



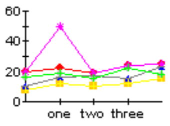
Point_Labels



Time_Labels

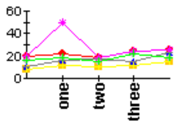


Value



Value_Labels

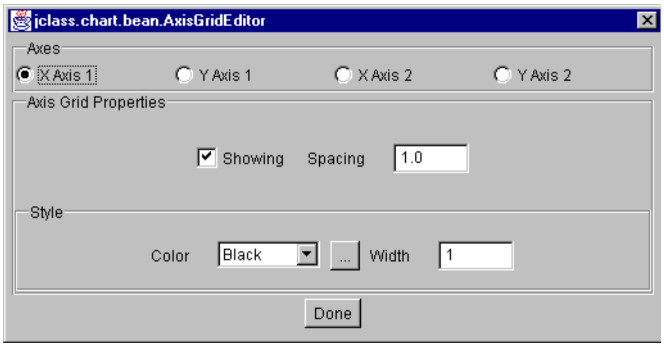
With the **Rotation** property, you can rotate the labels on the axis. The following example shows Value_Labels, rotated by 90 degrees and with bold, 12pt font:



Gap controls the space between annotations. If, for example, you used point labels, you could use the **Gap** property to make sure they have enough room to display properly.

AxisGrid

Use the `AxisGrid` editor to set up grid lines on each of the axes. There are also controls for color, line spacing, and line width of the grid lines.


jclass.chart.bean.AxisGridEditor

Axes

[● X Axis 1] ○ Y Axis 1 ○ X Axis 2 ○ Y Axis 2

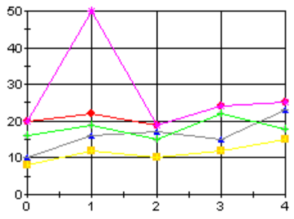
Axis Grid Properties

☑ Showing Spacing 1.0

Style

Color Black Width 1

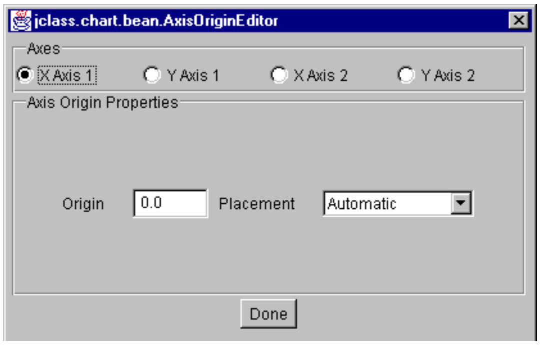
The following example sets X Axis 1 grid and Y Axis 1 grid to **Showing**, with **Spacing** = 1, and **Width** = 1:



AxisOrigin

The `AxisOrigin` editor allows you to specify an origin by coordinates, or by choosing an option from a pull down menu. By default axes origins are set automatically, based on the available data.

To place the origin, you can select one of the locations from the pull-down menu, such as Min, or Max. If you want to set the origin to a specific value on the axis, select `Value_Anchored` from the menu and then enter the value in the **Origin** field.:


jclass.chart.bean.AxisOriginEditor

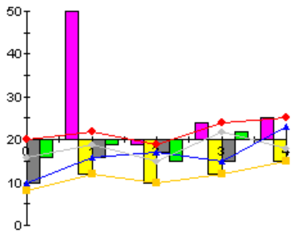
Axes

● X Axis 1 ○ Y Axis 1 ○ X Axis 2 ○ Y Axis 2

Axis Origin Properties

Origin 0.0 Placement Automatic

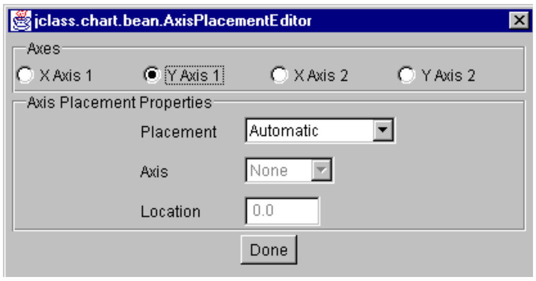
The following example anchors the origin of Y Axis 1 at 20 (default data):



Note that, by default, X Axis 1 is placed at the origin of Y Axis 1. To override this default, use the AxisPlacement editor.

AxisPlacement

Axis placement determines the placement of an axis in relation to another. By default, this is set automatically by `MultiChart`, based on the given data. Sometimes, however, you may want to locate an axis in a different location.



jclass.chart.bean.AxisPlacementEditor

Axes: ○ X Axis 1 [● Y Axis 1] ○ X Axis 2 ○ Y Axis 2

Axis Placement Properties

Placement: Automatic

Axis: None

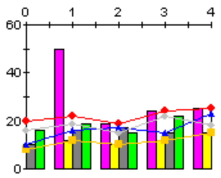
Location: 0.0

Using the **Placement** field, select the type of placement for the axis selected. Placement options include: `Min, Max, Automatic. Origin,` and `Value_Anchored`.

The **Axis** field selects the anchor-axis that you want to place the current axis against (e.g. place X Axis 1 in relation to Y Axis 2). If you select `None` as an Axis, `MultiChart` will use the default axis.

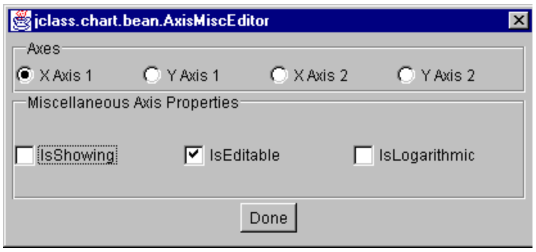
To place the axis at a specific value along another axis, select `Value_Anchored` from the pull-down menu, and enter the value in the **Location** field.

The following example shows X Axis 1, with a **Placement** of `Max` in relation to Y Axis 1:



AxisMisc

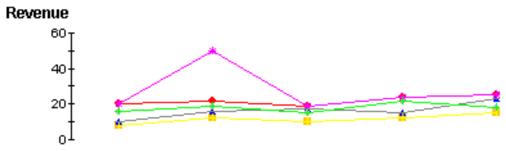
Use `AxisMisc` to show or hide any of the axes. It also allows you to make any axis logarithmic. The `IsEditable` property, when selected allows zooming, editing, and translation for the selected axis. For more information on interactive events, see Event Controls



Axes: [✓ X Axis 1] Y Axis 1 X Axis 2 Y Axis 2

Miscellaneous Axis Properties: IsShowing [✓ IsEditable] IsLogarithmic

The following example hides X Axis 1 from view by deselecting `IsShowing`:

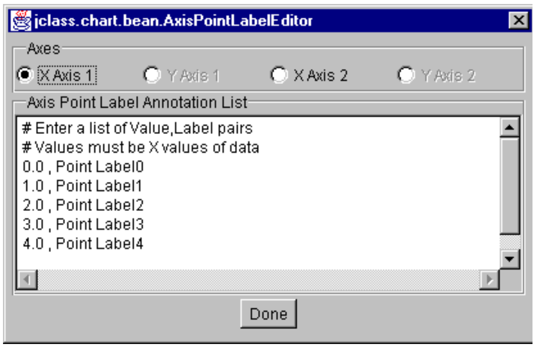


AxisPointLabels

Use the `AxisPointLabels` editor to create point labels (applies to X1 and X2 axes only). Point labels label specific points of data on the X axes.

The editor reads data from the data source associated with the selected axis and provides a list of point labels that are initially blank. To add text to these labels, enter the text alongside the point, with a comma (see editor below for an example).

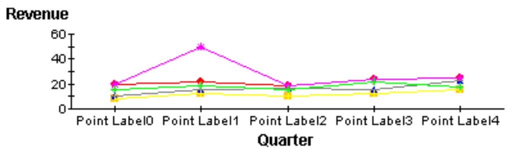
In order for the labels to appear on the chart, you also have to set the annotation method to **Point_Labels** in the `AxisAnnoation` editor.


jclass.chart.bean.AxisPointLabelEditor
Axes
X Axis 1
Y Axis 1
X Axis 2
Y Axis 2
Axis Point Label Annotation List
Done

Axes: [✓ X Axis 1] Y Axis 1 X Axis 2 Y Axis 2

```
# Enter a list of Value,Label pairs
# Values must be X values of data
0.0 , Point Label0
1.0 , Point Label1
2.0 , Point Label2
3.0 , Point Label3
4.0 , Point Label4
```

The following example shows how the default data's point labels appear on X Axis 1:



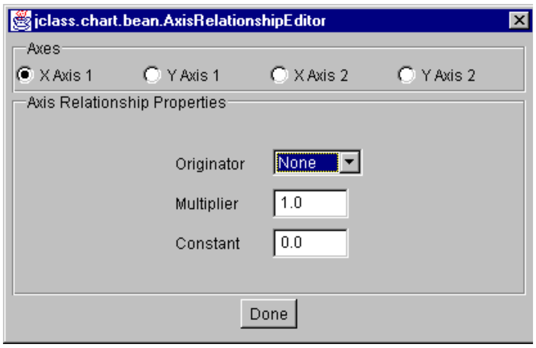
Note that if you are mapping multiple data sources against a single axis, then you will want to use value labels instead, as the `AxisPointLabels` editor only uses points from the first data source associated with the selected axis.

AxisRelationships

The `AxisRelationships` editor allows you to create a mathematical relationship between two axes. For example, if you want to create a thermometer chart with Celsius values on the left and the Fahrenheit values on the right, you could create a Celsius axis, and then base the Fahrenheit axis values on it.

There are three properties included in this calculation: **Originator**, **Multiplier**, and **Constant**. The calculation is based on the formula:

New Axis Value = Constant + Multiplier x Originator.


jclass.chart.bean.AxisRelationshipEditor

Axes

[✓ X Axis 1] Y Axis 1 X Axis 2 Y Axis 2

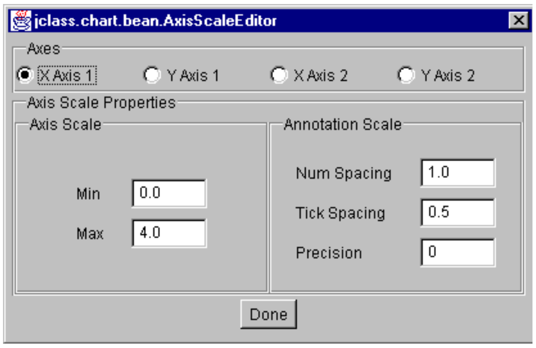
Axis Relationship Properties

Originator	None
Multiplier	1.0
Constant	0.0

To use this editor, first select an Axis that you want to affect (with a radio button). Then, select an axis from the **Originator** menu that your calculation will be based on; and enter a value in the **Multiplier** field that represents the relationship. The **Constant** value is optional.

AxisScale

The `AxisScale` editor controls the range on each axis, the interval of the numbering, and **Tick Spacing**. It is used primarily for the Value method of axis annotation (See the AxisAnnotation editor). **Precision** determines the numeric precision of the axis numbering. The Min and Max fields determine the range of data that is displayed on the chart. There are intelligent defaults in this editor that adjust to your data and other chart settings. You can override these settings with the fields provided.



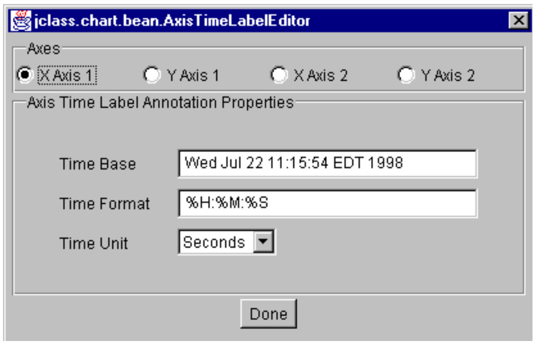
Axes: [✓ X Axis 1] Y Axis 1 X Axis 2 Y Axis 2

Min	0.0
Max	4.0
Num Spacing	1.0
Tick Spacing	0.5
Precision	0

AxisTimeLabels

The AxisTimeLabels editor allows you to control how the time labels appear. When you select the annotation method with `AxisAnnotations`, you can select time labels, which represent the values on the axis as units of time.

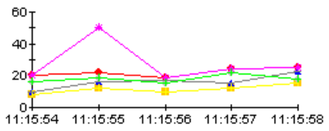
Time Base determines the date and time that the labelling starts from (default is current time/date). **Time Unit** is the unit of time the labels use, such as year, month, day, minute, second, etc.... The default time unit is minutes. **Time Format** field allows you to customize the text in the time labels with a set of formatting codes. See Axis Labelling and Annotation Methods in the Axis Controls chapter for a list of these codes.


jclass.chart.bean.AxisTimeLabelEditor
Axes
X Axis 1
Y Axis 1
X Axis 2
Y Axis 2
Axis Time Label Annotation Properties
Done

Axes: [✓ X Axis 1] Y Axis 1 X Axis 2 Y Axis 2

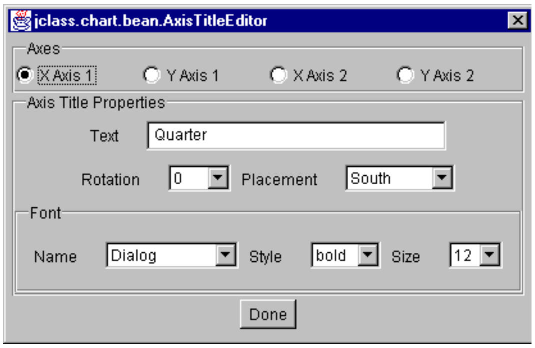
Time Base	Wed Jul 22 11:15:54 EDT 1998
Time Format	%H:%M:%S
Time Unit	Seconds

The following example uses time labelling on X Axis 1, with seconds as the time unit:



AxisTitle

Using the `AxisTitle` editor, you can add axis titles to each axis. There are also settings for the font, point, rotation and placement of the title


jclass.chart.bean.AxisTitleEditor

Axes

[✓ X Axis 1] Y Axis 1 X Axis 2 Y Axis 2

Axis Title Properties

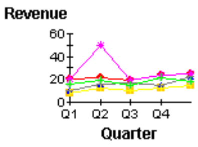
Text: Quarter

Rotation: 0 Placement: South

Font

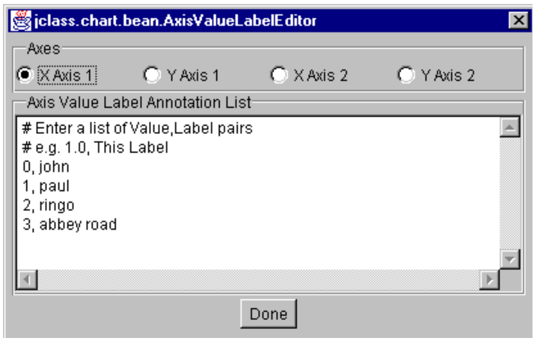
Name: Dialog Style: bold Size: 12

In the **Placement** field's pull down menu are a list of compass directions for title placement. Not all options are available to x and y axes. If you select a placement, and it returns to the previous selection, that placement is not available for that axis. The following example adds titles to X Axis 1 and Y Axis 1, and sets the font to bold, with a size of 12:



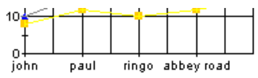
AxisValueLabels

Use the `AxisValueLabels` editor to enter value labels for the axes. Value labels appear on along the axis at specified values. You also have to set the annotation method to **Value_Labels**, in the `AxisAnnotation` editor before the labels will display.


jclass.chart.bean.AxisValueLabelEditor
Axes
X Axis 1
Y Axis 1
X Axis 2
Y Axis 2
Axis Value Label Annotation List
Done

```
# Enter a list of Value,Label pairs
# e.g. 1.0, This Label
0, john
1, paul
2, ringo
3, abbey road
```

To add value labels, enter the value, followed by a comma and a label (see above). The following example shows how the labels in the editor above appear on X Axis 1.

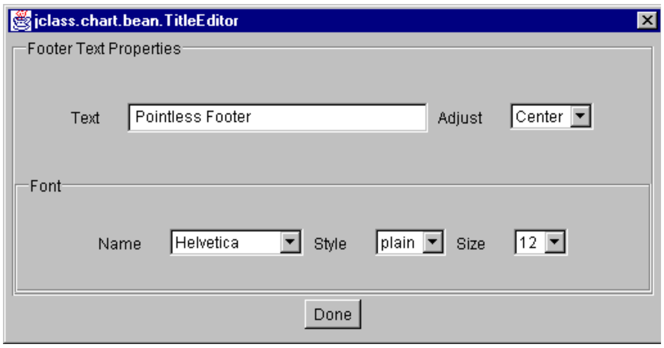


5.3.2 Headers, Footers, and Legends

FooterText

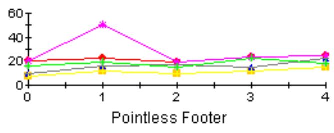
The `FooterText` editor allows you to enter text that will appear at the bottom of the chart area. You can also select a font, font style and size of the footer.

Note that the footer will not display unless you check the **Showing** box, in the FooterAppearance editor (this editor also controls footer borders, background and foreground).



Text: Pointless Footer
Adjust: Center
Name: Helvetica
Style: plain
Size: 12

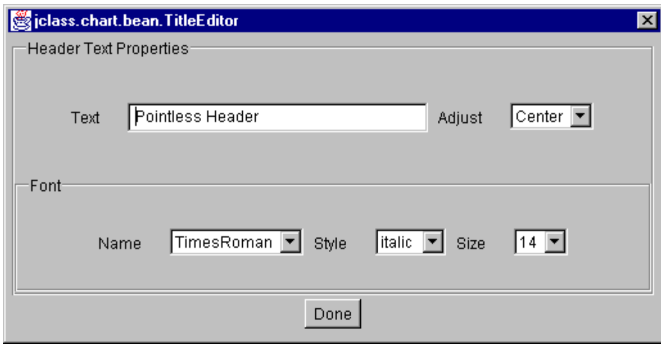
The following example shows how a ‘pointless footer’ appears on the chart area:



HeaderText

The `HeaderText` editor allows you to enter header text, that will appear at the top of the chart area. You can also select a font, font style and size of the header.

Note that the header will not display unless you check the **Showing** box, in the HeaderAppearance editor (which also controls header borders, background and foreground).


jclass.chart.bean.TitleEditor

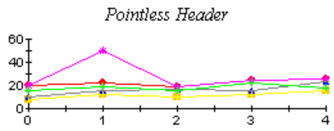
Header Text Properties

Text: Pointless Header
Adjust: Center

Font

Name: TimesRoman
Style: italic
Size: 14

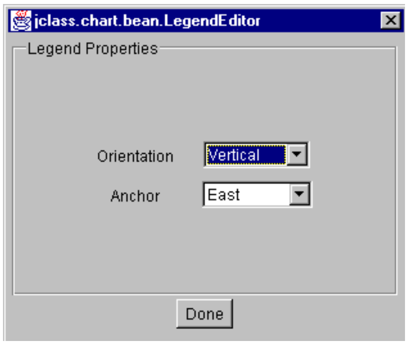
The following example shows how a ‘pointless header’ displays on the chart:



LegendLayout

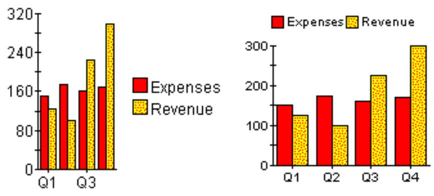
The `LegendLayout` editor controls the layout of the legends. **Orientation** determines how the legend items are placed in the legend (either vertically or horizontally). The **Anchor** property positions the entire legend on the chart, based on compass directions.

In order for the legend to display on your chart, the **Showing** checkbox in the LegendAppearance editor must be selected.



Orientation: Vertical
Anchor: East

Below are two examples of legend layout:



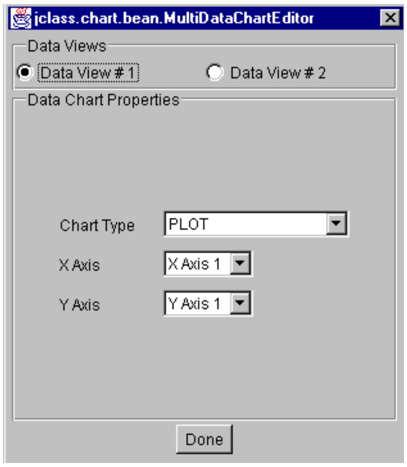
The example on the left uses the default settings with **Anchor** = `East` and **Orientation** = `Vertical`. In the example on the right, **Anchor** = `North`, and **Orientation** = `Horizontal`.

5.3.3 Data Source and Data View Controls

This group of editors manages the properties that control the data source, and the views on the data. `MultiChart` can load data from two different sources. Each of the data sources is assigned to a data view.

DataChart

The `DataChart` editor allows you to select the chart type of each data view, and which axes each data view will be mapped against.



jclass.chart.bean.MultiDataChartEditor

Data Views

[✓ Data View # 1] Data View # 2

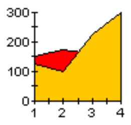
Data Chart Properties

Chart Type: PLOT

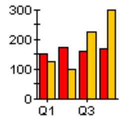
X Axis: X Axis 1

Y Axis: Y Axis 1

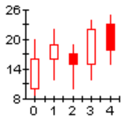
The ChartType property selects from the following chart types:



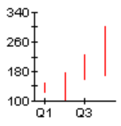
Area



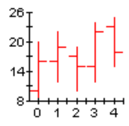
Bar



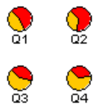
Candle



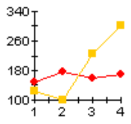
HiLo



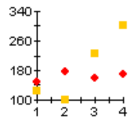
Hilo_Open_Close



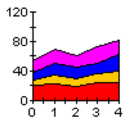
Pie



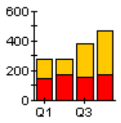
Plot



Scatter_Plot



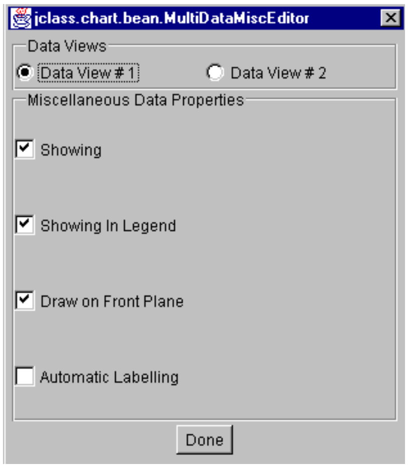
Stacking_Area



Stacking_Bar

DataMisc

The `DataMisc` editor controls several aspects of the data views.



Data Views

(● Data View # 1) ○ Data View # 2

Miscellaneous Data Properties

☑ Showing

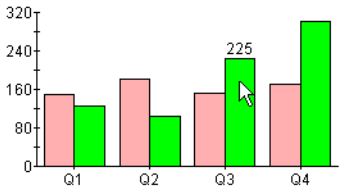
☑ Showing In Legend

☑ Draw on Front Plane

☐ Automatic Labelling

With the **Showing** property, you can show or hide each data view from the display area. **Showing In Legend** will show/hide a data view from the legend (but the data will still be charted).

Automatic Labelling attaches a dwell label to every data point in the chart. A dwell label is an interactive label that shows the value of a point, bar or slice, when a user's mouse moves over it. In the example below, '225' appears on top of the green bar as the cursor passes over it, indicating that the value of the bar is 225.



When **Draw on Front Plane** is selected, the data view will be mapped on the front plane of a three dimensional chart space. Applies only in cases where there are multiple data series, displayed on multiple axes, using 3D effects.

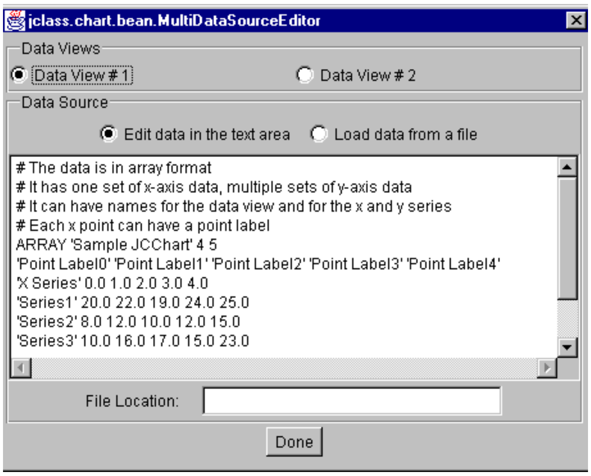
DataSource

There are three ways of loading data with the `MultiChart` Bean. Two are handled by this property: from a *.dat* file, or by entering data directly into the custom editor. Both methods are managed by the `DataSource` editor.

The third method is to use a Swing `TableModel`-type data object as a data source, instead of using the JClass Chart built-in data source. See `SwingDataModel` below for details.

The first step is to select a data view with one of the radio buttons. Then, follow the procedure below for each data view.

To load data from a file into a data view, click **Load data from a file**, enter the name of the file in the **File Location** field, and click **Done**:

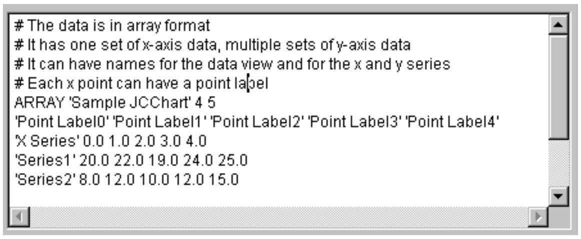


```
# The data is in array format
# It has one set of x-axis data, multiple sets of y-axis data
# It can have names for the data view and for the x and y series
# Each x point can have a point label
ARRAY 'Sample JCChart' 4 5
'Point Label0' 'Point Label1' 'Point Label2' 'Point Label3' 'Point Label4'
'X Series' 0.0 1.0 2.0 3.0 4.0
'Series1' 20.0 22.0 19.0 24.0 25.0
'Series2' 8.0 12.0 10.0 12.0 15.0
'Series3' 10.0 16.0 17.0 15.0 23.0
```

Specify the full path of the file. The file must be pre-formatted to the JClass Chart Standard (see Data Sources). Sample data files are located in the *JCLASS_HOME/jclass/chart/examples* directory.

You can use the data provided in the editor, as is, or you can modify it. To use existing data, just check the **Edit data in the text area** radio button, and click

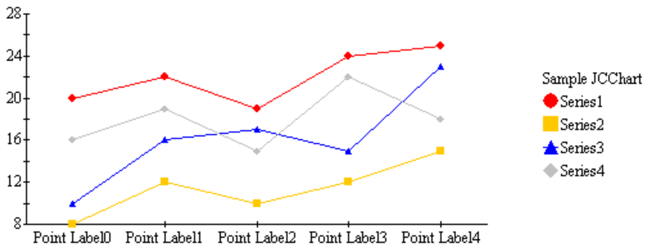
Done. Change data by deleting and inserting text in the area provided. Be careful to preserve the punctuation surrounding the original text:



```
# The data is in array format
# It has one set of x-axis data, multiple sets of y-axis data
# It can have names for the data view and for the x and y series
# Each x point can have a point label
ARRAY 'Sample JCChart' 4 5
'Point Label0' 'Point Label1' 'Point Label2' 'Point Label3' 'Point Label4'
'X Series' 0.0 1.0 2.0 3.0 4.0
'Series1' 20.0 22.0 19.0 24.0 25.0
'Series2' 8.0 12.0 10.0 12.0 15.0
```

The chart below shows how the default data for Data View 1 appears as a plot. Notice where the different elements are positioned. Each point on the x-axis is labelled with the names specified in the default data. The name of each series of y-values appears in the legend. The name of the data view is positioned directly above the legend.

In order for the default data to display this way, you must first set the `xAxisAnnotation` property to `Point_Labels`, and the `legendIsShowing` property to `true`.



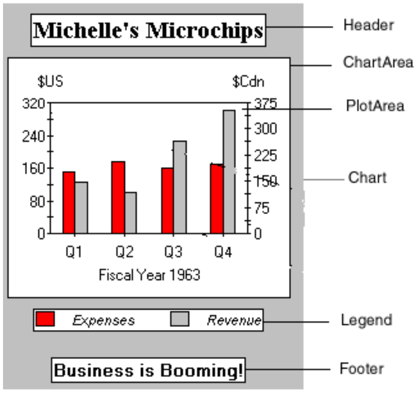
SwingDataModel

Instead of using the chart's internal data source, you can use a Swing `TableModel`-type data object that you have already created for your application. This saves reformatting your data to match the format used by JClass Chart.

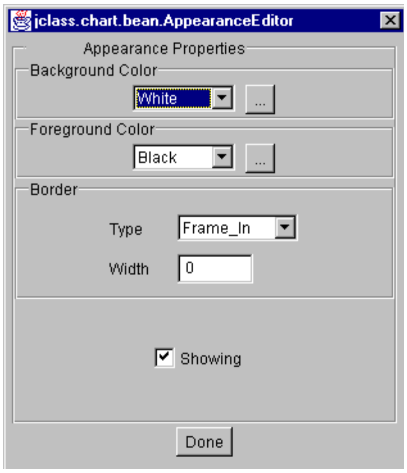
Use the `SwingDataModel1` property to specify an already-created Swing `TableModel` object to use as the data source for the first data view. Use `SwingDataModel2` to specify a `TableModel` object to use for the chart's second data view.

5.3.4 Appearance Controls

This group of editors allows you to control the look of specific chart areas. You can control font, borders, background and foreground for the chart, chart area, plot area, header, footer, and legend. The following diagram illustrates the different chart areas:.



All editors have the same basic functionality that apply to a specific chart area, as follows:



Background Color: White
Foreground Color: Black
Border
Type: Frame_In
Width: 0
[✓ Showing]

Small differences in each editor will be discussed below. Note that for most of the appearance editors, there are corresponding editors for controlling other properties of that chart element.

ChartAppearance

The `ChartAppearance` editor sets the foreground/background and borders for the chart. This editor affects the areas behind all other chart elements.

ChartAreaAppearance

The `ChartAreaAppearance` editor sets the foreground, background and borders for the chart area (see diagram above).

FooterAppearance

The `FooterAppearance` editor sets the foreground, background and borders for the footer. When **Showing** is checked, the footer will be displayed in the chart. By default the footer is not showing. The `FooterAppearance` editor works in conjunction with the FooterText editor, which is used to enter the footer text.

HeaderAppearance

The `HeaderAppearance` editor sets the foreground, background and borders for the header. When **Showing** is checked, a header will be displayed in the chart. By default the header is not showing. This editor works in conjunction with the HeaderText editor.

LegendAppearance

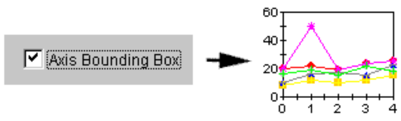
The `LegendAppearance` editor sets the foreground/background and borders for the legend and determines if it is displayed. When **Showing** is checked, a legend will be displayed in the chart.

The content of the legend come from the information in the data source. In order to change the contents of the legend, you have to change what is in the data source. For information on how to set up legend items in the data source, see Standard Data Formats., in the Data Sources chapter.

Other legend settings are found in the HeaderText editor.

PlotAreaAppearance

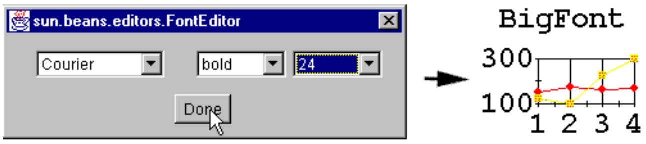
The `PlotAreaAppearance` editor sets the foreground and background for the plot area, and allows you to add an **Axis Bounding Box**. A bounding box is a graphical feature that 'closes off' the axes with two lines. forming a square.



Font

The `Font` editor sets the font defaults for your chart.

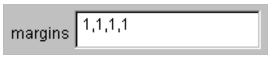
The font you choose will apply to all text on the chart simultaneously. The following example sets the font to **Courier, Bold, 24 point**:



This font editor sets up a default font for every element. You can, however, change font for selected elements using custom editors for each property. For example, the HeaderText, FooterText, and AxisAnnotation editors allow you to override the default font settings.

Margins

The margins property controls the distance from the data display to the chart area boundary:



margins 1,1,1,1

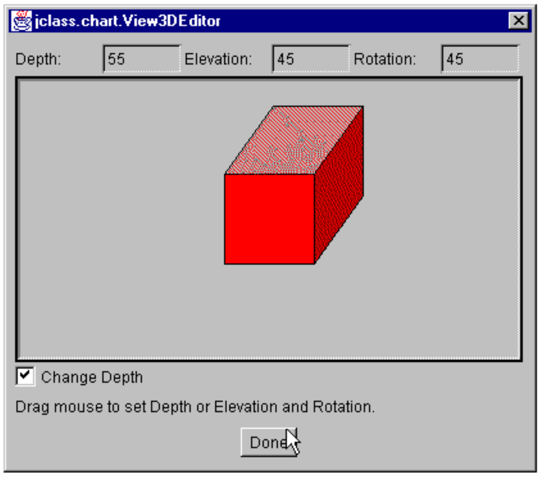
As you increase the margins, you shrink the data display area, including headers, footers, and the legend. The overall chart size remains the same.

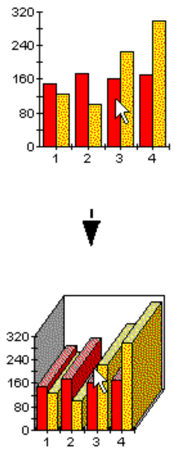
View3D

To add 3D effects to your chart, click the View3D property.

First drag the red square in the editor until it has the desired Elevation and Rotation. Then, check the **Change Depth** option box, and drag the red square until it has the

Depth you want to see on your chart. The degree of depth, elevation and rotation is displayed in numbers at the top of the editor. Click **Done** to set the changes:

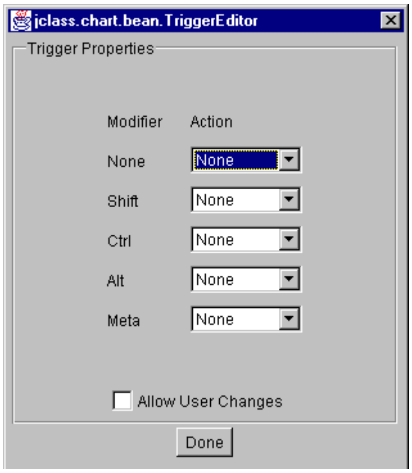

jclass.chart.View3DEditor
Depth: 55
Elevation: 45
Rotation: 45
Change Depth
Drag mouse to set Depth or Elevation and Rotation.
Done



5.3.5 Event Controls

TriggerList

The `TriggerList` editor sets up what user events the chart will handle, either from a mouse, or mouse-keyboard combination.



Modifier	Action
None	None
Shift	None
Ctrl	None
Alt	None
Meta	None

Actions are the available event mechanisms, such as `Zoom`, `Rotate`, `Depth`, `Customize`, `Pick` and `Translate`. By setting up these triggers, the end-user can examine data more closely or visually isolate part of the chart. The following list describes these interactions:

- **Translate** allows moving of the chart
- **Zoom** allows zooming into or out from the chart
- **Rotate** allows rotation (only for bar or pie charts displaying a 3D effect)
- **Depth** allows adding depth cues to the chart
- **Customize** allows the user to launch the chart Customizer. To use this feature, you must also check the **Allow User Changes** box.
- **Pick** allows you to set up pick events. The `pick` method is used to retrieve an *x,y* coordinate in a Chart from user input and then translate that into the data point nearest to it. This feature requires some non-bean programming. See Using Pick and Unpick for more details.

A **Modifier** is a keyboard event that can ‘modify’ a mouse click.

It is also possible in most cases for the user to reset the chart to its original display parameters. The interactions described here affect the chart displayed inside the `ChartArea`; other chart elements like the header are not affected.

6

Chart Programming Tutorial

6.1 Introduction

This tutorial shows you how to start using JClass Chart, by compiling and running an example program. It is different from the SimpleChart Bean tutorial, because it focusses on programmatic use of JClass Chart. For a Bean tutorial, see the SimpleChart Bean Tutorial. This program, *plot1.java*, will graph the 1963 Quarterly Expenses and Revenues for "Michelle's Microchips", a small company a little ahead of its time.

The following table shows the data to be displayed:

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

6.2 A Basic Plot Chart

The following listing displays the program *plot1.java*. This is a minimal Java program that creates a new chart component and loads data into it from a file. It can be run as an applet or a standalone application. The source code can be found in the JClass Chart distribution in the *JCLASS_HOME/jclass/chart/examples* directory.

```
1   package jclass.chart.examples;
2
3   import java.awt.GridLayout;
4   import jclass.chart.JCChart;
5   import jclass.chart.ChartDataView;
6   import jclass.chart.FileDataSource;
7
8   public class plot1 extends java.applet.Applet {
9
10  public void init() {
11      setLayout(new GridLayout(1,1));
12
13      try {
14          JCChart c = new JCChart();
15          c.getDataView(0).setDataSource(new
                                FileDataSource("plot1.dat"));
16          add(c);
17      }
18      catch (Exception e) {
19          e.printStackTrace(System.out);
20      }
21  }
22
23  public static void main(String args[]) {
24      ExampleFrame f = new ExampleFrame("plot1");
25      plot1 p = new plot1();
26      p.init();
27      f.setLayout(new GridLayout(1,1));
28      f.add(p);
29      f.pack();
30      f.resize(400, 300);
31      f.show();
32  }
33
34  }
```

Most of the code in *plot1.java* should be familiar to Java programmers. The first few lines (3–6) import the classes necessary to run *plot1.java*. In addition to the standard AWT `GridLayout` class, three classes in the `jclass.chart` package are needed: `JCChart` (the main chart class), `ChartDataView` (the data view object) and `FileDataSource` (a stock data source). Line 8 provides the class definition for this program, a subclass of `java.applet.Applet`.

Lines 10–21 define the `init()` method, used when the program is run as an applet. The `Layout` property on line 11 lays out a simple grid structure to display the components it holds. A new chart is then instantiated on line 14. Line 15 loads data from a file named *plot1.dat* into a new data source object (`FileDataSource`) and tells the chart to display this data.

Lines 23–31 define the `main()` method needed when the program is run as a standalone Java application. Notice that it calls the `init()` method defined earlier to create the chart and add the data.

When *plot1.java* is compiled and run, the window shown below is displayed:

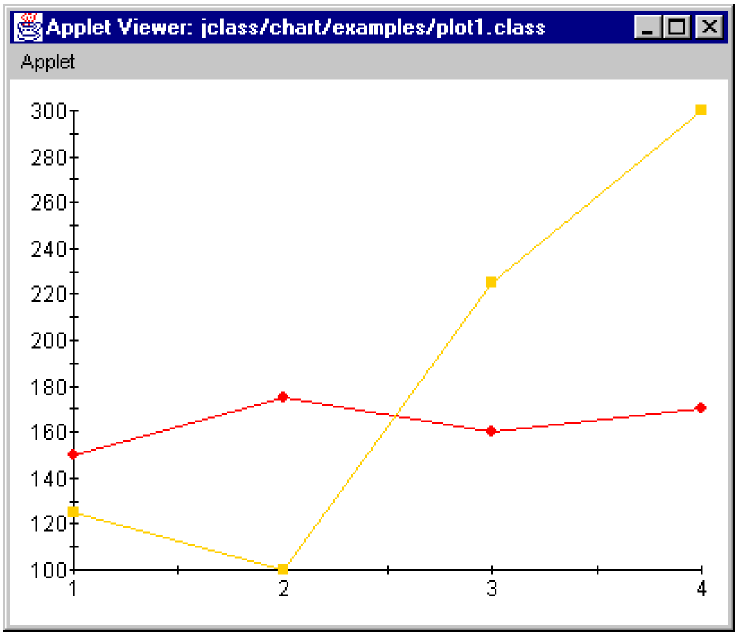


Figure 11   The plot1.java program displayed

JClass Chart supports ten different types of charts: Plot, Scatter Plot, Area, Stacking Area, Bar, Stacking Bar, Pie, Hi-Lo, Hi-Lo-Open-Close, and Candle. Because Plot is the default, we do not need to set a property in this example.

6.3 Loading Data From a File

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

JClass Chart includes several stock (built-in) data sources that you can use (or you can define your own). This program uses the data source that reads data from a file: `FileDataSource`. With this understanding we can look more closely at line 15:

```
15          c.getDataView(0).setDataSource(new
                                 FileDataSource("plot1.dat"));
```

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

The following shows the contents of the *plot1.dat* file:

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

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

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

For complete details on using data with JClass Chart, see Chapter 8, Data Sources.

6.4 Adding Header, Footer and Labels

The plot displayed by *plot1.java* is not very useful to an end-user. There is no header, footer, or legend, and the X-axis numbering is not very meaningful.

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

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

Adding Headers and Footers

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

```
c.getFooter().setIsShowing(true);
c.getFooter().getLabel().setText(
        "1963 [COLOR=BLUE]Quarterly[DEFAULT_COLOR] Results", true);
```

`IsShowing` displays the header/footer. `Text` specifies the text displayed in the header/footer. The footer text uses `JCString` properties to change the color of the word "Quarterly" in the footer. JCStrings enable an application to display text containing images, URLs, or richly-formatted text. More information on JCStrings can be found in Appendix B, JCString Properties.

Adding a Legend and Labelling Points

A legend clarifies the chart by showing a label for each series in the chart. We would also like to display more meaningful labels for the points along the X-axis. Both types of information can be easily specified in the data file itself. The following lists *plot2.dat*, a modified version of the previous data file that includes series labels (for the legend), and point labels (for the X-axis):

```
ARRAY '' 2 4
'Q1' 'Q2' 'Q3' 'Q4'
'' 1.0  2.0  3.0  4.0
'Expenses' 150.0 175.0 160.0 170.0
'Revenue' 125.0 100.0 225.0 300.0
```

The second line specifies the point labels (“`Q1`”, “`Q2`”, etc.). Subsequent lines of data begin with a data series label (“`Expenses`”, “`Revenue`”, etc.).

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

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

```
JCChart c = new JCChart();
c.getDataView(0).setDataSource(new FileDataSource("plot2.dat"));
c.getHeader().setIsShowing(true);
c.getHeader().getLabel().setText("Michelle's Microchips", false);
c.getFooter().setIsShowing(true);
c.getFooter().getLabel().setText(
        "1963 [COLOR=BLUE]Quarterly[DEFAULT_COLOR] Results", true);
c.getLegend().setIsShowing(true);
c.getChartArea().getXAxis(0).setAnnotationMethod(
                                JCAxis.POINT_LABELS);
add(c);
```

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

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

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

The chart resulting from these changes is displayed below. Full source code can be found in the *plot2.java* program, located in the *JCLASS_HOME/jclass/chart/examples* directory.

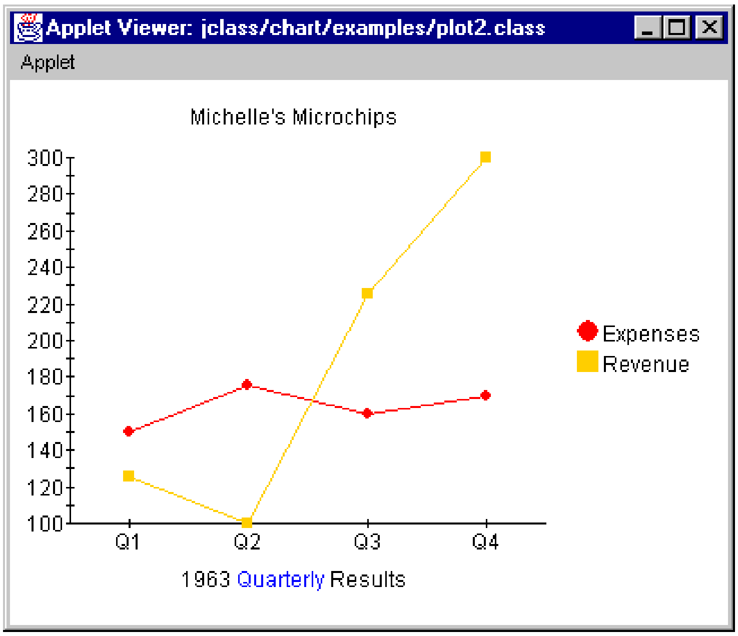


Figure 12 The program created by plot2.java

6.5 Changing to a Bar Chart

A powerful feature of JClass Chart is the ability to change the chart type independently of any other property.[1] For example, to change the plot2 chart to a bar chart, the following code can be used:

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

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

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

1. Although there are interdependencies between some properties, most properties are completely orthogonal.

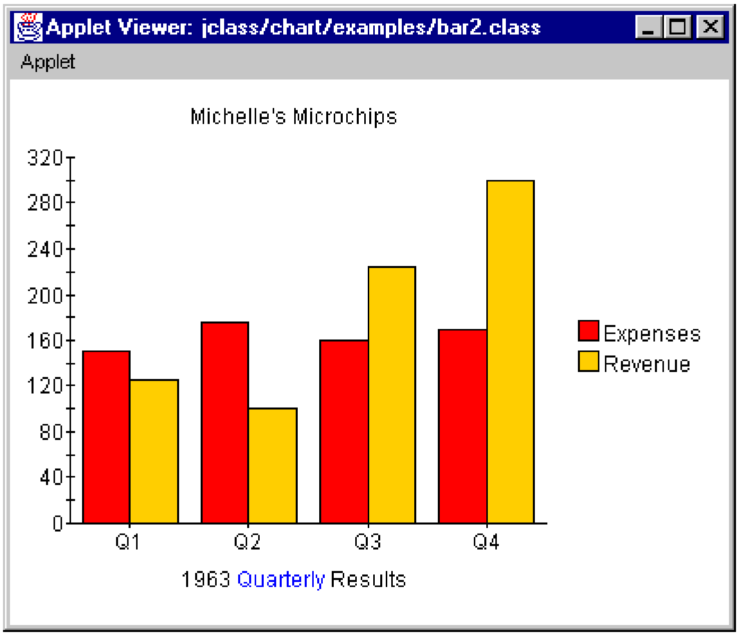


Figure 13   The bar2.java program displayed

The full code for this program (*bar2.java*) can be found in with the other examples.

JClass Chart can display data as one of ten different chart types. For more information on chart types, see Chart Types on page 20.

6.6 Inverting Chart Orientation

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

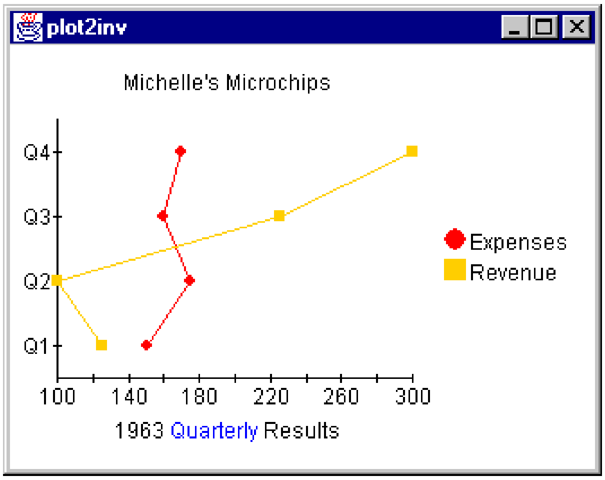
In a plot, inverting causes the Y-values to be plotted against the horizontal axis, and the X-values to be plotted against the vertical. In a bar chart, it causes the bars to be displayed horizontally instead of vertically.

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

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

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

The following shows the windows created by *plot2.java* and *bar2.java* when inverted:



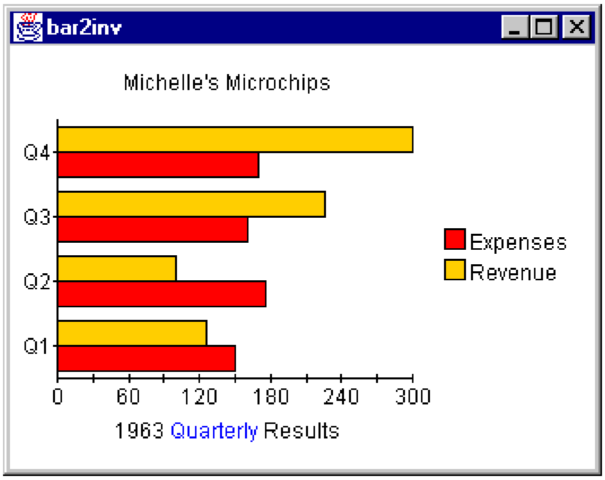


Figure 14   plot2 and bar2 windows with IsInverted set to true

Full code for these examples can be found in the *JCLASS_HOME/jclass/chart/examples/* directory.

6.7 End-User Interaction

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

- Translation–users can move a graph or a series of graphs along the X- and or Y-axes.
- Rotate–users can change the vantage point of a chart type, to better view information contained with a JClass Chart component.
- Zoom–users can zoom in or out of a JClass Chart component to better view information contained with a JClass Chart component.
- Depth–users can change the apparent depth of a 3D chart.
- Pick–users can change the placement of data points within a chart.
- Edit–users can alter the other display features of a chart, (such as color, label names or the numerical value of data points) that comprise a chart display.

6.8 Get Started Programming with JClass Chart

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

- Check out the sample code – the example and demo programs included with JClass Chart are useful in showing what JClass Chart can do, and how to do it. Run them and examine the source code. They can all be found in the *JCLASS_HOME/jclass/chart/demos/* and *JCLASS_HOME/jclass/chart/examples/* directories. A collection of HTML-only charts can be found in the *JCLASS_HOME/jclass/chart/applet/* directory.
- Browse the JClass Chart API documentation – complete reference documentation on the API is available online in HTML format, generated by *javadoc*. All of the properties, methods and events for each component are completely documented.

7

Axis Controls

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

If you developing your chart application using one of the JClass Chart Beans, please refer to the Bean Reference chapter instead.

7.1 Creating a New Chart in a Nutshell

1. If one exists, use an existing chart as a starting point for the new one. The sample charts provided as part of the JClass Chart package are a good starting point. Load a chart description resembling the new chart.
2. Load your data into the chart.
3. Set the chart type.
4. Annotate and format the axes and data if necessary, described as follows:
   - Axis annotation (`Values` (default), `ValueLabels`, `PointLabels`, `TimeLabels`
   - Positioning Axis Annotations
   - Chart Orientation and Axis Direction
   - Setting Axis Bounds
   - Customizing Origins
   - Logarithmic Axes

- Titling Axes and Rotating Axis Elements
- Adding Grid Lines
- Adding a Second Axis.

7.2 Axis Labelling and Annotation Methods

There are several ways to annotate the chart's axes, each suited to specific situations. The chart can automatically generate numeric annotation appropriate to the data it is displaying; you can provide a label for each point in the chart (X-axis only); you can provide a label for specific values along the axis; or the chart can automatically generate time-based annotation.[1]

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.

7.2.1 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.VALUE` (default)	The chart chooses appropriate axis annotation automatically (with possible callbacks to a label generator), based on the chart type and the data itself.
`JCAxis.POINT_LABELS` (X-axis only)	The chart spaces the points based on the X-values and annotates them with text you specify (in the data source) for each point.
`JCAxis.VALUE_LABELS`	The chart annotates the axis with text you define for specific X-or Y-axis coordinates.
`JCAxis.TIME_LABELS`	The chart interprets the X- or Y-values as units of time, automatically choosing time/date annotation based on the starting point and format you specify.

Note: Point-labels annotation (`JCAxis.POINT_LABELS`) is only valid for an X-axis when it has been added to the X-axis collection in `JCChartArea`. This means that a new `JCAxis` instance that has not yet been added to `JCChartArea` will not be considered an X-axis.

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

1. None of the axis properties discussed in this section apply to pie charts, since pie charts don't have axes.

7.2.2 Values Annotation

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

Numbering Precision

Use the `Precision` axis property to set the number of decimal places to use when displaying each number. The `PrecisionIsDefault` property allows the chart to automatically determine precision based on the data. The effect of `Precision` depends on whether it is positive or negative:

- *Positive* values add that number of places after the decimal place. For example, a value of 2 displays an annotation of 10 as “10.00”.
- *Negative* values indicate the minimum number of zeros to use before the decimal place. For example, a value of -2 displays annotation in multiples of 100.

The default value of `Precision` is calculated from the data supplied.

Numbering and Ticking Increments

Use the `NumSpacing` axis property to set the increment between labels along an axis. The `NumSpacingIsDefault` property allows the chart to automatically determine the increment.

Use the `TickSpacing` axis property to set the increment between ticks along an axis. This should generally divide equally into `NumSpacing` on axes using `Values` annotation. The `TickSpacingIsDefault` property allows the chart to automatically determine the increment. Note that if the `AnnotationMethod` property is set to `POINT_LABELS`, tick lines appear at point values.

7.2.3 PointLabels Annotation

`PointLabels` annotation displays defined labels along an X-axis. This is useful for annotating the X-axis of any chart using `Array` data layout, including bar, stacking bar, and pie charts. It is possible to add, remove, and edit `PointLabels`. In JClass Chart, `PointLabels` are typically defined with the data.

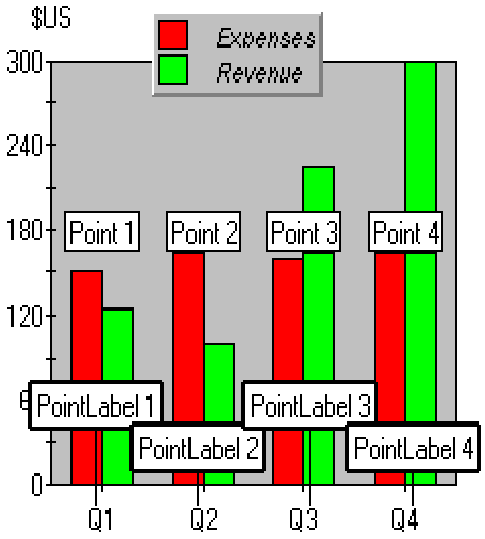


Figure 15 PointLabels X-axis annotation

`PointLabels` are a collection of labels. The first label applies to the first point, the second label applies to the second point, and so on.

The labels can also be supplied by setting the `PointLabels` property of the `ChartDataView` object for this chart. For example, the following code specifies labels for each of the three points on the X-axis:

```
c.getChartArea.getxAxis(0).setAnnotationMethod(JCAxisPOINT_LABELS);
ChartDataView cd = c.getDataView(0);
cd.setPointLabel(0, "Point 1");
cd.setPointLabel(1, "Point 2");
cd.setPointLabel(2, "Point 3");
```

7.2.4 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.

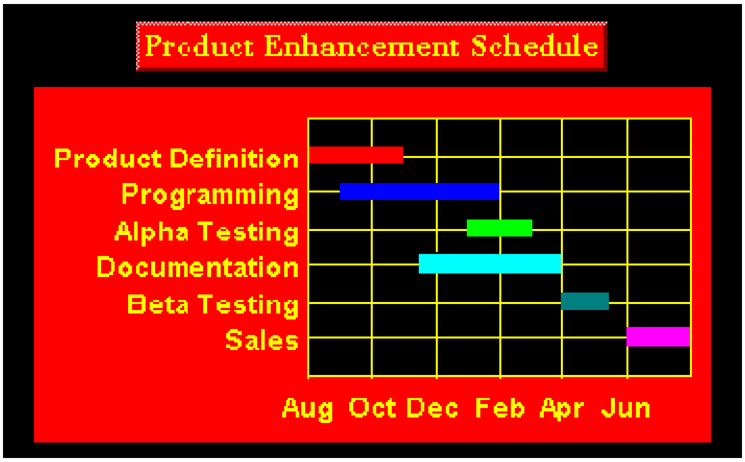


Figure 16 Using ValueLabels to annotate axes

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

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

```
JCAxis x=c.getChartArea.getXAxis(0);
x.setValueLabel(0, new JCValueLabel(10, 0, "Label"));
x.setValueLabel(1, new JCValueLabel(20, 0, "Label 2"));
x.setValueLabel(2, new JCValueLabel(30, 0, "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.getChartLabelArea().getXAxis(0);
    getValueLabel(2);
// this retrieves the label at chart coordinate 2.0
v1=c.getChartLabelArea().getXAxis(0);
    getValueLabel(2.0);
```

7.2.5 TimeLabels Annotation

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

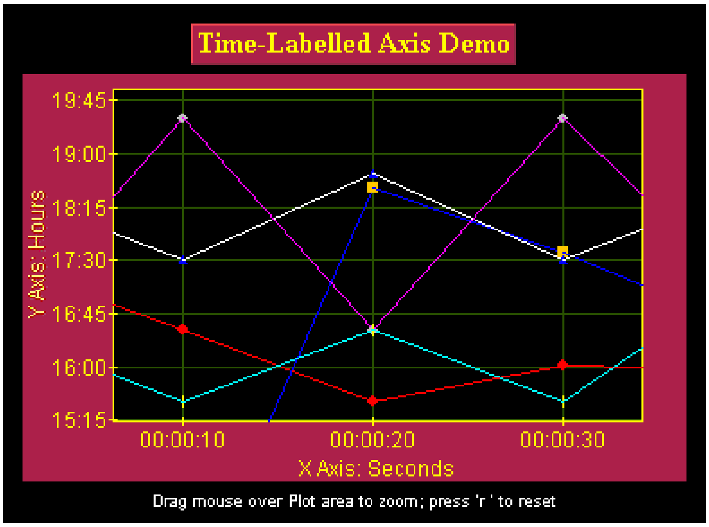


Figure 17   TimeLabels annotating X- and Y-axes

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

- `TimeUnit`
- `TimeBase`
- `TimeFormat`
- `AnnotationMethod` (set to `JCAxis.TIME_LABELS` to use this annotation method)

Time Unit

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

Time Base

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

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

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

Time Format

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

`TimeFormat` specifies a *time format*. You build a time format using special format codes. The chart displays only the parts of the date/time specified by `Format`. The following lists all of the valid format codes:

Format Code	Meaning
%a	Weekday name, abbreviated
%A	Weekday name
%b	Month name, abbreviated
%B	Month name
%c	Appropriate date/time representation
%d	Day of month (01 to 31)
%H	Hour (0 to 23)
%I	Hour (0 to 12)
%m	Month number (01 to 12)
%M	Minute of hour (00 to 59)
%p	AM or PM
%S	Seconds (00 to 61)
%U	Week number of year (00 to 53), Sunday first day of week 1
%w	Weekday number (0 to 6), Sunday = 0
%x	Appropriate date representation
%X	Appropriate time representation
%y	Year number within century (00 to 99)
%Y	Year

The default for `TimeFormat` is calculated based on the value for `TimeUnit`.

Using Date Methods

The `dateToValue()` method converts a Java date value into a time-axis position (a floating-point value). To convert a time-axis position to a date, use the `valueToDate()` method. For example:

```
    JCAxis y;
    Date d = y.valueToDate(3.0);
    double val = y.dateToValue(new Date(90,1,0));
```

To convert a date to a formatted text string, use the `timeLabel()` method (defined in `JCChartTimeUtil`). This method takes a format string and a Java `Date` object.

7.3 Positioning Axis Annotations

Axis annotation typically appears beside its axis. This may be a problem on charts with an origin that is not at the axis minimum or maximum. The chart can automatically determine where to place annotation in different situations, depending on the chart type. If this does not give the desired results, specify where to place the annotation for each axis.

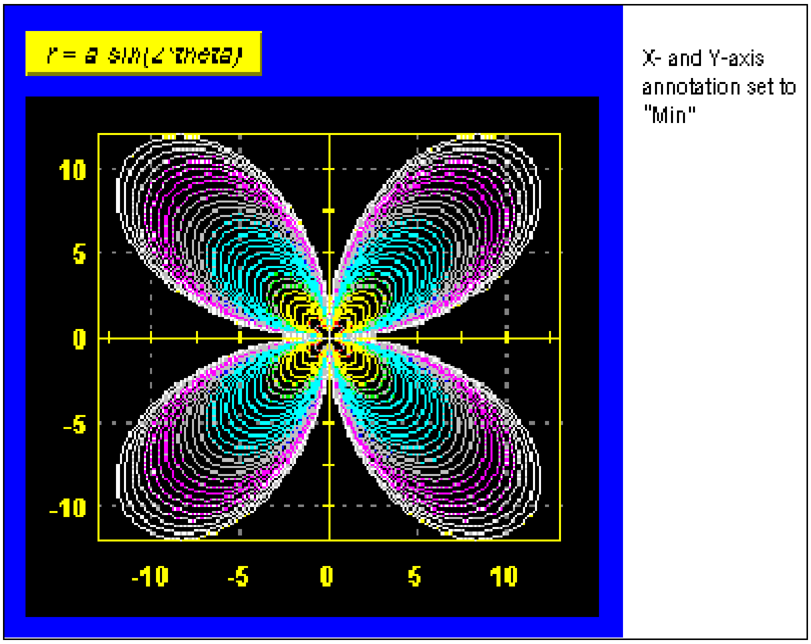


Figure 18 X- and Y-axes set to Min

7.3.1 Specifying Annotation placement

Use the `Placement` property to specify annotation placement for an axis. Use the `PlacementIsDefault` property to specify whether the chart determines the annotation placement. When `PlacementIsDefault` is set to `true`, annotations on plot charts are placed at the origin; on bar and stacking bar charts they are placed at the end of the axis closest to the origin. You can also explicitly place annotations at the origin, the axis minimum, or the axis maximum.

Note: In some cases, the chart may change the placement. For example, `Placement` for the Y-axis is ignored on bar and stacking bar charts and plot and area charts when using `PointLabels` annotation. And, when `Placement` is set to `Origin`, changing the axis origin will move axes placed at that origin.

7.4 Chart Orientation and Axis Direction

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

7.4.1 Inverting Chart Orientation

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

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

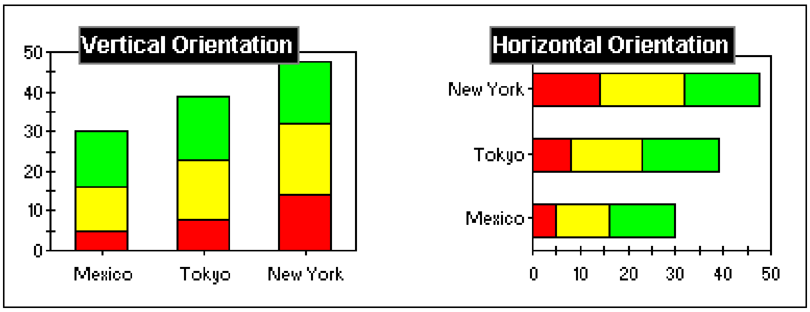


Figure 19 Normal and inverted orientation

7.4.2 Changing Axis Direction

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

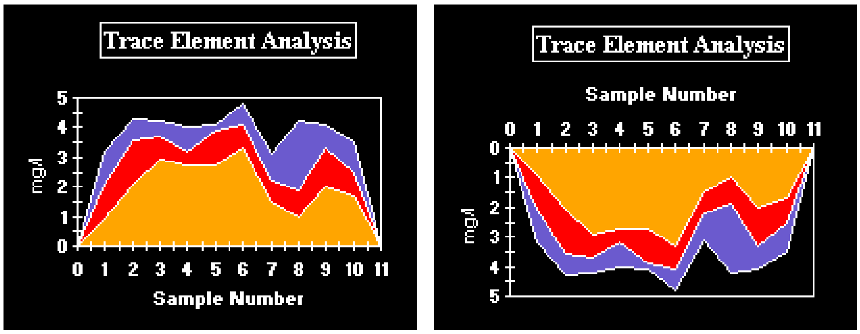


Figure 20   Two charts depicting a normal and reversed Y- axis

7.5   Setting Axis Bounds

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

Min and Max

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

7.6   Customizing Origins

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

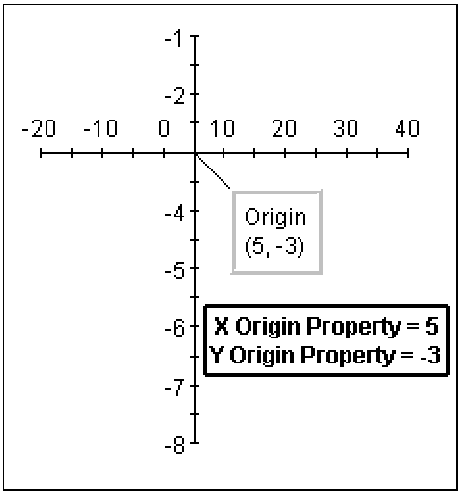


Figure 21 Defining origins for X- and Y-axes

Origin Placement

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

Origin Coordinates

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

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

7.7 Logarithmic Axes

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

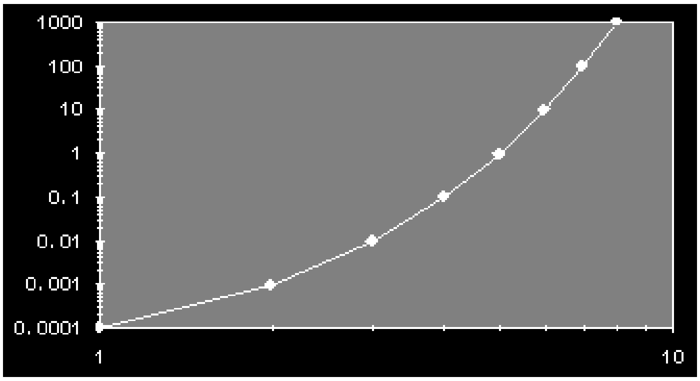


Figure 22 Logarithmic X- and Y-axes

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

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

Specifying a Logarithmic Axis

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

Note: Pie charts are not affected by logarithmic axes.

7.8 Titling Axes and Rotating Axis Elements

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

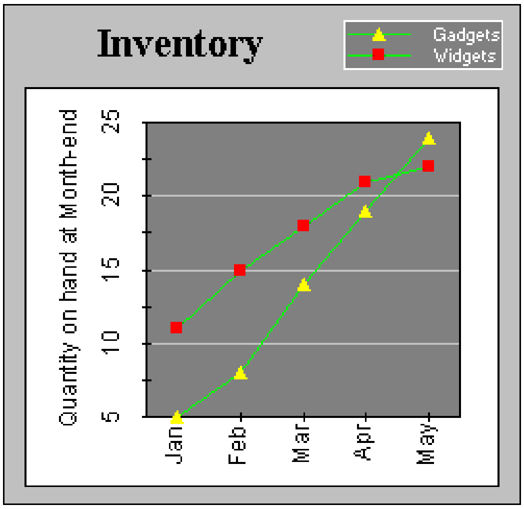


Figure 23 Rotated axis title and annotation

Adding an Axis Title

Use the `Title` property to add a title to an axis. It sets the `JCAxisTitle` object associated with the `JCAxis`. `JCAxisTitle` controls the appearance of the axis title. `JCAxisTitle`'s `Text` property specifies the title text. The text can be either a regular string, or a JCString. For more information on JCStrings, see Appendix B, JCString Properties.

Axis Title Rotation

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

Rotating Axis Annotation

Use the `AnnotationRotation` property of `JCAxis` to rotate the axis annotation to either 90, 180, or 270 degrees counterclockwise. 270-degree rotation usually looks best on second Y-axis.

7.9 Adding Grid Lines

Displaying a grid on a chart can make it easier to see the exact value of data points. The spacing between lines on the grid can be defined to determine how a grid is displayed.

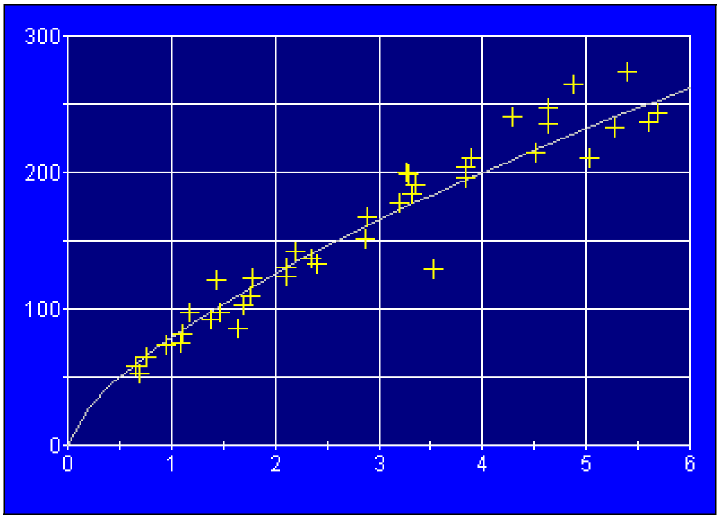


Figure 24 JClass Chart illustrating the effects of grid lines

Horizontal gridlines are a property of the Y-axis. Vertical gridlines are a property of the X-axis. Set `GridIsShowing` to `true` to display gridlines.

Grid Spacing

Use the `GridSpacing` property to customize the grid spacing for an axis. The `GridSpacingIsDefault` property allows the chart to space the grid automatically, drawing a gridline wherever there is annotation. By default, gridlines will correspond with axis annotations.

Grid Appearance

Use the grid `GridStyle` properties to customize the line pattern, thickness, and color of the gridlines. The following code fragment provides a sample of `GridStyle` and `GridIsShowing` used within a program:

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

7.10 Adding a Second Axis

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

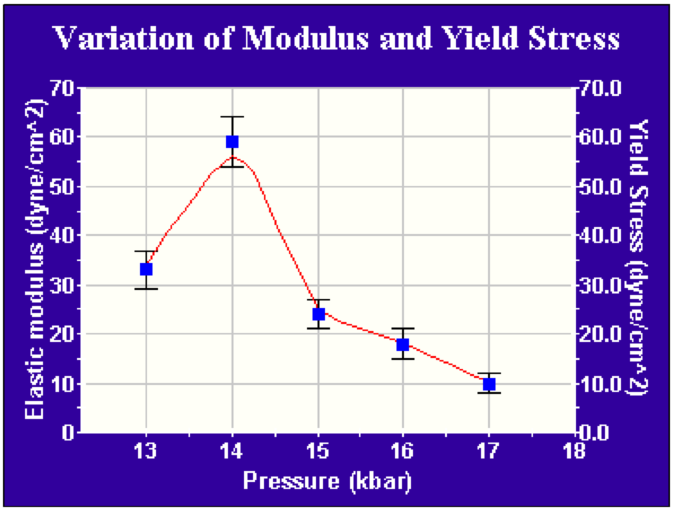


Figure 25 Chart containing multiple Y-axes

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

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

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

Defining Axis Multiplier

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

Using a Constant Value

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

Hiding the Second Axis

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

Other Second-Axis Properties

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

8

Data Sources

8.1 Overview

Data is loaded into a chart by attaching one or more chartable 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 can use. Once your data source is attached, you can chart the data in a variety of ways.

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

Creating your own data sources can be time consuming, however. For that reason, JClass Chart provides pre-built chartable data sources for most common real-world data: files, URL's, Applets, strings, and databases.

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

8.2 File Data Source

An easy way to bring data into a chart is to load it from a formatted file using `FileDataSource`. To load data this way, you create a data file that follows JClass Chart's standard format (see Standard Data Formats.)

Then, you instantiate a `FileDataSource` object and attach it to a view in your chart application. And that's it. The following example shows how to instantiate and attach a `FileDataSource`:

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

8.3 URL Data Source

You can chart data from a URL address using `URLDataSource`. To load data this way, you create a data file that follows JClass Chart's standard format (see Standard Data Formats.)

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

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

Parameter options for URLDataSource:
The following are valid parameter combinations for `URLDataSource`:

- URL
- base, file
- host, file

host: The WWW hostname
file: The fully-qualified name of the file on the server
URL: The URL address of a data file. e.g. *http://www.klg.com/datafile.dat*
base: A URL object representing the directory where the file is located

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

8.4 Applet Data Source

You can chart data from an Applet using `AppletDataSource`.

To prepare the data, put it into the standard format, (see Standard Data Formats), and insert it into the HTML file that calls your Applet. In most cases, that will be i*ndex.html*. The HTML syntax is as follows:

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

‘Your_Data_Name’ is used by your Applet to select the right set of information. Use the same name in the Applet and the HTML source.

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

```
    chart.getDataView(0).setDataSource(new
          AppletDataSource(“Your_Data_Name"));
```

Example of Data in an HTML file

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

<PARAM NAME=data VALUE="

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

8.5 ChartSwingDataSource for Swing TableModel

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

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

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

8.6 Standard Data Formats

`FileDataSource`, `URLDataSource`, and `AppletDataSource` all require that data be pre-formatted. The following table illustrates the formatting requirements of data for pre-built data sources. There are two main ways to format data: Array and General.

Array Format is the recommended standard, because it works well with all of the chart types. General Format may not display data properly in Stacking Bar, Stacking Area, Pie charts and Bar Charts.

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

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

8.6.1 Formatted Data Examples

Array Data Format (Recommended)

```
ARRAY 2 3                         # 2 series of 3 points
HOLE 10000                      # Use only if custom hole value needed
'Point 0' 'Point 1' 'Point 2'     # Optional Point-labels
# X-values common to all points
             1.0   2.0   3.0
# Y-values
'Series 0'  50.0  75.0  60.0      # Series-label is optional
'Series 1'  25.0  10.0  50.0
```

Transposed Array Data Format (same data as previous)

```
ARRAY 2 3 T                       # 2 series of 3 points, Transposed
HOLE 10000
  ''                 'Series 0' 'Series 1'   # Optional Series-labels
#          X-values  Y0-values  Y1-values
'Point 0'    1.0       50.0       25.0     # Point-labels are optional
'Point 1'    2.0       75.0       10.0
'Point 2'    3.0       60.0       50.0
```

General Data Format (Use if X data is different for each series)

```
GENERAL 2 4                 # 2 series, max 4 points in each
HOLE -10000                 # Use only if custom hole value needed
'Series 0' 2                # 2 points, optional series label
  1.0  3.0                  #   X-values
 50.0 60.0                  #   Y-values
'Series 1' 4                # 4 points
  2.0   2.5   3.5   5.0     #   X-values
 45.0  60.0  HOLE  70.0     #   Y-values, including data hole
```

Transposed General Data Format (same data as previous)

```
GENERAL 2 4 T            # 2 series, max 4 points in each, Transposed
HOLE -10000
'Series 0' 2             # 2 points, optional series label
# X   Y
 1.0 50.0
 3.0 60.0
'Series 1' 4             # 4 points
# X   Y
 2.0 45.0
 2.5 60.0
 3.5 HOLE
 5.0 70.0
```

8.6.2 Explanation of Format Elements

Initialization — Data Layout, Data Size, Hole Value

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

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

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

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

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

```
HOLE 10000
```

You can also indicate that any particular point is a hole by specifying the word "`HOLE`" for that X- or Y-value. For example:

```
50.0 75.0 HOLE 70.0
```

Adding Comments

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

Point Labels

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

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

The Data — Array layout

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

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

The Data — General layout

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

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

Series Labels

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

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

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

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

Transposed Data

JClass Chart can also interpret transposed data, where the meaning of the data series and points is switched. Note that transposing data also transposes series and point labels. To indicate that the data is transposed, add a "T" to the first line specifying the data layout and size. The following illustrates how data is interpreted when transposed:

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

8.7 Data Binding: Loading Data from Databases

In order to chart data from a database, your application must be able to establish a connection, perform necessary queries on the data, and then put the data into a chartable format.

This type of database connectivity is often called 'data binding' and components that can be connected to a database are considered 'data bound'. JClass Chart is a data bound component.

Perhaps the easiest way to bind a chart to a database is to use one of the data binding Beans (`DSdbChart`, `VCdbChart`, `JBdbChart`) in an IDE or the BeanBox. There are Beans for connecting to a database using Borland JBuilder, Visual Café, and the JClass DataSource. See the Bean Reference for complete details on using these Beans in an IDE.

More complex chart features, however, can only be accessed programmatically. To do data binding programmatically, you can use one of the solutions listed in the table below:

Class	Use with:
JCChart	■ JDBCDataSource ■ An application that provides connection to database and passes an SQL result set to JDBCDataSource.
DSdbChart	■ JClass DataSource component
JBdbChart	■ Borland JBuilder 2.0 components
VCdbChart	■ Visual Café 2.5 components

The following sections provide a brief outline of these different data binding methods.

8.7.1 Data Binding using JDBCDataSource

`JDBCDataSource` is not a full data binding solution. It ia 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.

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

```
chart.getDataView(0).setDataSource(new JDBCDataSource(ResultSet));
```

8.7.2 Data Binding with Visual Cafe

VCdbChart allows you to bind to Visual Café's QueryNavigator, for a full data binding solution. The following example illustrates how to connect to the necessary Visual Cafe components:

```
{
   //{{INIT_CONTROLS

   //Cafe data binding object setup
   setLayout(null);
   setSize(672,393);
   Products_Navigator = new QueryNavigator();
   Products_Navigator.setAutoStart(true);
   Products_Navigator.setClassName("Products_Record");
   Products_Navigator.setAliasName("Applet1_QNAlias");

   //Create Visual Cafe databinding version of JClass Chart
   vCdbChart1 = new jclass.chart.db.vcafe.VCdbChart();
   vCdbChart1.setXAxisAnnotationMethod
(jclass.chart.JCAxis.POINT_LABELS);

   // Specify columns for point labels and data series
   vCdbChart1.setDataBindingMetaData(new
jclass.chart.db.DataBindingMetaData
       ( "ProductID",new String[] {"UnitPrice"},0,100));
   //Connect chart to QueryNavigator instance
   vCdbChart1.setDataBinding("Applet1_QNAlias@UnitPrice,ProductID");
   vCdbChart1.setBounds(60,24,552,288);
   add(vCdbChart1);
   //}}
   }
```

8.7.3 Data Binding with JBuilder

`JBdbChart` allows you to bind to JBuilder's DataSet, for a full data binding solution. The following example illustrates how to connect to the necessary JBuilder components:

```
public class Frame1 extends DecoratedFrame {
  Database database1 = new Database();
  QueryDataSet queryDataSet1 = new QueryDataSet();
  jclass.chart.db.jbuilder.JBdbChart jBdbChart1 = new
      jclass.chart.db.jbuilder.JBdbChart();

...

private void jbInit() throws Exception {
    this.setSize(new Dimension(792, 593));
    database1.setConnection(new
      borland.sql.dataset.ConnectionDescriptor
      ("jdbc:odbc:JClassDemoSQLAnywhere", "dba", "sql", false,
      "sun.jdbc.odbc.JdbcOdbcDriver"));
    queryDataSet1.setQuery(new
        borland.sql.dataset.QueryDescriptor(database1, "select * from
        OrderDetails", null, true, Load.ALL));

jBdbChart1.setXAxisAnnotationMethod(jclass.chart.JCAxis.POINT_LABELS);
    jBdbChart1.setDataSet(queryDataSet1);
    jBdbChart1.setDataBindingMetaData(new
      jclass.chart.db.DataBindingMetaData( "PRODUCTID",new String[]
      {"QUANTITY","SALESTAX"},0,100));
    this.add(jBdbChart1, BorderLayout.CENTER);
  }
}
```

8.7.4 Data Binding with JClass DataSource

JClass DataSource is a full data binding solution. It is a robust hierarchical, multiple-platform data source that you can use to bind and query any JDBC compatible database. It can also bind to platform-specific data solutions in JBuilder and Visual Café. It comes with JClass Enterprise, JClass HiGrid, or can be purchased separately.

To bind a chart to a database through JClass DataSource, use `DSdbChart`.

The following example illustrates the main parts of binding with DSdbChart:

- The variable declaration.
- Setting the data source in the Chart using ChartDataView
- The contents of makeChartDataSource().

```
private JCChart chart = null;
private TreeData treeData = null;// this is the database
private DataBoundSource dbSource = null; // chart's datasource linked to
  a db (i.e. treeData)
private ChartDataBinding navigator = null;

  ...

makeChartDataSource();
chart.getDataView( 0 ).setDataSource( dbSource );

 ...

private void makeChartDataSource() {ChartDataBinding chartDataBinding =
  new ChartDataBinding();

chartDataBinding.setDataBinding( treeData, "Orders|OrderDetails" );
dbSource = new DataBoundSource(
  chartDataBinding.getDataBindableObject() );
chartDataBinding.addDBUpdateListener( dbSource );
dbSource.addChartDataListener( chart );
dbSource.setSeriesColumns( new String[] {"TotalLessTax","SalesTax"} );
dbSource.setPointLabelsColumn( "DateSold" );
dbSource.setName("Order Details");
}
```

Note that treeData is a `TreeData` instance, and should be configured separately. Consult the JClass DataSource documentation for details.

8.8 Using Multiple Data Sources

A chart can connect to and display more than one data source. The data from different sources can even be plotted on the same set of axes. The `ChartDataView` object properties control which data source to display, and how to display the data.

To use multiple data sources:

- Each data source must be assigned to a separate ChartDataView.
- Each ChartDataView must be assigned an x and y axis.

Note that while plotting multiple ChartDataViews against the same x or y axis, JClass Chart will adjust the range of the axes to the size of the union of the two ranges. See Axis Controls, for more details on controlling axis settings.

8.9 The Data Model

The following diagram illustrates the main classes and interfaces for JClass Chart data sources:

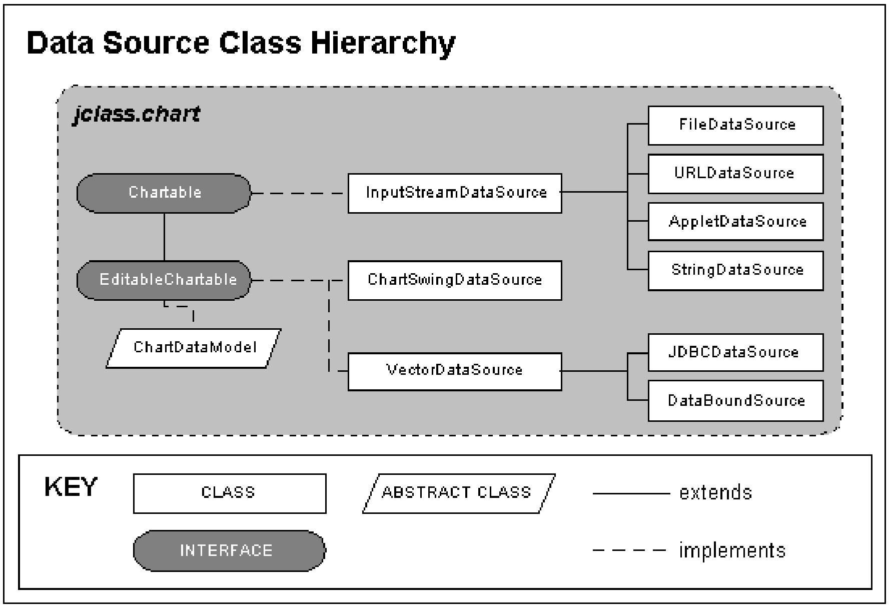


Figure 26 Class hierarchy of the jclass.chart package data sources

The Chartable Interface

In order for a data source object to work with JClass Chart, it must implement the `Chartable` interface. The `Chartable` interface is intended for use with existing data objects. `Chartable` assumes that data is available in table form, and is well-suited to retrieve data that originates from databases or other “flat” files.

The ChartDataView

The `ChartDataView` object contains a collection of `ChartDataViewSeries` objects; one for each data series in the data view. `ChartDataView` and `ChartDataViewSeries` are used to control how the data is used by JClass Chart. This task is performed by the data source. `DataSource` is a property of `ChartDataView` that points to the data source. In JClass Chart, the `DataSource` is an object derived from `Chartable`, `EditableChartable` or `ChartDataModel`.

The `Chartable` interface contains elements for retrieving data. An object that implements the `Chartable` interface can be attached to `ChartDataView` as a data source using the DataSource property. ChartDataView uses its DataSource property to retrieve data for display. Example uses of Chartable can be found throughout the demos. The basic demo (*JCLASS_HOME/jclass/chart/demos/basic/*) contains *ArrayData.java* and *GeneralData.java*, two code samples which contain good examples of data sources that use the Chartable interface.

EditableChartable Interface

The EditableChartable interface extends the Chartable interface including a method for changing data in the data source. An object that implements the EditableChartable interface can be attached to the DataSource property of ChartDataView. In this case, user changes to the data are sent back to the data source. This represents another level of connectivity between JClass Chart and the external data source. A good example can be found in the table demo (*JCLASS_HOME/jclass/chart/demos/table/*) in *Data.java*.

ChartDataModel

The ChartDataModel class is an abstract class that extends EditableChartable. In order to make use of ChartDataModel, developers must use it as a base class. The advantage of using ChartDataModel is that the ChartDataModel can use model-view to update JClass Chart. In other words, ChartDataView will be registered as an Observer of ChartDataModel. This allows the external data object to inform JClass Chart of changes to the data.

The model-view aspect of ChartDataModel is handled by the Java utility classes Observer and Observable. ChartDataView inherits from Observer. ChartDataModel inherits from Observable. If a ChartDataModel is attached to the DataSource property of a ChartDataView instance, it will register itself with the ChartDataModel. In the object derived from ChartDataModel (i.e. the external data source object), a call to update() will send a message to ChartDataView.

The details of the messages sent from ChartDataModel to ChartDataView are handled by the ChartDataModelUpdate class. An instance of ChartDataModelUpdate must be constructed by the external data source and passed to the update() method. Possible messages include CHANGE_VALUE, NEW_VALUE, CHANGE_ROW, NEW_ROW. For a complete list of messages, see the API documentation for ChartDataModelUpdate. A good example of an updating data source can be found in the strip chart demonstration program (*JCLASS_HOME/jclass/chart/demos/stripper/*) in *StripperData.java*.

8.10 Creating Custom Data Sources

On some occasions, you may find it necessary to create a custom data source. You have two basic options: you can create a data source from scratch, using the Chartable interface (see Implementing the Chartable Interface). Or, you can subclass/extend one of the existing implementations.

For example, `InputDataStreamSource` implements the `Chartable` interface for use with input stream data sources. `FileDataSource`, `AppletDataSource`, `URLDataSource` and `StringDataSource` all subclass from it.

`VectorDataSource` also implements the `Chartable` interface. It allows you to chart data from an in memory vector. This has many uses. For example, `JDBCDataSource` extends `VectorDataSource` to add the ability to chart a Result Set.

Whatever method you use, once you create a data source, use the same syntax, as you would any other data source to attach it to your chart:

```
JCChart chart = new JCChart();
chart.getDataView(0).setDataSource(new MyDataSource());
```

8.10.1 Implementing the Chartable Interface

A data source object you create must implement (at minimum) the `Chartable` interface. It must implement all the methods defined in `Chartable`, but the following four are the most important:

1. `getDataInterpretation()`

   This method returns either `Chartable.GENERAL` or `Chartable.ARRAY`.

   `ARRAY` means that the first row in the table of data contains the X-values used for every series, and all subsequent rows contain Y-values. `GENERAL` means the data is interpreted in pairs, the first row containing the X-values, and the second row containing the Y-values.

2. `getDataItem(int row, int column)`

   This method reads data from the data source into the chart.

3. `getRow(int row)`

   This method also reads data from the data source into the chart, but an entire row at a time.

4. `getNumRows()`

   This method returns the number of rows in the table of data.

The remaining methods in the `Chartable` interface can be useful, but aren't required to get data into the chart (return `null` in your data source):

`getPointLabels()`	Retrieves Point-labels for the data.
`getSeriesName()`	Retrieves a particular series name (used to retrieve a `ChartDataViewSeries` object by name).
`getSeriesLabel()`	Retrieves a particular Series-label.
`getName()`	Retrieves the name for the data.

Row Indices

`getDataItem()` and `getRow()` index data source entries by rwo number. The meaning of the number depends on the value returned by `getDataInterpretation()`. If ARRAY, row 0 represents the x values, and row 1-n (where n+number of series) are the y values. `getNumRows()` should return n+1. If GENERAL, every even row number contains X values, and every odd row number contains Y values. `getNumRows()` should return n*2.

Column Indices

The column number used in `getDataItem()` starts at 0 and goes to n-1, where n is the number of points in a series

Attaching your data source to the chart is no different than attaching a stock data source, for example:

```
JCChart chart = new JCChart();
chart.getDataView(0).setDataSource(new MyDataSource());
```

The following example creates a data source that returns data from a matrix:

```
package jclass.chart.demos.basic;

import jclass.chart.Chartable;
import java.util.Vector;

public class ArrayData implements Chartable {

// The data - first row is x, all the rest are y
public double rawData[][] = {
  { 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7 },
  { 0.3, 0.2, 0.16, 0.15, 0.145, 0.143, 0.142 },
  { -0.1, -0.2, 0.4, 0.3, 0.2, 0.1, 0.0},
  { 0.4, 0.3, 0.4, 0.3, 0.4, 0.3, 0.4 },
  { 0.2, 0.5, 0.2, 0.5, 0.2, 0.5, 0.2 },
  { 0.8, 0.7, 0.8, 0.7, 0.8, 0.7, 0.8 },
  { 0.2, 0.1, 0.2, 0.1, 0.2, 0.1, 0.2 },
  { 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3 }
};
```

```
// Holds the data
Vector data = new Vector();

// Default constructor.  Puts default rawData into data vector.
public ArrayData() {
    makeData(rawData);
}

// Typical constructor.  Puts supplied data into the data vector.
public ArrayData(double data[][]) {
    makeData(data);
}

// Create the data vector given an 2D array of doubles
private void makeData(double array[][]) {
    for (int i = 0; i < array.length; i++) {
        Vector row = new Vector();
        for (int j = 0; j < array[i].length; j++) {
            row.addElement(new Double(rawData[i][j]));
        }
        data.addElement(row);
    }
}

// Overridden from Chartable
public Object getDataItem(int row, int column) {
        Object rval = null;
        try {
               rval = ((Vector)data.elementAt(row)).elementAt(column);
        }
        catch (Exception e) {
        }
        return rval;
}
// Overridden from Chartable
public Vector getRow(int row) {
        Vector rval = null;
        try {
                rval = (Vector)data.elementAt(row);
        }
        catch (Exception e) {
        }
        return rval;
}

// Overridden from Chartable
public int getDataInterpretation() {
        return ARRAY;
}

// Overridden from Chartable
public int getNumRows() {
        if (data == null) return 0;
        return data.size();
}
```

```
// Overridden from Chartable
public String[] getPointLabels() {
        return null;
}

// Overridden from Chartable
public String getSeriesName(int row) {
        return null;
}

// Overridden from Chartable
public String getSeriesLabel(int row) {
        return getSeriesName(row);
}

// Overridden from Chartable
public String getName() {
        return new String("Array Data");
}

}
```

9

Text and Style Elements

This chapter describes the different formatting elements available within JClass Chart, and how they can be used. If you developing your chart application using one of the JClass Chart Beans, please refer to the Bean Reference chapter instead.

9.1 Header and Footer Titles

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

Title Text and Alignment

Use the `Text` title property to add, change, or remove text for a title. To enter multiple lines of text, press **RETURN** between each line.

Use the `Adjust` title property to specify whether to center, left-justify, or right-justify a multi-line title.

Title Positioning

Use the `Left` and `Top` location properties to customize the location of the title. When their respective `IsDefault` properties are used, the header is centered over the `ChartArea` and the footer is centered below the `ChartArea`. See Positioning Chart Elements later in this chapter for more information.

Title Border

Use the `BorderType` and `BorderWidth` border properties to customize the title's border. See Borders later in this chapter for more information.

Title Colors

Use the `Background` and `Foreground` properties to customize background and text colors of a title. See Colors later in this chapter for more information.

Title Font

Use the `Font` property to customize the font used for a title. See Fonts later in this chapter for more information.

9.2 Legends

A legend shows the visual attributes (or `ChartStyle`) used for each series in the chart, with text that labels the series. You can customize the series labels, positioning, border, colors and font used for the legend.

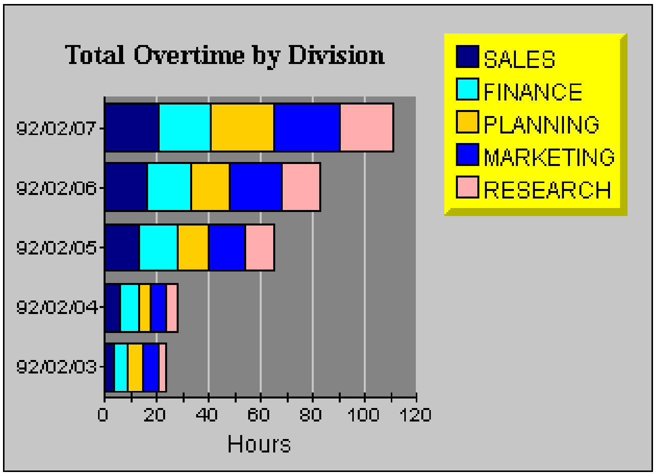


Figure 27 Vertically-oriented legend anchored NorthEast

Legend Text and Orientation

The legend displays the text contained in the `Label` property of each `Series` in a `DataView`. The `IsShowingInLegend` property of the series determines whether the `Series` will appear in the `Legend`.

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

Legend Positioning

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

Use the `Left` and `Top` location properties to fine-tune the positioning. When their respective `IsDefault` properties are used, the chart automatically positions the legend. See Positioning Chart Elements later in this chapter for more information.

Legend Border
Use the `BorderType` and `BorderWidth` border properties to customize the legend's border. See Borders later in this chapter for more information.

Legend Colors
Use the `Background` and `Foreground` properties to customize background and text colors of the legend. See Colors later in this chapter for more information.

Legend Font
Use the `Font` property to customize the font used for the legend. See Fonts later in this chapter for more information.

9.2.1 Customizing Legends

JClass Chart provides a legend toolkit for users who want more control over legend behavior. You can create custom legends by overriding the existing legends provided, or by subclassing from the `JCLegend` class. Create a custom legend by subclassing `JCLegend`, and overriding the abstract method `layoutLegend()`. If you just want to make minor changes to the legend, override `JCGridLegend` or `JCMultiColLegend`. With either method, call `setLegend()` on the `JCChart` instance with the new legend.

Custom legend demo

The legend demo, located in *JCLASS_HOME/jclass/chart/demos/legend/* has several examples of how to use the legend toolkit.

The upper left chart on the demo uses the `JCMultiColLegend` class to create a legend. The upper right chart creates a custom legend called `sepLegend` that overrides the `JCGridLegend` class and inserts a separator between the title and the body, adding both layout and drawing rules for it. The lower left chart creates a custom legend called `revLegend` that overrides the `JCGridLegend` class to reverse the order of items in the legend. The lower right implements a custom legend `flowLegend` by overriding the abstract class `JCLegend` to layout the legend items in a flow style.

9.3 Chart Labels

Chart labels allow you to add more information to your chart. There are static labels that display continuously and interactive labels that pop-up when a cursor moves over a data item. Labels can be attached to different parts of a chart: absolute coordinates, coordinates in the plotting area, or a specific data item. To see a wide range of label uses, browse the demos in the *JCLASS_HOME/jclass/chart/demos/labels/* directory.

9.3.1 Label Implementation

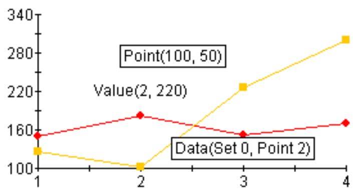
`JCChart` contains a list of labels, managed by the `ChartLabels` property. The list is initially empty. When you create a label, it is added to `ChartLabels`. Labels are instances of the `JCChartLabel` class.

9.3.2 Adding Labels to a Chart

Labels are added to a chart in two ways: with the `AutoLabels` property of `ChartDataView`, or by attaching an instance of `JCChartLabel` to a chart element.

Individual labels are attached in three ways: to coordinates on the chart area (`ATTACH_COORD`); coordinates on the plot area (`ATTACH_DATACOORD`); or to a data item (`ATTACH_DATAINDEX`). Interactive labels must use the `ATTACH_DATAINDEX` method.

Each label on the chart below uses a different attachment method. The "Point(100,50) label, is attached to coordinates originating from the top left corner of the chart area. "Value(2,220)" is attached to axes coordinates, and Data(Set0,Point2) is attached to a specific data item.



Attaching a Label to a Data Item

To attach a label to a point, bar or slice, set the `AttachMethod` property to `ATTACH_DATAINDEX`. The following example puts a label on a chart next to the fourth data point in the second data series.

```
cl = new JCChartLabel( Double.toString( series.getY( nPoint ) ),
false);
cl.setDataView( view );
cl.setDataIndex(new JCDataIndex(3, 1));
cl.setAttachMethod(JCChartLabel.ATTACH_DATAINDEX);
cl.setIsDwellLabel( false );
cl.setAnchor(JCChartLabel.AUTO);
chart.addChartLabel(cl);
```

Attaching a Label to Chart Area Coordinates

To attach a label to a point on the chart area, set the `AttachMethod` property to `ATTACH_COORD`. The coordinate origin for this method is the top left corner of the chart area. The following example sets up a point label and takes advantage of `JCString` tags.

```
JCChartLabel cl = new JCChartLabel(
"[COLOR=blue][FONT=TimesNewRoman-bold-20]Label 1", true );
cl.setAttachMethod( JCChartLabel.ATTACH_COORD );
cl.setCoord( new Point( 100, 75 ) );
chart.addChartLabel( cl );
```

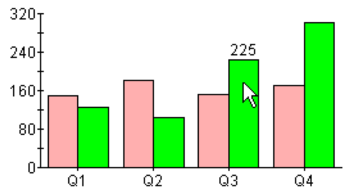
Attaching a Label to Plot Area Coordinates

To attach a label to coordinates on the plot area, set the `AttachMethod` property to `ATTACH_DATACOORD`. The plot area is defined by the chart's x and y axes. The following example places a label in the plot area at x-value 2.5, y-value 160.

```
JCChartLabel cl = new JCChartLabel("Attached to the data
coordinate", false);
cl.setDataCoord( new JCDataCoord( 2.5, 160 ) );
cl.setAnchor( JCChartLabel.NORTH );
cl.setAttachMethod( JCChartLabel.ATTACH_DATACOORD );
cl.setBorderType( Border.ETCHED_OUT );
cl.setBorderWidth( 5 );
chart.addChartLabel(cl);
```

9.3.3 Interactive Labels

You can have labels pop-up as a cursor dwells over a point, bar or slice (a **dwell label**). This allows you to create an interactive chart where information is hidden until the user wants to see it. The `AutoLabel` property will set up a compete series of **dwell labels** for your chart. In the example below, '225' appears on top of the green bar as the cursor passes over it, to indicate the value of the bar.



Automatically Generated Dwell Labels

The `AutoLabel` property of `ChartDataView` will generate a complete series of dwell labels if set to true. It attaches dwell labels to every data index. The following code, added to the `init()` method, adds automatic dwell labels to the data:

```
chart.getDataView(0).setAutoLabel(true);
```

Adding Individual Dwell Labels

Attaching an individual dwell label is the same as attaching a static label to a data item, except that the `IsDwellLabel` property is set to `true`:

```
cl.setIsDwellLabel( true );
```

A dwell label can only be used when the `AttachMethod` property is set to `ATTACH_DATAINDEX`.

9.3.4 Adding and Formatting Label Text

Adding Label Text

You can add text to a label with the constructor method, or with the `Text` property. To add text to a label when it is constructed, include the text in the constructor's argument, as follows:

```
JCChartLabel cl = new JCChartLabel("I'm a Label", false)'
```

To add text using the `Text` property, use the `setText` method, as follows:

```
cl.setText("I'm a Label");
```

Formatting Label Text

Use the `Font` property to set the font for a label:

```
Font f = new Font("timesroman", Font.BOLD, 24);
cl.setFont(f);
```

In addition to the properties defined in `JCChartLabel`, you can add tags for embedded colors, fonts, images, URL's and more. The following example uses `JCString` tags to set the font and color of text in a label:

```
JCChartLabel cl = new JCChartLabel(
"[COLOR=blue][FONT=TimesNewRoman-bold-20]Label 1", true );
```

Setting the second parameter to true indicates that the first argument is a `JCString`. See JCString Properties in Appendix B for complete details on `JCString` tags.

9.3.5 Positioning Labels

The `Anchor` property determines the position of the label, relative to the point of attachment. Valid positions include: `NORTH`, `NORTHEAST`, `NORTHWEST`, `EAST`, `WEST`, `SOUTHEAST`, `SOUTHWEST`, `SOUTH`. The following example shows the syntax:

```
cl.setAnchor(JCChartLabel.EAST);
```

9.3.6 Setting Label Borders and Colors

`JCChartLabel` has border and color properties that you can use to enhance the appearance of your chart. You can add graphical borders to your chart label, and set the border's width. The following example puts a plain border with a width of 1 on a label:

```
cl.setBorderType(Border.PLAIN);
cl.setBorderWidth(1);
```

For a complete description of the different border styles, see Borders.

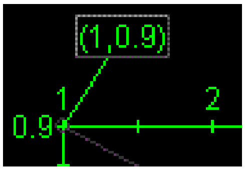
Use the Foreground and Background properties to set label colors. The syntax for setting color is as follows:

```
cl.setBackground(Color.white);
cl.setForeground(Color.black);
```

For a complete description of color control, see Colors.

9.3.7 Adding Connecting Lines

You can add lines that connect a label to its point of attachment. This can help the end-user pinpoint what a label refers to on a chart.



To add a connecting line to a label, set the IsConnected property to true, as follows:

```
cl.setIsConnected( true );
```

9.4 Chart Styles

Chart styles define all of the visual attributes of how a data series of data appears in the chart, including:

- Lines and points in plots and financial charts
- Color of each bar in bar charts
- Slice colors in pie charts
- Color of each filled area in area charts

Each series in a data view has its own JCChartStyle object; as new series are added, new JCChartStyle objects are created automatically by the chart. JClass Chart defines a set of visually different styles for *n* series, so while you can customize any chart style, you may not need to.

Every ChartStyle has a FillStyle, a LineStyle, and a SymbolStyle. FillStyles are used for bar, stacking bar, area and pie charts. LineStyles and SymbolStyles are used for plots.

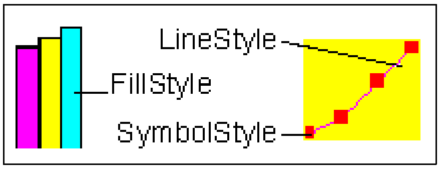


Figure 28 Types of ChartStyles available

`ChartStyle` is an indexed property of `ChartDataView` that “owns” the `JCChartStyle` objects for that data view. It can be manipulated like any other indexed property, for example:

```
        arr.setChartStyle(0, new JCChartStyle());
```

This adds the specified `ChartStyle` to the indexed property at the specified index. If the `ChartStyle` is null, the `JCChartStyle` at the specified point is removed. The following lists some of the other ways `ChartStyle` can be used:

- `getChartStyle(index)` – retrieves the chart style at the specified index
- `setChartStyle(JCChartStyle[])` – replaces all existing chart styles
- `JCChartStyle [] getChartStyle()` – retrieves a copy of the array of chart styles

Normally, you will not need to add or remove `JCChartStyle` objects from the collection yourself. If a `JCChartStyle` object already exists when its corresponding series is created, the previously created `JCChartStyle` object is used to display the data in this series.

Customizing Existing ChartStyles

Each `JCChartStyle` object contains three smaller objects that control different aspects of the style:`JCFillStyle`, `JCLineStyle`, and `JCSymbolStyle`.

The most common chart style sub-properties are repeated in `JCChartStyle`. For example, `FillColor` is a property of `JCChartStyle` that corresponds to the `Color` property of `JCFillStyle` object. The following lists all of the repeated properties:

- `LinePattern`, `LineWidth` and `LineColor` repeat `JCLineStyle` properties
- `SymbolShape`, `SymbolColor`, `SymbolSize`, and `SymbolCustomShape` repeat `JCSymbol` properties
- `FillColor`, `FillPattern` and `FillImage` repeat `JCFillStyle` properties.

FillStyle

`JCFillStyle` controls fills, used in bar, pie, area, and candle charts. Its properties include `Color` and `Pattern`. Use `Pattern` to set the fill drawing pattern and `Color` to set the fill color.

Note: Patterned fills are not currently supported within Java.

LineStyle

`JCLineStyle` controls line drawing, used in line and hi-lo charts. Its properties are `Color`, `Pattern` and `Width`. Use `Pattern` to set the line drawing pattern, `Color` to set the line color, and `Width` to set the line width.

Note: Line patterns are currently not supported within Java.

SymbolStyle

`JCSymbolStyle` controls the symbol used to represent points in a data series, used in plot or scatter plot charts. Its properties are `Shape`, `Color` and `Size`. Use `Shape` to set the symbol type, `Size` to set its size, and `Color` to set the symbol color.

The valid symbols are shown below:

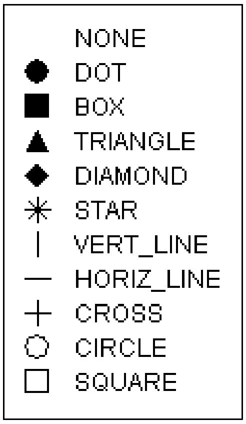


Figure 29 Symbols available in JCSymbolStyle

You can also provide a custom shape by implementing an abstract class `JCShape` and assigning it to the `CustomShape` property.

Customizing All ChartStyles

By looping through the `JCChartStyle` indexed property, you can quickly change the appearance of all of the bars, lines or points in a chart. For example, the following code lightens all of the bars in a chart whenever the mouse is clicked:

```
JCChartStyle[] style=c.getDataView(1).getChartStyle();
    for(int i=0; i < style.length, i++)
    {JCFillStyle fs=style[i].getFillStyle();
    fs.setColor(fg.getColor().brighten());}
```

9.5 Borders

One way to highlight important information or improve the chart's appearance is to use a border. You can customize the border of the following chart objects:

- Header and Footer titles
- Legend
- ChartArea
- each ChartLabel added to the chart
- the entire chart

BorderType and BorderWidth properties

Use the `BorderType` property to set the border style, and the `BorderWidth` property to set its thickness. The valid border types are shown below (defined in `jclass.base.Border`); you can also specify that no border is used.

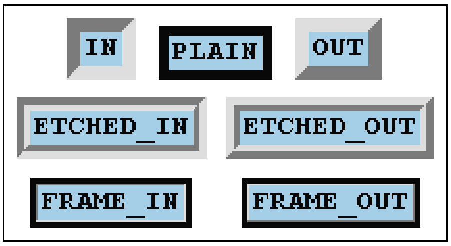


Figure 30 Border styles

9.6 Fonts

A chart can have more impact when you customize the fonts used for different chart elements. You may also want to change the font size to make an element better fit the overall size of the chart. Any font available when the chart is running can be used. You can set the font for the following chart elements:

- `Header` and `Footer` titles
- `Legend`
- Axis annotation and title
- each `ChartLabel` added to the chart.

Changing a Font

Use the font properties to set the font, style, and size attributes.

9.7 Colors

Color can powerfully enhance a chart's visual impact. You can customize chart colors using Java colornames or RGB values. Using an interactive tool like the Chart Customizer can make selecting custom colors quick and easy. Each of the following visual elements in the chart has a background and foreground color that you can customize:

- the entire chart
- `Header` and `Footer` titles
- `Legend`
- `ChartArea`
- `PlotArea` (foreground colors `JCChartArea`'s `AxisBoundingBox`)
- each `ChartLabel` added to the chart.

Other Chart objects have color properties too, including `ChartDataView` (bar/pie outline color), `ChartStyles`, `GridLines`, and `Markers`.

Color Defaults

If colors have not been set, each Chart element inherits its colors from the element in which it is contained. This means, for example, that setting the background color of `JCChart` also changes the background color of the chart area, header, footer, legend, and plot area. However, once the application sets the colors of an element, they do not change when other elements' colors change.

Specifying Foreground and Background Colors

Each chart element listed above has a `Background` and `Foreground` property that specifies the current color of the element. The easiest way to specify a color is to use the built-in colornames defined in `java.awt.Color`. The following table summarizes these colors:

Built-in Colors in java.awt.Color

black	blue	cyan
darkGray	gray	green
lightGray	magenta	orange
pink	red	white
	yellow	

Alternately, you can specify a color by its RGB components, useful for matching another RGB color. RGB color specifications are composed of a value from 0 – 255 for each of the red, green and blue components of a color. For example, the RGB specification of Cyan is "0-255-255" (combining the maximum value for both green and blue with no red). An extensive listing of available RGB color values can be found in Appendix C, "Colors and Fonts".

The following example sets the header background using a built-in color, and the footer background to an RGB color (a dark shade of Turquoise):

```
c.getHeader().setBackground(Color.cyan);

mycolor = new Color(95,158,160);
c.getFooter().setBackground(mycolor);
```

Take care not to choose a background color that is also used to display data in the chart. The default ChartStyles use all of the built-in colors in the following order: Red, Orange, Blue, Light gray, Magenta, Yellow, Gray, Green, Dark Gray, Cyan, Black, Pink and White.

9.8 JCStrings

There are often cases where it would be desirable to add such things as images, text and hyperlinks within a JClass Chart component, particularly if the component is to be used within an applet. Most JClass Chart components support a rich text format called "JCString", which allows a mixture of hypertext, images and text within Chart components. Text can also appear in a variety of colors, fonts and styles, including underline and strikeout.

For example, the following code adds a multi-line JCString to the header:

```
    c.getHeader().getLabel().setText("[IMAGE=yuri.gif][NEWLINE]
[FONT=TimesRoman-italic-20] Yuri Presents...", true);
```

For more information on the types of JCStrings available in JClass Chart, see Appendix A, "JCString Properties".

9.9 Positioning Chart Elements

Each of the main chart elements (`Header`, `Footer`, `Legend`, `ChartArea`, and `ChartLabels`) has properties that control its position and size. While the chart can automatically control these properties, you can also customize the following:

- Positioning of any element
- Size of any element.

When the chart controls positioning, it first allows space for the `Header`, `Footer`, and `Legend`, if they exist (size is determined by contents, border, and font). The `ChartArea` is sized and positioned to fit into the largest remaining rectangular area. Positioning adjusts when other chart properties change.

ChartLabels do not figure into the overall Chart layout. Instead, they are positioned above all other Chart elements.

Changing the View Location

Use the `Left` location property to specify the number of pixels from the edge of the chart to the left edge of the chart element. Use the `Top` location property to specify the number of pixels from the edge of the chart to the top of the chart element. `Left` and `Top` have `IsDefault` properties that allow the chart to automatically position the element.

Changing Width and Height

Use the `Width` and `Height` location properties to specify the width and height of the chart element. These properties have `IsDefault` properties that allow the chart to automatically size the chart element.

Note that `ChartLabels` do not have `Width` and `Height` properties. Instead, `ChartLabel` size is adjusted using `Right` and `Bottom`.

9.10 3D Effect

Data in bar, stacking bar and pie charts can be displayed with a three-dimensional appearance using several `JCChartArea` properties:

- `Depth` – Specifies the apparent depth as a percentage of the chart's width. No 3D effect appears unless this property is set greater than zero.
- `Elevation` – Specifies the eye's position above the horizontal axis, in degrees.
- `Rotation` – Specifies the number of degrees the eye is positioned to the right of the vertical axis. This property has no effect on pie charts.

You can set the visual depth and the "elevation angle" of the 3D effect. You can also set the "rotation angle" on bar and stacking bar charts. `Depth`, `Rotation` and `Elevation` are all properties of the `ChartArea`.

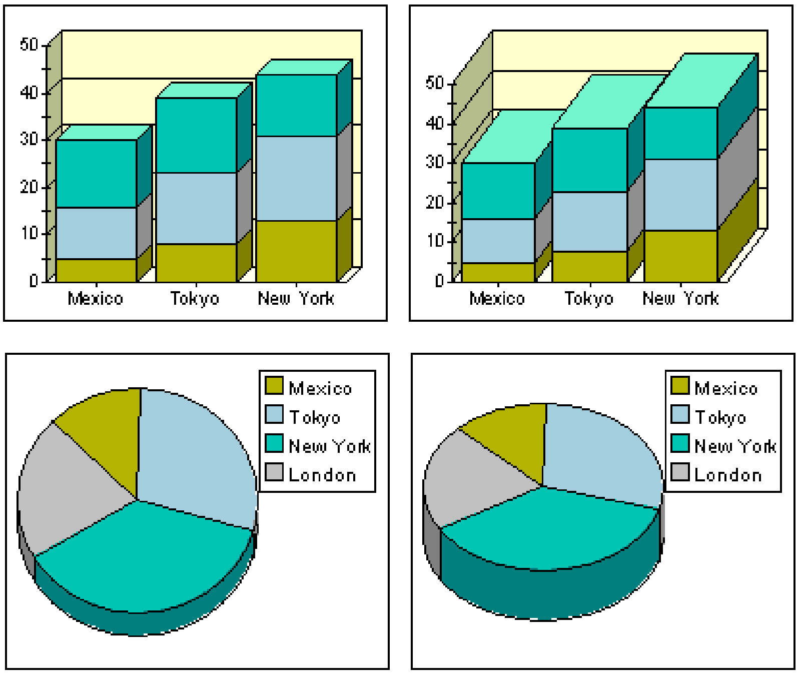


Figure 31 Two JClass Charts illustrating the effects of 3D

9.11 Special Bar Chart Properties

Bar charts display each point as one bar in a *cluster*. There are several properties defined in `JCBarChartFormat` that control exactly how the bars are spaced and displayed. Use the `getBarChartFormat()` method to retrieve and set these properties.

Stacking Bar Charts

Data is displayed as a stacking bar chart when the `ChartType` property is set to `JCChart.STACKING_BAR`. In stacking bar charts, there is only one bar per cluster. All Y-values less than zero are stacked below the Y-axis. The syntax is as follows:

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

Cluster Overlap

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

```
dataView.getBarChartFormat().setClusterOverlap(50);
```

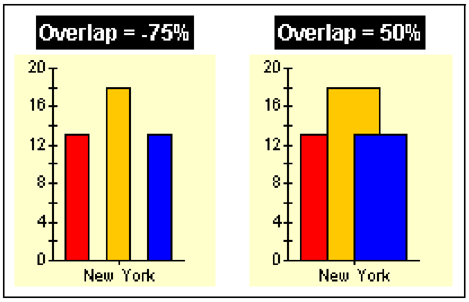


Figure 32 Negative and positive bar cluster overlap

Cluster Width

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

```
dataView.getBarChartFormat().setClusterWidth(100);
```

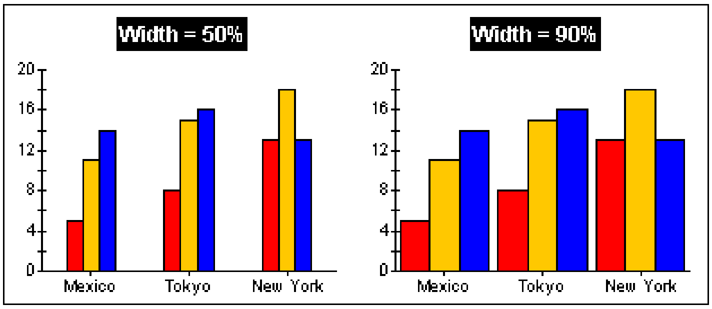


Figure 33 Setting different bar cluster widths

100-Percent Stacking Bar Charts

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

```
dataView.getBarChartFormat().set100Percent(true);
```

9.12 Special Pie Chart Properties

Pie charts are quite different from the other chart types. They do not have the concept of a two-dimensional grid or axes. They also introduce a special category called *"Other"*, into which all data values below a certain threshold are grouped.

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

```
JCPieChartFormat pcf = arr.getPieChartFormat();
pcf.setOtherLabel("[FONT=Helvetica-italic-
14]Other[NEWLINE]Bands");
pcf.setThresholdValue(10.0);
pcf.setThresholdMethod(JCPieChartFormat.PIE_PERCENTILE);
pcf.setSortOrder(JCPieChartFormat.DATA_ORDER);
```

9.12.1 Building the "Other" Slice

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

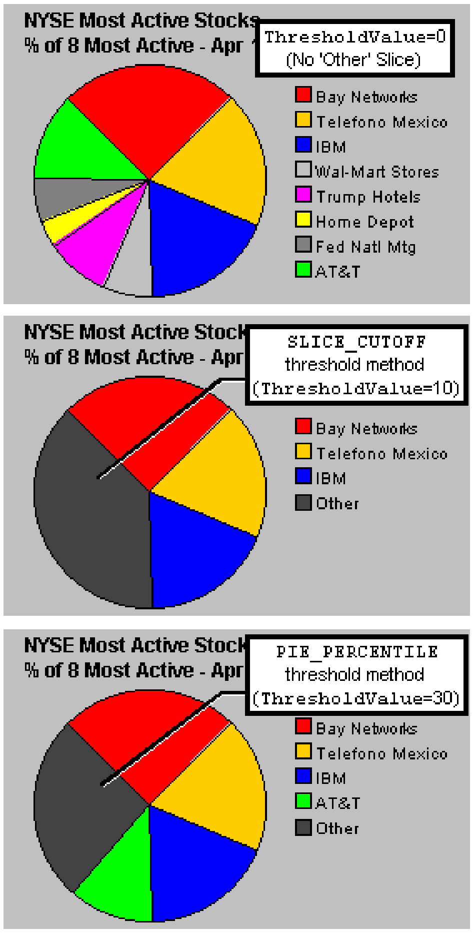


Figure 34   Three JClass Charts illustrating how the “Other” slice can be used

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

9.12.2 “Other” Slice Style and Label

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

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

9.12.3 Pie Ordering

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

9.12.4 Exploded Pie Slices

It is possible to have individual slices of a pie "explode" (i.e. detach itself from the rest of the pie) when a user clicks on the slice. The slice "implodes" (i.e. re-joins the rest of the pie) if the user clicks on it again.

Two properties of JCPieChartFormat are responsible for this function: ExplodeList and ExplodeOffset.

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

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

The following code sample shows how ExplodeList and ExplodeOffset can be used. First, the list of exploded pie slices is set:

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

Given a pick event, the picked pie slice explodes or implodes, depending on whether or not it is already exploded.

```
public void pick(JCPickEvent e)
{
JCDataIndex di = e.getPickResult();
if (di == null) return;
Object obj = di.getObject();
ChartDataView vw = di.getDataView();
ChartDataViewSeries srs = di.getSeries();
int slice = di.getSeriesIndex();
int pt = di.getPoint();
int dist = di.getDistance();
if (vw != null && slice != -1) {
  JCPieChartFormat pcf = vw.getPieChartFormat();
  Point[] exList = pcf.getExplodeList();
  if (exList == null) return;
  // implode existing exploded slices
  for (int i = 0; i < exList.length; i++) {
      if ((exList[i].x == pt) && (exList[i].y == slice)) {
          Point[] newList = new Point[exList.length - 1];
          for (int j = 0; j < i; j++)
              newList[j] = exList[j];
          for (int j = i; j < newList.length; j++)
```

```
                newList[j] = exList[j + 1];
            pcf.setExplodeList(newList);
            vw.setChanged(true);
            return;
        }
    }
    // explode new slice
    Point[] newList = new Point[exList.length + 1];
    for (int j = 0; j < exList.length; j++)
        newList[j] = exList[j];
    newList[exList.length] = new Point(pt, slice);
    pcf.setExplodeList(newList);
    vw.setChanged(true);
  }
  }
```

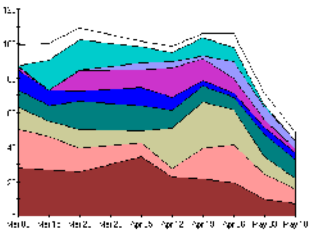
The full code for this program can be found in the *exploding pie charts demo*. For more information on `pick`, see "Using Pick and Unpick".

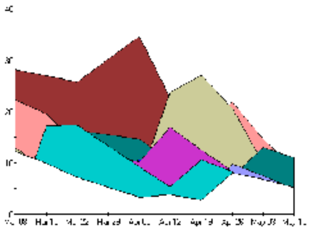
9.13 Special Area Chart Properties

Similar to the stacking bar type, JClass Chart provides a stacking area type. To see an example of a stacking area chart, launch "Area Chart Demo" from *JCLASS_HOME/jclass/chart/demos/index.html* .

Stacking Area Charts

A stacking area chart places each Y-series on top of the last. This shows the area relationships between each series and the total. The following example shows the same set of data as displayed by stacking area and area types:


Stacking Area Chart

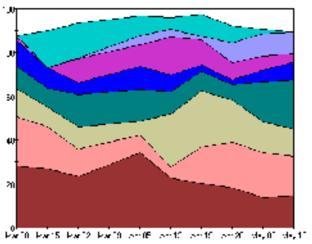

Area Chart

To create a stacking area chart, set the `ChartType` property to `JCChart.STACKING_AREA`., as follows:

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

100-Percent Stacking Area Charts

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



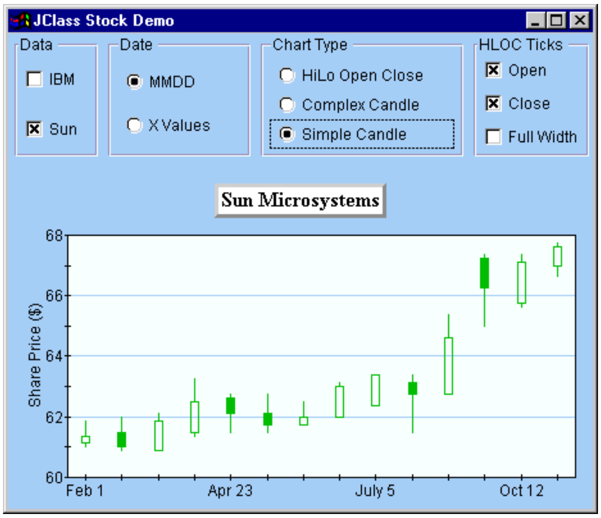
Use the following syntax to display data in 100-Percent mode:

```
dataView.getAreaChartFormat().set100Percent(true);
```

9.14 Hi-Lo and Candle Charts

JClass Chart's Hi-Lo, Hi-Lo-Open-Close, and Candle financial chart types use the Y-values in multiple series to construct each "bar". Hi-Lo charts use every *two* series and Hi-Lo-Open-Close and candle charts use every *four* series. Each series defines a specific portion of the bar:

- First series – High value
- Second series – Low value
- Third series (if needed) – Open value
- Fourth series (if needed) – Close value



Data: [ ] IBM [✓ Sun]; Date: [✓ MMDD] ( ) X Values; Chart Type: ( ) HiLo Open Close ( ) Complex Candle [✓ Simple Candle]; HLOC Ticks: [✓ Open] [✓ Close] [ ] Full Width

Figure 35 Simple Candle chart displayed by stock demo

It is useful to think of each group of series as one "logical series". But note that most JClass Chart properties or methods that use a series (such as chart labels attached by DataIndex) use the *actual* series index.

Hi-Lo-Open-Close Charts

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

`IsShowingOpen`	Displays or hides open tick marks
`IsShowingClose`	Displays or hides close tick marks
`IsOpenCloseFullWidth`	Displays open/close ticks across both sides of the bar. This is useful for creating error bar charts.

Customizing ChartStyles

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

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

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

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

Simple and Complex Candle Charts

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

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

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

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

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

Example Code

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

```
JCChartStyle chartStyle;
JCCandleChartFormat candleFormat;

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

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

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

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

10

Advanced Chart Programming

Controlling the chart in an application program is generally straightforward once you are familiar with the programming basics and the object hierarchy. For most JClass Chart objects, all the information needed to program them can be found in the Application Programming Interface (API). In addition, extensive information on how they can be used can be found in the numerous example and demonstration programs provided with JClass Chart.

This chapter covers some basic programming concepts for JClass Chart and also looks at more complex chart programming tasks.

10.1 Batching Chart Updates

Normally, the chart is repainted immediately after a property is set. To make several changes to a chart before causing a repaint, set the `IsBatched` property of the `JCChart` object to `true`. Property changes do not cause a repaint until `IsBatched` is reset to `false`.

The `IsBatched` property is also defined for the `ChartDataView` object. This `IsBatched` property is independent of `JCChart.IsBatched`. It is used to control the update requests sent from the `DataSource` to the chart.

It is recommended that you batch around the creation or updating of multiple chart labels.

10.2 Coordinate Conversion Methods

The `ChartDataView` object provides methods which enable you to do the following:

- Convert from data coordinates to pixel coordinates and vice versa.
- Determine the pixel coordinates of a given data point in a series, or the closest point to a given set of pixel coordinates.

These functions can be accomplished by using `coordToDataCoord()` and `dataIndexToCoord()` respectively, or by using their functional equivalents `map` and `unmap`.

10.2.1 CoordToDataCoord and DataIndexToCoord

To convert from data coordinates to pixel coordinates, call the `dataCoordToCoord()` method. For example, the following code obtains the pixel coordinates corresponding to the data coordinates (5.1, 10.2):

```
Pointp=c.getDataView(0).dataCoordToCoord(5.1,10.2);
```

This works in the same way as `unmap`. Note that the pixel coordinate positioning is relative to the upper left corner of the `JCChart` component display.

To convert from pixel coordinates to data coordinates, call `coordToDataCoord()`. For example, the following converts the pixel coordinates (225, 92) to their equivalent data coordinates:

```
JCDataCoord cd=c.getDataView(0).coordToDataCoord(225,92);
```

This works in the same manner as `map`. So, `coordToDataCoord()` returns a `JCDataCoord` object containing the *x* and *y* values in the data space.

To determine the pixel coordinates of a given data point, call `dataIndexToCoord()`. For example, the following code obtains the pixel coordinates of the third point in the first data series:

```
JCDataIndex di=new JCDataIndex(3,c.getDataView(0).getSeries(0));
Paint cdc=c.getDataView(0).dataIndexToCoord(di);
```

To determine the closest data point to a set of pixel coordinates, call `coordToDataIndex()`:

```
JCDataIndex di=c.getDataView(0).coordToDataIndex(225,92,
                                    ChartDataView.PICK_FOCUSXY);
```

Essentially, these last two examples demonstrate that `dataIndexToCoord()` works in much the same way as `pick` and `unpick`. The third argument passed to `coordToDataIndex()` specifies how the nearest series and point value are determined. This argument can be one of `ChartDataView.PICK_FOCUSXY`, `ChartDataView.PICK_FOCUSX` or `ChartDataView.PICK_FOCUSY`. For more information on the `pick` and `unpick` methods, see the Using Pick and Unpick section later in this chapter.

`JCDataIndex` contains the series and point value corresponding to the closest data point, and also returns the distance in pixels between the pixel coordinates and the point. `coordToDataIndex()` returns a `JCDataIndex` instance.

10.2.2 Map and Unmap

The `map` and `unmap` are functionally equivalent to the `coordToDataCoord()` and `dataIndexToCoord()` methods. They are provided as convenience methods, and are more in keeping with typical Java terminology than `coordToDataCoord()` and `dataIndexToCoord()`, which are used as method names by KL Group's Olectra Chart product for Windows developers.

10.3 Double Buffering

Double buffering is a graphics technique which will reduce the amount of flashing perceived by a user when a chart changes. By default, double buffering is in effect. To turn off double buffering for a `JCChart` object, set its `DoubleBuffered` property to `false`.

When double buffering is turned on, every time the chart changes, it will:

- Allocate (if necessary) and clear an off-screen bitmap.
- Render the complete graph to the off-screen bitmap.

When double buffering is turned off, the chart clears the screen image every time a chart is changed (possibly causing a visual flash). Then, it renders the complete image to the visible window (possibly allowing the user to see the chart being drawn piece by piece).

Turning off double buffering can improve the chart's graphing performance and reduce its memory requirements, but it will likely result in a chart display that "flashes" repeatedly as updates are continually drawn to the screen. By default `IsDoubleBuffered` is `true`.

10.4 FastAction

The `FastAction` property determines whether chart actions will use an optimized mode in which it does not bother to update display axis annotations or grid lines during a chart action. Default value is `false`.

Using `FastAction` can greatly improve the performance of a chart display, because relatively more time is needed to draw such things as axis annotations or grid lines than for simply updating the points on a chart. It is designed for use in dynamic chart displays, such as charts that enable the user to perform translation or rotation actions.

The following line of code shows how `FastAction` can be used in a program:

```
    c.getChartArea().setFastAction(true);
```

10.5 Programming End-User Interaction

An end-user can interact with a chart more directly than using the Customizer. Using the mouse and keyboard, a user can examine data more closely or visually isolate part of the chart. JClass Chart provides the following interactions:

- moving the chart
- zooming into or out of the chart
- rotation (only for bar or pie charts displaying a 3D effect)
- adding depth cues to the chart
- interactively change data points (using the pick feature)

It is also possible in most cases for the user to reset the chart to its original display parameters. The interactions described here effect the chart displayed inside the `ChartArea`; other chart elements like the header are not affected.

Note: The keyboard/mouse combinations that perform the different interactions can be changed or removed by a programmer. The interactions described here may not be enabled for your chart.

A chart action is a user event that causes some interactive action to take place in the control. In JClass Chart, actions like zoom, translate and rotate can be mapped to a mouse button and a modifier. For example, it is possible to bind the translate event to the combination of mouse button 2 and the **Control** key. Whenever the user hits **Control** and mouse button 2 and drags the mouse, the chart will move.

10.5.1 Event Triggers

An event trigger is a mapping of a mouse operation and/or a key press to a chart action. In the example above, the trigger for translate is a combination of mouse button 2 and the **Control** key.

An event trigger has two parts:

- the modifier, which specifies the combination of meta keys and mouse buttons that will trigger the action; and,
- the action, which specifies the combination of chart action that will occur.

Valid actions include `EventTrigger.ZOOM`, `EventTrigger.TRANSLATE`, `EventTrigger.ROTATE` and `EventTrigger.DEPTH`.

10.5.2 Valid Modifiers

The value of a modifier is specified using AWT modifiers, as shown in the following list:

- `Event.SHIFT_MASK`
- `Event.CTRL_MASK`
- `Event.ALT_MASK`

- `Event.META_MASK`

If you are using the JDK 1.1, you can also specify the mouse button using one of the following modifiers:

- `InputEvent.BUTTON1_MASK`
- `InputEvent.BUTTON2_MASK`
- `InputEvent.BUTTON3_MASK`

10.5.3 Programming Event Triggers

To program an event trigger, use the `setTrigger` method to add the new action mapping to the collection.

For example, the following tells JClass Chart to perform a zoom operation when **Shift** and mouse button are pressed:

```
c.setTrigger(0,newEventTrigger(Event.SHIFT_MASK,
              EventTrigger.ZOOM);
```

10.5.4 Removing Action Mappings

To remove an existing action mapping, set the trigger to null, as in the following example:

```
c.setTrigger(0,null);
```

10.5.5 Calling an Action Directly

In JClass Chart, it is possible to force some actions by calling a method of `JCChart`. The following is a list of the methods that can be called upon to force a particular action:

- Translation - `translateStart()`, `translate()`, `translateEnd()`
- Rotation - `rotateStart()`, `rotate()`, `rotateEnd()`
- Zoom - `zoomStart()`, `zoom()`, `zoomEnd()`
- Scale - `scale()`
- Reset - `reset()`

10.5.6 Specifying Action Axes

Actions like translation occur with respect to one or more axes. In JClass Chart, the axes can be set using the `HorizActionAxis` and `VertActionAxis` properties of `JCChartArea`, as the following code fragment illustrates:

```
ChartDataView arr = c.getDataView(0);
c.getChartArea().setHorizActionAxis(arr.getXAxis());
c.getChartArea().setVertActionAxis(arr.getYAxis());
```

Note that it is possible to have a null value for an action axis. This means that chart actions like translation do not have any effect in that direction. By default, the

HorizActionAxis is set to the default X-axis, and the VertActionAxis is set to the default Y-axis.

10.6 Image-Filled Bar Charts

It is possible to use image files as chart elements within a bar chart. This is accomplished by using the Image method. Image sets the image used to paint the fill region of bar charts. It takes img as a parameter, which is an AWT Image class representing the image to be used to paint image fills. If set to null, no image fill is done.

The following code fragment shows how Image can be incorporated into a program:

```
// Set the labels of each series.
String seriesLabels[] = {"CD", "Cassette"};
String imageStrings[] = {"cd.gif", "tape.gif"};
ChartDataViewSeries seriesList[] = arr.getSeries();
for (int i = 0; i < arr.getNumSeries(); i++) {
    if (i < seriesLabels.length) {
        seriesList[i].setLabel(seriesLabels[i]);
        if (imageStrings[i] != null) {
            seriesList[i].getStyle().getFillStyle().setImage(
                this, imageStrings[i]);
        }
    }
}
```

The effects can be seen in the ImageBar demonstration program, (in the *JCLASS_HOME/jclass/chart/demos/imagebar/* directory), which comes with JClass Chart.

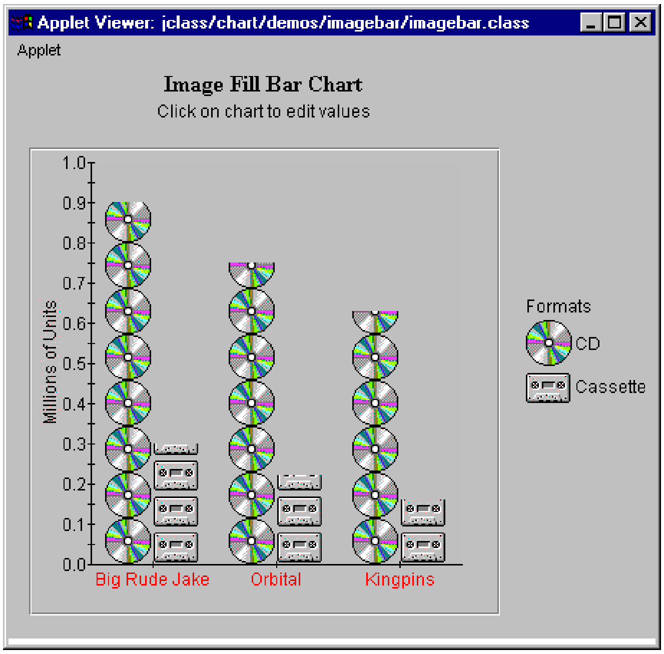


Figure 36 The imagebar demonstration program

The image is clipped at the point of the highest value indicated for the bar chart.

`Image` only tiles the image along a single axis. For example, if the bars were widened in the above illustration, it would still tile along the vertical Y-axis only, and would not fill in the image across the horizontal X-axis. This same principle applies (though along different axes) when the bar chart is rotated 90 degrees.

Note: `Image` can only be used with the image formats that can be used in Java.

10.7 Using Pick and Unpick

The `pick` method is used to retrieve an *x,y* coordinate in a Chart from user input and then translate that into selecting the data point nearest to it. For example, if a user clicks within a single bar within a bar chart, `pick` takes the coordinates of the mouse-click and selects that bar for any action within the program. Similarly, if a user clicks in an area immediately above a bar chart, `pick` is used to select the bar that is closest to the mouse click.

To use the pick listener, you must first set up a PICK event trigger on the chart. See, Programming Event Triggers for more details.

Consider the following code listing (the code that comprises the DrillDown demonstration program that comes with JClass Chart, in

JCLASS_HOME/jclass/chart/demo/drilldown/) that demonstrates how pick can be used to “drill down” to reveal more information:

```
package jclass.chart.demos.drilldown;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Event;
import java.awt.GridLayout;
import jclass.bwt.BWTEnum;
import jclass.chart.ChartDataView;
import jclass.chart.ChartDataViewSeries;
import jclass.chart.EventTrigger;
import jclass.chart.JCAxis;
import jclass.chart.JCPickListener;
import jclass.chart.JCPickEvent;
import jclass.chart.JCChartStyle;
import jclass.chart.JCChart;
import jclass.chart.ChartText;
import jclass.chart.JCDataIndex;
import jclass.chart.JCLegend;
import jclass.chart.JCChartArea;
import jclass.chart.demos.DemoFrame;

/*
 * This applet shows using pick to drill down to more refined data
 */
public class DrillDown extends Applet implements JCPickListener
{
Data d = null;

JCChart c = null;

public void init()
{
    setLayout(new BorderLayout(10,10));

    d = new Data();

    Color Turquoise = new Color(64,224,208);
    Color DarkTurquoise = new Color(0x00,0xce,0xd1);

    c = new JCChart();
    c.setTrigger(0, new EventTrigger(0, EventTrigger.PICK));
    c.setBackground(DarkTurquoise);

    c.getChartArea().getPlotArea().setBackground(Turquoise);
    c.getChartArea().setBorderWidth(4);
    c.getChartArea().setBorderType(BWTEnum.SHADOW_ETCHED_IN);

    c.getHeader().setBackground(Turquoise);
    c.getHeader().getLabel().setText("[FONT=TimesRoman-bold-16]Drill
Down Demo\n[FONT=TimesRoman-plain-16]Independent Comic Book Sales
1996", true);
    c.getHeader().setBorderType(BWTEnum.SHADOW_ETCHED_OUT);
    c.getHeader().setBorderWidth(1);
    c.getHeader().setIsShowing(true);

    c.getLegend().setIsShowing(true);
    c.getLegend().setBackground(Turquoise);
    c.getLegend().setBorderType(BWTEnum.SHADOW_IN);
    c.getLegend().setBorderWidth(1);

    c.setIsBatched(false);
    c.getDataView(0).setDataSource(d);
    c.getDataView(0).setChartType(JCChart.BAR);
    c.getDataView(0).setHoleValue(-1000);

    c.getFooter().setIsShowing(true);
    c.getFooter().getLabel().setText("[FONT=TimesRoman-italic-12]Drill
Down -> Mouse Down on Bar or Legend\nDrill Up -> Mouse Down on Other
Area of Graph", true);
    c.getFooter().setAdjust(ChartText.LEFT);

    c.getChartArea().setDepth(10);
    c.getChartArea().setElevation(20);
    c.getChartArea().setRotation(20);

    // Set colors for each data series
    setSeriesColor();

    // Set up pick and rotate trigger
    c.setTrigger(0, new EventTrigger(0, EventTrigger.PICK));
    c.setTrigger(0, new EventTrigger(Event.SHIFT_MASK,
                                     EventTrigger.ROTATE));

    // Add listener for pick events
    c.addPickListener(this);

    add("Center",c);
}

void setSeriesColor()
{
    // Set colors for each data series
   Color colors[] = {Color.red, Color.blue, Color.white, Color.magenta,
                     Color.green, Color.cyan, Color.orange,
                     Color.yellow};
    ChartDataView arr = c.getDataView(0);
    ChartDataViewSeries seriesList[] = arr.getSeries();
    for (int i = 0; i < arr.getNumSeries(); i++) {
            seriesList[i].getStyle().setFillColor(colors[i]);
    }
}

/** Pick event listener.  Upon receipt of a pick event, it either drills
 *  up or down to more general or refined data.
 */
public void pick(JCPickEvent e)
{
    boolean doLevel = false;
    boolean doUpLevel = true;
    JCDataIndex di = e.getPickResult();
    int srs = 0;

    // If clicked on bar or legend item, drill down.  If clicked on
    // any other area of chart, drill up.
    if (di != null) {
        Object obj = di.getObject();
        ChartDataView vw = di.getDataView();
        srs = di.getSeriesIndex();
        int pt = di.getPoint();
        int dist = di.getDistance();

        if (vw != null && srs != -1) {
            if (srs >= 0) {
                if ((obj instanceof JCLegend) ||
                    (obj instanceof JCChartArea && dist == 0)) {
                    doLevel = true;
                    doUpLevel = false;
                }
                else {
                    doLevel = true;
                }
            }
        } else {
            doLevel = true;
        }
    } else {
        doLevel = true;
    }

    if (doLevel) {
        c.setIsBatched(true);
        if (doUpLevel)
            d.upLevel();
        else
            d.downLevel(srs);
        setSeriesColor();
        c.setIsBatched(false);
    }
}

public static void main(String args[]) {
    DemoFrame f = new DemoFrame("Basic Drilldown example");
    DrillDown tc = new DrillDown();
    tc.init();
    f.setLayout(new GridLayout(1,1));
    f.add(tc);
    f.pack();
    f.show();
    f.resize(500,400);
}

}
```

When compiled and run, the *DrillDown.class* program displays the following:

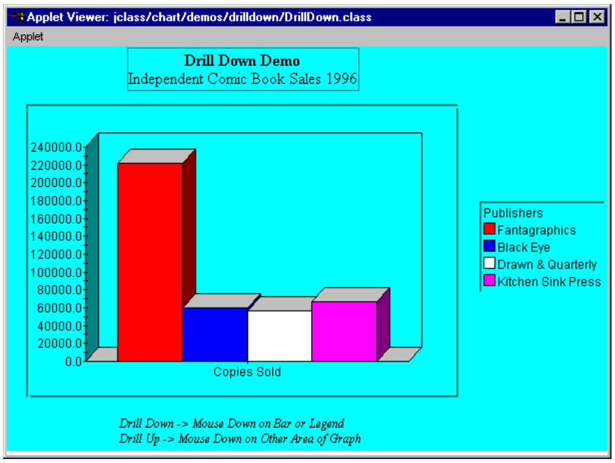


Figure 37   The DrillDown demonstration program displayed

When a bar or legend within this chart is clicked by the user, the program “drills down” to reveal more refined data comprising that bar. If an area outside of the bars is clicked upon, then the program “drills up” to reveal more general data.

`pick` is key to this program, determining the way the program interacts with the user. `pick` requires an event trigger and listener to work, as the following code fragment shows:

```
    c.setTrigger(0, new EventTrigger(0, EventTrigger.PICK));
    c.setTrigger(0, new EventTrigger(Event.SHIFT_MASK,
                                     EventTrigger.ROTATE));
    c.addPickListener(this);
    add("Center",c);
}
public void pick(JCPickEvent e)
{
    JCDataIndex di = e.getPickResult();
```

When a user clicks in the DrillDown demonstration program, the event is triggered, and the *x,y* coordinates are passed along to the `pick` event listener, which in turn takes the information and performs the indicated action. The `pick()` method returns a `JCDataIndex` which encapsulates the point index and data series of the selected point.

10.7.1 Pick Focus

`pick` normally takes an *x,y* coordinate value, but it can take an *x* or *y* value only, which is useful for specific chart types. This can be specified using the `PickFocus` property, which specifies how distance is determined for pick operations. When set to `PICK_FOCUS_XY` (default), a pick operation will use the actual distance between the point and the drawn data. When set to values of `PICK_FOCUS_X` or `PICK_FOCUS_Y`, only the distance along the X- or Y-axis is used.

This is a particularly useful method to use within programs that display typical bar charts. In most cases it is more desirable to know which bar the user is *over* than which bar the user is *closest to* when the user clicks their mouse over a chart.

For example, a user may click over a relatively small bar in a bar chart, with the intention of raising the value of the bar displayed. If an adjacent bar in the chart is closer to the area of the mouse click along the Y-axis than the X-axis, then the adjacent bar could be selected instead of the intended target bar.

To overcome this, use `PickFocus` and select the axis whose values are to be reported back to the program. For example, the following line of code sets `PickFocus` to only report the x coordinate of a pick event:

```
arr.setPickFocus(ChartDataView.PICK_FOCUS_X);
```

10.7.2 Unpick

The `unpick` method essentially functions in the opposite manner of `pick`: given a data series and a data point within that series, `unpick` returns the pixel co-ordinates of that point relative to the chart area. It takes two sets of parameters: `pt` for the point index, and `series` for the data series. For bar charts it returns the top-middle location for a given bar, and the middle of an arc for a pie chart. `unpick` can be used to display information at a given point in a chart, and can be used for attaching labels to chart regions.

Part II

Reference Appendices

A

JClass Chart Property Listing

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

A.1 ChartDataView

Name	Description
`AutoLabel`	The `AutoLabel` property determines if the chart automatically generates labels for each point in each series. The default is `false`. The labels are stored in the `AutoLabelList` property. They are created using the `Label` property of each series.
`AutoLabelList`	The `AutoLabelList` property is a two-dimensional array of automatically-generated JCChartLabel instances, one for every point and series. The inner array is indexed by point; the outer array by series. Default is empty.
`BarChartFormat`	The `BarChartFormat` property represents the JCBarChartFormat for the ChartDataView instance. Unless the `ChartType` property is a bar chart, the `BarChartFormat` property is `null`.

Name	Description
BufferPlotData	The BufferPlotData property controls whether plot data is to be buffered to speed up the drawing process. This property is applicable for Plot, Scatter, Area, Hilo, HLOC, and Candle chart types only. Normally it is true. The property is ignored if the FastUpdate property is true. Plot data will be buffered for FastUpdate.
CandleChartFormat	The CandleChartFormat property represents the JCCandleChartFormat for the ChartDataView instance. Unless the ChartType property is a candle chart, the CandleChartFormat property is null.
Changed	The Changed property manages whether the data view requires recalculation. If set to true, a recalculation may be triggered. Default value is true.
ChartStyle	The ChartStyle property contains all the ChartStyles for the data series in this data view. Default value is generated.
ChartType	The ChartType property of the ChartData object specifies the type of chart used to plot the data. Valid values are: JCChart.AREA, JCChart.BAR, JCChart.CANDLE, JCChart.HILO, JCChart.HILO_OPEN_CLOSE, JCChart.PIE, JCChart.PLOT (default), JCChart.SCATTER_PLOT, and JCChart.STACKING_BAR.
DataSource	The DataSource property, if non-null, is used as a source for data in the ChartDataView. The DataSource can refer to an object that implements Chartable or EditableChartable, or it can refer to an object that extends the abstract class ChartDataModel. JCChart will do the "right thing" based on the object provided.
DrawingOrder	The DrawingOrder property determines the drawing order of items. When the DrawingOrder property is changed, the order properties of all ChartDataView instances managed by a single JCChart object are normalized.
DrawFrontPlane	The DrawFrontPlane property determines whether a data view that has both axes on the front plane of a 3d chart will draw on the front or back plane of that chart. If true, it will draw on the front plane; if false it will draw on the back plane. If either axis associated with the data view is on the back plane, this property will be ignored and the data view will automatically be drawn on the back plane. This property is also ignored for 3d chart types such as bar and stacking bars that automatically appear on the front plane.
FastUpdate	The FastUpdate property controls whether column appends to the data are performed quickly, only recalculating and redrawing the newly-appended data.
HiloChartFormat	The HiloChartFormat property represents the JCHiloChartFormat for the ChartDataView instance. Unless the ChartType property is a HiLo chart, the HiloChartFormat property is null.
HLOCChartFormat	The HLOCChartFormat property represents the JCHLOCChartFormat for the ChartDataView instance. Unless the ChartType property is a HiLoOpenClose chart, the HLOCChartFormat property is null.

Name	Description
HoleValue	The HoleValue property is a floating point number used to represent a hole in the data. Internally, ChartDataView places this value in the x and y arrays to represent a missing data value.
IsBatched	The IsBatched property controls whether the ChartDataView is notified immediately of data source changes, or if the changes are accumulated and sent at a later date.
IsInverted	If the IsInverted property is set to true, the x axis becomes vertical, and the y axis becomes horizontal. Default value is false.
IsShowing	The IsShowing property determines whether the dataview is showing or not. Default value is true.
IsShowingInLegend	The IsShowingInLegend property determines whether or not the view name and its series will appear in the chart legend.
Name	The Name property is used as an index for referencing particular ChartDataView objects.
NumPointLabels	The NumPointLabels property determines the number of labels in the PointLabels property. The PointLabels property is an indexed property consisting of a series of strings representing the desired label for a particular data point.
NumSeries	The NumSeries property determines how many data series there are in a ChartDataView.
OutlineColor	The OutlineColor property determines the color with which to draw the outline around a filled chart item (e.g. bar, pie).
PickFocus	The PickFocus property specifies how distance is determined for pick operations. When set to PICK_FOCUS_XY, a pick operation will use the actual distance between the point and the drawn data. When set to values of PICK_FOCUS_X or PICK_FOCUS_Y, the distance only along the X or Y axis is used.
PieChartFormat	The PieChartFormat property represents the JCPieChartFormat for the ChartDataView instance. Unless the ChartType property is a pie chart, the PieChartFormat property is null.
Pointlabel	Sets a particular PointLabel from the PointLabels property (see below).
PointLabels	The PointLabels property is an indexed property consisting of a series of strings representing the desired label for a particular data point.
Series	The Series property is an indexed property that contains all data series for a particular ChartDataView. The order of ChartDataViewSeries objects in the series array corresponds to the drawing order.
XAxis	The XAxis property determines the x axis against which the data in ChartDataView is plotted.
YAxis	The YAxis property determines the y axis against which the data in ChartDataView is plotted.

A.2 ChartDataViewSeries

Name	Description
DrawingOrder	The DrawingOrder property determines the order of display of data series. When the DrawingOrder property is changed, ChartDataView will normalize the order properties of all the ChartDataViewSeries objects that it manages.
FirstPoint	The FirstPoint property controls the index of the first point displayed in the ChartDataSeries.
IsIncluded	The IsIncluded property determines whether a data series is included in chart calculations (like axis bounds).
IsShowing	The IsShowing property determines whether the data series is showing in the chart area. Note that data series that are not showing are still used in axis calculations. See the IsIncluded property for details on how to omit a data series from chart calculations.
IsShowingInLegend	The IsShowingInLegend property determines whether or not this series will appear in the chart legend.
Label	The Label property controls the text that appears next to the data series inside the legend. It can be an unparsed JCString.
LastPoint	The LastPoint property controls the index of the first point displayed in the ChartDataSeries.
Name	The Name property represents the name of the data series. In JClass Chart, data series are named, and can be retrieved by name.
Style	The Style property defines the rendering style for the data series.
X	The X property determines the x value at a specified index.
Y	The Y property determines the y value at a specified index.

A.3 ChartRegion

Name	Description
Background	The Background property determines the foreground color used to draw inside the chart region. Note that the Background property is inherited from the parent JCChart.
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT.
BorderWidth	The BorderWidth property determines the width of the border drawn around the region.

Name	Description
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent JCChart.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent JCChart.
Height	The Height property determines the height of the ChartRegion.
HeightIsDefault	The HeightIsDefault property determines whether the height of the chart region is calculated by Chart (true) or taken from the Height property (false).
Insets	The Insets property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.
IsShowing	The IsShowing property determines whether the associated ChartRegion is currently visible.
Left	The Left property determines the location of the left of the ChartRegion
LeftIsDefault	The LeftIsDefault property determines whether the left position of the chart region is calculated by Chart (true) or taken from the Left property (false).
Name	The Name property specifies a string identifier for the ChartRegion object.
Parent	The Parent property assures the connection to the chart on which the ChartRegion appears. Default value is null.
ParentRegion	The ParentRegion property is the ChartRegion parent. Default value is null.
Top	The Top property determines the location of the top of the ChartRegion.
TopIsDefault	The TopIsDefault property determines whether the top position of the chart region is calculated by Chart (true) or taken from the Top property (false).
Width	The Width property determines the width of the ChartRegion.
WidthIsDefault	The WidthIsDefault property determines whether the width of the chart region is calculated by Chart (true) or taken from the Width property (false).

A.4 ChartText

Name	Description
Adjust	The Adjust property determines how text is justified (positioned) in the label. Valid values include ChartText.LEFT, ChartText.CENTER and ChartText.RIGHT. The default value is ChartText.LEFT.
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent ChartRegion.
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT.
BorderWidth	The BorderWidth property determines the width of the border drawn around the region. The default value is 0 .
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent ChartRegion.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent ChartRegion.
Height	The Height property determines the height of the ChartRegion. The default value is calculated.
HeightIsDefault	The HeightIsDefault property determines whether the height of the chart region is calculated by Chart (true) or taken from the Height property (false). The default value is true.
Insets	The Insets property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.
IsShowing	The IsShowing property determines whether the associated ChartRegion is currently visible. Default value is true.
Left	The Left property determines the location of the left of the ChartRegion. The default value is calculated.
LeftIsDefault	The LeftIsDefault property determines whether the left position of the chart region is calculated by Chart (true) or taken from the Left property (false). The default value is true.
Name	The Name property specifies a string identifier for the ChartRegion object.
Parent	The Parent property assures the connection to the chart on which the ChartRegion appears. Default value is null.
ParentRegion	The ParentRegion property is the ChartRegion parent. Default value is null.

Name	Description
Rotation	The Rotation property controls the rotation of the label. Valid values include ChartText.DEG_90, ChartText.DEG_180, ChartText.DEG_270 and ChartText.DEG_0. The default value is ChartText.DEG_0.
Text	The Text property is a string property that represents the text to be displayed inside the chart label. In some cases, the Text property is used to create a JCString instance. Default value is " " (empty string).
Top	The Top property determines the location of the top of the ChartRegion. The default value is calculated.
TopIsDefault	The TopIsDefault property determines whether the top position of the chart region is calculated by Chart (true) or taken from the Top property (false). The default value is true.
Width	The Width property determines the width of the ChartRegion. The default value is calculated.
WidthIsDefault	The WidthIsDefault property determines whether the width of the chart region is calculated by Chart (true) or taken from the Width property (false). The default value is true.

A.5 JCAxis

Name	Description
AnnotationMethod	The AnnotationMethod property determines how axis annotations are generated. Valid values are JCAxis.VALUE (annotation is generated by Chart, with possible callbacks to a label generator); JCAxis.VALUE_LABELS (annotation is taken from a list of value labels provided by the user -- a value label is a label that appears at a particular axis value); JCAxis.POINT_LABELS (annotation comes from the data source's point labels that are associated with particular data points); and JCAxis.TIME_LABELS (Chart generates time/date labels based on the TimeUnit, TimeBase and TimeFormat properties). Default value is JCAxis.VALUE.
AnnotationRotation	The AnnotationRotation property specifies the rotation of each axis label. Valid values are JCAxis.ROTATE_90, JCAxis.ROTATE_180, JCAxis.ROTATE_270 or JCAxis.ROTATE_NONE. Default value is JCAxis.ROTATE_NONE.
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent ChartRegion.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent ChartRegion.

Name	Description
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent ChartRegion.
Formula	The Formula property determines how an axis is related to another axis object. If set, the Formula property overrides all other axis properties. See JCAxisFormula for details.
Gap	The Gap property determines the amount of space left between adjacent axis annotations.
GeneratedValueLabels	The GeneratedValueLabels property reveals the value label at the specified index in the list of value labels generated for this axis.
GridIsShowing	The GridIsShowing property determines whether a grid is drawn for the axis. Default value is false.
GridSpacing	The GridSpacing property controls the spacing between grid lines relative to the axis. Default value is 0.0.
GridSpacingIsDefault	The GridSpacingIsDefault property determines whether Chart is responsible for calculating the grid spacing value. If true, Chart will calculate the grid spacing. If false, Chart will use the provided grid spacing. Default value is true.
GridStyle	The GridStyle property controls how grids are drawn. The default value is generated.
Height	The Height property determines the height of the ChartRegion. The default value is calculated.
HeightIsDefault	The HeightIsDefault property determines whether the height of the chart region is calculated by Chart (true) or taken from the Height property (false). Default value is true.
IsEditable	The IsEditable property determines whether the axis can be affected by edit/translation/zooming. Default value is true.
IsLogarithmic	The IsLogarithmic property determines whether the axis will be logarithmic (true) or linear (false). Default value is false.
IsReversed	The IsReversed property of JCAxis determines if the axis order is reversed. Default value is false.
IsShowing	The IsShowing property determines whether the associated Axis is currently visible. Default value is true. Note that the Font property is inherited from the parent ChartRegion.
IsVertical	The IsVertical property determines whether the associated Axis is vertical. Default value is false.

Name	Description
LabelGenerator	The LabelGenerator property holds a reference to an object that implements the JCLabelGenerator interface. This interface is used to externally generate labels if the AnnotationMethod property is set to JCAxis.VALUE. Default value is null.
Left	The Left property determines the location of the left of the ChartRegion. The default value is calculated.
LeftIsDefault	The LeftIsDefault property determines whether the left position of the chart region is calculated by Chart (true) or taken from the Left property (false). Default value is true.
Max	The Max property controls the maximum value shown on the axis. The data max is determined by Chart. Default value is calculated.
MaxIsDefault	The MaxIsDefault property determines whether Chart is responsible for calculating the maximum axis value. If true, Chart calculates the axis max. If false, Chart uses the provided axis max. Default value is true.
Min	The Min property controls the minimum value shown on the axis. The data min is determined by Chart. Default value is calculated.
MinIsDefault	The MinIsDefault property determines whether Chart is responsible for calculating the minimum axis value. If true, Chart will calculate the axis min. If false, Chart will use the provided axis min. Default value is true.
Name	The Name property specifies a string identifier for the ChartRegion object. Note that the Name property is inherited from the parent ChartRegion.
Parent	The Parent property assures the connection to the chart on which the JCAxis appears. Default value is null. Note that the Parent property is inherited from the parent ChartRegion.
ParentRegion	The ParentRegion property is the ChartRegion parent. Default value is null. Note that the ParentRegion property is inherited from the parent ChartRegion.
NumSpacing	The NumSpacing property controls the interval between axis labels. The default value is calculated.
NumSpacingIsDefault	The NumSpacingIsDefault property determines whether Chart is responsible for calculating the numbering spacing. If true, Chart will calculate the spacing. If false, Chart will use the provided numbering spacing. Default value is true.
Origin	The Origin property controls location of the origin along the axis. The default value is calculated.
OriginIsDefault	The OriginIsDefault property determines whether Chart is responsible for positioning the axis origin. If true, Chart calculates the axis origin. If false, Chart uses the provided axis origin value. Default value is true.

Name	Description
OriginPlacement	The OriginPlacement property specifies where the origin is placed. Note that the OriginPlacement property is only active if the Origin property has not been set. Valid values include AUTOMATIC (places origin at minimum value). ZERO (places origin at zero), MIN (places origin at minimum value on axis) or MAX (places origin at maximum value on axis). Default value is AUTOMATIC.
OriginPlacementIsDefault	The OriginPlacementIsDefault property determines whether Chart is responsible for determining the location of the axis origin. If true, Chart calculates the origin positioning. If false, Chart uses the provided origin placement.
Placement	The Placement property determines the method used to place the axis. Valid values include JCAxis.AUTOMATIC (Chart chooses an appropriate location), JCAxis.ORIGIN (appears at the origin of another axis, specified via the PlacementAxis property), JCAxis.MIN (appears at the minimum axis value), JCAxis.MAX (appears at the maximum axis value) or JCAxis.VALUE_ANCHORED (appears at a particular value along another axis, specified via the PlacementAxis property). Default value is AUTOMATIC.
PlacementAxis	The PlacementAxis property determines the axis that controls the placement of this axis. In JCChart, it is possible to position an axis at a particular position on another axis (in conjunction with the PlacementLocation property or the Placement property). Default value is null.
PlacementIsDefault	The PlacementIsDefault property determines whether Chart is responsible for determining the location of the axis. If true, Chart calculates the axis positioning. If false, Chart uses the provided axis placement.
PlacementLocation	The PlacementLocation property is used with the PlacementAxis property to position the current axis object at a particular point on another axis. Default value is 0.0.
Precision	The Precision property controls the number of zeros that appear after the decimal place in chart-generated axis labels. The default value is calculated.
PrecisionIsDefault	The PrecisionIsDefault determines whether Chart is responsible for calculating the numbering precision. If true, Chart will calculate the precision. If false, Chart will use the provided precision. Default value is true.
TickSpacing	The TickSpacing property controls the interval between tick lines on the axis. Note: if the AnnotationMethod property is set to POINT_LABELS, tick lines appear at point values. The default value is calculated.

Name	Description
TickSpacingIsDefault	The TickSpacingIsDefault property determines whether Chart is responsible for calculating the tick spacing. If true, Chart will calculate the tick spacing. If false, Chart will use the provided tick spacing. Default value is true.
TimeBase	The TimeBase property defines the start time for the axis. Default value is the current time.
TimeFormat	The TimeFormat property controls the format used to generate time labels for time labelled axes. The formats supported are similar to those supported by the C function strftime(). Default value is calculated based on TimeUnit.
TimeFormatIsDefault	The TimeFormatIsDefault property determines whether a time label format is generated automatically, or the user value for TimeFormat is used. Default value is true.
TimeUnit	The TimeUnit property controls the unit of time used for labelling a time labelled axis. Valid TimeUnit values include JCAxis.SECONDS, JCAxis.MINUTES, JCAxis.HOURS, JCAxis.DAYS, JCAxis.WEEKS, JCAxis.MONTHS and JCAxis.YEARS. Default value is JCAxis.SECONDS.
Title	The Title property controls the appearance of the axis title.
Top	The Top property determines the location of the top of the ChartRegion. The default value is calculated.
TopIsDefault	The TopIsDefault property determines whether the top position of the chart region is calculated by Chart (true) or taken from the Top property (false). Default value is true.
ValueLabels	The ValueLabels property is an indexed property containing a list of all annotations for an axis. Default value is null, no value labels.
Width	The Width property determines the width of the ChartRegion. The default value is calculated.
WidthIsDefault	The WidthIsDefault property determines whether the width of the chart region is calculated by Chart (true) or taken from the Width property (false). Default value is true.

A.6 JCAxisFormula

Name	Description
Constant	The Constant property specifies the "c" value in the axis relationship y2 = my + c.
Multiplier	The Multiplier property specifies the "m" value in the relationship y2 = my + c.
Originator	The Originator property specifies an object representing the axis that is related to the current axis by the formula y = mx + c. The originator is "x".

A.7 JCAxisTitle

Name	Description
Adjust	The Adjust property determines how text is justified (positioned) in the label. If the contents of the ChartText are a JCString, this has no effect. Valid values include ChartText.LEFT, ChartText.CENTER and ChartText.RIGHT. Default value is ChartText.LEFT.
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent ChartText.
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT. Default is null.
BorderWidth	The BorderWidth property determines the width of the border drawn around the region.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent ChartText.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent ChartText.
Height	The Height property defines the height of the chart region. The default value is calculated.
HeightIsDefault	The HeightIsDefault property determines whether the height of the chart region is calculated by Chart (true) or taken from the Height property (false).
Insets	The Insets property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.

Name	Description
IsShowing	The IsShowing property determines whether the associated Axis is currently visible. Default value is true.
Left	The Left property determines the location of the left of the chart region. This property is read-only.
LeftIsDefault	The LeftIsDefault property determines whether the left position of the chart region is calculated by Chart (true) or taken from the Left property (false).
Placement	The Placement property controls where the JCAxis title is placed relative to the "opposing" axis. Valid values include JCLegend.NORTH or JCLegend.SOUTH for horizontal axes, and JCLegend.EAST, JCLegend.WEST, JCLegend.NORTHEAST, JCLegend.SOUTHEAST, JCLegend.NORTHWEST or JCLegend.SOUTHEAST for vertical axes. The default value is calculated.
PlacementIsDefault	The PlacementIsDefault property determines whether Chart is responsible for calculating a reasonable default placement for the axis title. Default value is true.
Rotation	The Rotation property controls the rotation of the label. Valid values include ChartText.DEG_90, ChartText.DEG_180, ChartText.DEG_270 and ChartText.DEG_0. Default value is ChartText.DEG_0.
Text	The Text property is a string property that represents the text to be displayed inside the chart label. In some cases, the Text property is used to create a JCString instance. Default value is " " (nothing).
Top	The Top property determines the location of the top of the chart region. This property is read-only.
TopIsDefault	The TopIsDefault property determines whether the top position of the chart region is calculated by Chart (true) or taken from the Top property (false).
Width	The Width property defines the width of the chart region. The default value is calculated.
WidthIsDefault	The WidthIsDefault property determines whether the width of the chart region is calculated by Chart (true) or taken from the Width property (false).

A.8 JCBarChartFormat

Name	Description
100Percent	The 100Percent property determines whether stacking bar charts will be charted versus an axis representing a percentage between 0 and 100. Default value is false.
ClusterOverlap	The ClusterOverlap property specifies the overlap between bars. Valid values are between -100 and 100. Default value is 0.
ClusterWidth	The ClusterWidth property determines the percentage of available space which will be occupied by the bars. Valid values are between 0 and 100. Default value is 80.

A.9 JCBorderStyle

Name	Description
Type	The Type property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT. Default value is FRAME_IN.
Width	The Width property determines the width of the border. Default value is 0.

A.10 JCCandleChartFormat

Name	Description
CandleOutlineStyle	The CandleOutlineStyle determines the the candle outline style of the complex candle chart.
FallingCandleStyle	The FallingCandleStyle determines the candle style of the falling candle style of the complex candle chart.
HiloStyle	The HiloStyle determines the candle style of the simple candle or the HiLo line of the complex candle chart.
IsComplex	The IsComplex property determines whether candle charts use the simple or the complex display style. When false, Chart only uses the style referenced by getHiLoStyle() for the candle appearance. When set to true, all four styles are used. Default value is false.
RisingCandleStyle	The RisingCandleStyle determines the rising candle style of the complex candle chart.

A.11 JCChart

Name	Description
About	The About property displays contact information for KL Group in the bean box.
AllowUserChanges	The AllowUserChanges property determines whether the user viewing the graph can modify graph values. Default value is false.
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent JCComponent.
BorderType	Determines the style of border drawn around the Chart. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT..
BorderWidth	The BorderWidth property determines the width of the border drawn around the chart. Default value is 0.
BottomMargin	The BottomMargin property controls the bottom margin on the chart. Default value is 1.
CancelKey	The CancelKey property specifies the key used to perform a cancel operation.
Changed	The Changed property determines whether the chart requires recalculation. Default value is false.
ChartArea	The ChartArea property controls the object that controls the display of the graph. Default value is null.
ChartLabels	The ChartLabels property controls a list of chart labels managed by the chart. Default value is an empty JCVector.
DataView	The DataView property is an indexed property that contains all the data to be displayed in Chart. See ChartDataView for details on data format. By default, one ChartDataView is created.
DoubleBuffer	The DoubleBuffer property controls whether the chart rendering uses an off-screen image or renders directly to the screen. Default value is true.
FillColorIndex	The FillColorIndex property controls the fill color index. Default value is 0.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent JCComponent.
Footer	The Footer property controls the object that controls the display of the footer. Default value is null.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent JCComponent.
Header	The Header property controls the object that controls the display of the header. Default value is null.

Name	Description
IsBatched	The IsBatched property controls whether chart updates are accumulated. Default value is false.
Legend	The Legend property controls the object that controls the display of the legend. Default value is null.
LeftMargin	The LeftMargin property controls the left margin on the chart. Default value is 1.
LineColorIndex	The LineColorIndex property controls the line color index. Default value is 0.
NumChartLabels	The NumChartLabels property indicates how many chart labels there are. This property is read-only. Default value is calculated.
NumData	The NumData property indicates how many ChartDataView objects are stored in JCChart. This is a read-only property. Default value is 1.
NumTriggers	The NumTriggers property indicates how many event triggers have been specified.
ResetKey	The ResetKey property specifies the key used to perform a reset operation.
RightMargin	The RightMargin property controls the right margin on the chart. Default value is 1.
SaveParamStream	The SaveParamStream property specifies the output stream to save Chart HTML tags to. Default value is null.
SymbolColorIndex	The SymbolColorIndex property controls the symbol color index. Default value is 0.
SymbolShapeIndex	The SymbolShapeIndex property controls the symbol shape index. Default value is 1.
TopMargin	The TopMargin property controls the top margin on the chart. Default value is 1.
Trigger	The Trigger property is an indexed property that contains all the information necessary to map user events into Chart actions. The Trigger property is made up of a number of EventTrigger objects. Default value is empty.
Version	The Version property specifies the JClass Chart version number. This property is read-only. Default value is calculated.

A.12 JCChartArea

Name	Description
AngleUnit	The AngleUnit property determines the unit of all angle values. Default value is DEGREES.
AxisBoundingBox	The AxisBoundingBox property determines whether a box is drawn around the area bound by the inner axes.
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent ChartRegion.
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT.
BorderWidth	The BorderWidth property determines the width of the border drawn around the chart area. (Note that legend, header, footer and chart area are all ChartRegion instances). Default value is 0.
Depth	The Depth property controls the apparent depth of a graph. Default value is 0.0.
Elevation	The Elevation property controls distance from the X axis. Default value is 0.0.
FastAction	The FastAction property determines whether chart actions will use an optimized mode in which it does not bother to display axis annotations or gridlines. Default value is false.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent ChartRegion.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent ChartRegion.
Height	The Height property determines the height of the ChartRegion. The default value is calculated.
HeightIsDefault	The HeightIsDefault property determines whether the height of the chart area is calculated by Chart (true) or taken from the Height property (false). Default value is false.
HorizActionAxis	The HorizActionAxis property determines the axis used for actions (zooming, translating) in the horizontal direction. Default value is null.
Insets	The Insets property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.
IsShowing	If true, the ChartRegion will appear on the screen. If false, it will not appear on the screen. (Note that legend, header, footer and chart area are all ChartRegion instances). Default value is true.

Name	Description
Left	The Left property determines the location of the left of the ChartRegion. The default value is calculated.
LeftIsDefault	The LeftIsDefault property determines whether the left position of the chart area is calculated by Chart (true) or taken from the Left property (false). Default value is false.
Markers	The Markers property is an indexed property that controls the markers that appear on a particular JCChartArea instance. JCChartArea will pre-allocate an array of JCMarkers. Initially, the markers will have their axis and series properties set to null, so the markers will not appear.
PlotArea	The PlotArea property represents the region of the ChartArea that is inside the axes.
Rotation	The Rotation property controls the position of the eye relative to the Y axis. Default value is 0.0.
Top	The Top property determines the location of the top of the ChartRegion. The default value is calculated.
TopIsDefault	The TopIsDefault property determines whether the top position of the chart area is calculated by Chart (true) or taken from the Top property (false). Default value is false.
VertActionAxis	The VertActionAxis property determines the axis used for actions (zooming, translating) in the vertical direction. Default value is null.
Width	The Width property determines the width of the ChartRegion. The default value is calculated.
WidthIsDefault	The WidthIsDefault property determines whether the width of the chart area is calculated by Chart (true) or taken from the Width property (false). Default value is false.
XAxis	The XAxis property is an indexed property that contains all the x axes for the chart area. Default value is one x axis.
YAxis	The YAxis property is an indexed property that contains all the y axes for the chart area. Default value is one y axis.

A.13 JCChartComponent

Name	Description
AllowUserChanges	The AllowUserChanges property determines whether the user viewing the graph can modify graph values. Default value is false.
AxisBoundingBox	The AxisBoundingBox property determines whether a box is drawn around the area bound by the inner axes. (AxisBoundingBox is actually a property of JCChartArea).

Name	Description
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent JCComponent.
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT.
BorderWidth	The BorderWidth property determines the width of the border drawn around the chart. Default value is 0.
BottomMargin	The BottomMargin property controls the bottom margin on the chart. Default value is 1.
CancelKey	The CancelKey property specifies the key used to perform a cancel operation.
Changed	The Changed property determines whether the chart requires recalculation. Default value is false.
ChartArea	The ChartArea property controls the object that controls the display of the graph. Default value is null.
ChartLabels	The ChartLabels property controls a list of chart labels managed by the chart. Default value is an empty JCVector.
DataView1	The DataView1 property controls the file or URL used for the first set of data in chart.
DataView1ChartType	The DataView1ChartType property determines the chart type of the first set of data in the chart.
DataSources	The DataSources property is an indexed property used to get data into Chart. Each element in DataSources is either a filename or a valid URL from which properly-formatted data can be retrieved.
DataViewIsInverted	The DataViewIsInverted property determines whether the x and y axis are inverted.
DataView2	The DataView2 property controls the file or URL used for the first set of data in chart.
DataView2ChartType	The DataView2ChartType property determines the chart type of the first set of data in the chart.
Depth	The Depth property controls the apparent depth of a graph when displayed in 3D mode. (Depth is actually a property of JCChartArea). Default value is 0.0.
DoubleBuffer	The DoubleBuffer property controls whether the chart rendering uses an off-screen image or renders directly to the screen. Default value is true.

Name	Description
FillColorIndex	The FillColorIndex property controls the fill color index. Default value is 0.
Elevation	The Elevation property controls the distance form the x axis when the chart is displayed in 3D mode. (Elevation is actually a property of JCChartArea). Default value is 0.0.
FastAction	The FastAction property determines whether chart actions will use an optimized mode in which it does not bother to display axis annotations or grid lines. (FastAction is actually a property of JCChartArea).
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent JCComponent.
FooterIsShowing	The FooterIsShowing property determines whether the footer is visible. Default value is false.
FooterText	The FooterText property holds the text that is displayed in the footer. Default value is " " (empty string).
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent JCComponent.
HeaderIsShowing	The HeaderIsShowing property determines whether the header is visible. Default value is false.
HeaderText	The HeaderText property holds the text that is displayed in the header. Default value is " " (empty string).
IsBatched	The IsBatched property controls whether chart updates are accumulated. Default value is false.
LeftMargin	The LeftMargin property controls the left margin on the chart. Default value is 1.
Legend	The Legend property controls the object that controls the display of the legend. Default value is null.
LegendAnchor	The LegendAnchor property determines the position of the legend relative to the ChartArea. Valid values include NORTH, SOUTH, EAST, WEST, NORTHWEST, SOUTHWEST, NORTHEAST and SOUTHEAST. Default value is EAST.
LegendIsShowing	The LegendIsShowing property determines whether the legend is visible. Default value is false.
LegendOrientation	The LegendOrientation property determines how legend information is laid out. Valid values include VERTICAL and HORIZONTAL. Default value is VERTICAL.

Name	Description
LineColorIndex	The LineColorIndex property controls the line color index. Default value is 0.
NumChartLabels	The NumChartLabels property indicates how many chart labels there are. This property is read-only. Default value is calculated.
NumData	The NumData property indicates how many ChartDataView objects are stored in JCChart. This is a read-only property. Default value is 1.
NumTriggers	The NumTriggers property indicates how many event triggers have been specified.
ResetKey	The ResetKey property specifies the key used to perform a reset operation.
RightMargin	The RightMargin property controls the right margin on the chart. Default value is 1.
Rotation	The Rotation property controls the position of the eye relative to the y axis when the chart is displayed in 3D mode. (Rotation is actually a property of JCChartArea). Default value is 0.0.
SaveParamStream	The SaveParamStream property specifies the output stream to save Chart HTML tags to. Default value is null.
SymbolColorIndex	The SymbolColorIndex property controls the symbol color index. Default value is 0.
SymbolShapeIndex	The SymbolShapeIndex property controls the symbol shape index. Default value is 1.
TopMargin	The TopMargin property controls the top margin on the chart. Default value is 1.
Trigger	The Trigger property is an indexed property that contains all the information necessary to map user events into Chart actions. The Trigger property is made up of a number of EventTrigger objects. Default value is empty.
Version	The Version property specifies the JClass Chart version number. This property is read-only. Default value is calculated.
X1AxisAnnotationMethod	The X1AxisAnnotationMethod property determines how axis annotations are generated. Valid values include VALUE (annotation is generated by Chart, with possible callbacks to a label generator); VALUE_LABELS (annotation is taken from a list of value labels provided by the user -- a value label is a label that appears at a particular axis value); POINT_LABELS (annotation comes from the data source's point labels that are associated with particular data points); and TIME_LABELS (Chart generates time/date labels based on the TimeUnit, TimeBase and TimeFormat properties). Default value is VALUE.

Name	Description
X1AxisAnnotationRotation	The X1AxisAnnotationRotation property specifies the rotation of each axis label. Valid values are ROTATE_90, ROTATE_180, ROTATE_270 or ROTATE_NONE. Default value is ROTATE_NONE.
X1AxisGridIsShowing	The X1AxisGridIsShowing property determines whether a grid is drawn for the axis. Default value is false.
X1AxisGridSpacing	The X1AxisGridSpacing property controls the spacing between grid lines relative to the axis. Default value is calculated.
X1AxisGridSpacingIsDefault	The X1AxisGridSpacingIsDefault property determines whether Chart is responsible for calculating the grid spacing value. If true, Chart will calculate the grid spacing. If false, Chart will use the provided grid spacing. Default value is true.
X1AxisIsLogarithmic	The X1AxisIsLogarithmic property determines whether the first x axis will be logarithmic (true) or linear (false). Default value is false.
X1AxisIsReversed	The X1AxisIsReversed property determines whether the first x axis will be reversed in direction. Default value is false.
X1AxisIsShowing	The X1AxisIsShowing property determines whether the first x axis is currently visible. Default value is true.
X1AxisMax	The X1AxisMax property controls the maximum value shown on the axis. The data max is determined by Chart. Default value is calculated.
X1AxisMaxIsDefault	The X1AxisMaxIsDefault property determines whether Chart is responsible for calculating the maximum axis value. If true, Chart will calculate the axis max. If false, Chart will use the provided axis max. Default value is true.
X1AxisMin	The X1AxisMin property controls the minimum value shown on the axis. The data min is determined by Chart. Default value is calculated.
X1AxisMinIsDefault	The X1AxisMinIsDefault property determines whether Chart is responsible for calculating the minimum axis value. If true, Chart will calculate the axis min. If false, Chart will use the provided axis min. Default value is true.
X1AxisNumSpacing	The X1AxisNumSpacing property controls the interval between axis labels. Default value is calculated.

Name	Description
X1AxisNumSpacingIsDefault	The X1AxisNumSpacingIsDefault property determines whether Chart is responsible for calculating the numbering spacing. If true, Chart will calculate the spacing. If false, Chart will use the provided numbering spacing. Default value is true.
X1AxisOrigin	The X1AxisOrigin property controls location of the origin along the axis. Default value is calculated.
X1AxisOriginIsDefault	The X1AxisOriginIsDefault property determines whether Chart is responsible for positioning the axis origin. If true, Chart calculates the axis origin. If false, Chart uses the provided axis origin value. Default value is true.
X1AxisOriginPlacement	The X1AxisOriginPlacement property specifies where the origin is placed. Note that the X1AxisOriginPlacement property is only active if the X1AxisOrigin property has not been set. Valid values include AUTOMATIC (places origin at minimum value or at zero if there are negative and positive values), ZERO (places origin at zero), MIN (places origin at minimum value on axis) or MAX (places origin at maximum value on axis). Default value is calculated.
X1AxisPlacement	The X1AxisPlacement property determines the method used to place the first x axis. Valid values include AUTOMATIC (Chart chooses an appropriate location), ORIGIN (appears at the origin of another axis), MIN (appears at the minimum axis value), MAX (appears at the maximum axis value) or VALUE_ANCHORED (appears at a particular value along another axis). Default value is AUTOMATIC.
X1AxisPrecision	The X1AxisPrecision property controls the number of zeros that appear after the decimal place in chart-generated axis labels. Default value is calculated.
X1AxisPrecisionIsDefault	The X1AxisPrecisionIsDefault determines whether Chart is responsible for calculating the numbering precision. If true, Chart will calculate the precision. If false, Chart will use the precision provided in X1AxisPrecision. Default value is true.
X1AxisTickSpacing	The X1AxisTickSpacing property controls the interval between tick lines on the axis. Note: if the X1AxisAnnotationMethod property is set to POINT_LABELS, tick lines appear at point values. Default value is calculated.
X1AxisTickSpacingIsDefault	The X1AxisTickSpacingIsDefault property determines whether Chart is responsible for calculating the tick spacing. If true, Chart will calculate the tick spacing. If false, Chart will use the provided tick spacing. Default value is true.

Name	Description
X1AxisTimeFormat	The X1AxisTimeFormat property controls the format used to generate time labels for time labelled axes. The formats supported are similar to those supported by the C function strftime(). Default value is based on value of X1AxisTimeUnit.
X1AxisTimeFormatIsDefault	The X1AxisTimeFormatIsDefault property determines whether a time label format is generated automatically, or the user value for X1AxisTimeFormat is used. Default value is true.
X1AxisTimeUnit	The X1AxisTimeUnit property controls the unit of time used for labelling a time labelled axis. Valid X1AxisTimeUnit values include SECONDS, MINUTES, HOURS, DAYS, WEEKS, MONTHS and YEARS. Default value is SECONDS.
X1AxisTitleIsShowing	The X1AxisTitleIsShowing property determines whether the x1 axis is visible. Default value is false.
X1AxisTitlePlacement	The X1AxisTitlePlacement property controls where the JCAxis title is placed relative to the “opposing” axis. Valid values include JCLegend.NORTH or JCLegend.SOUTH for horizontal axes, and JCLegend.EAST, JCLegend.WEST, JCLegend.NORTHEAST, JCLegend.SOUTHEAST, JCLegend.NORTHWEST or JCLegend.SOUTHEAST for vertical axes. Default value is calculated.
X1AxisTitlePlacementIsDefault	The X1AxisTitlePlacementIsDefault property determines whether Chart is responsible for calculating a reasonable default placement for the axis title. Default value is true.
X1AxisTitleText	The X1AxisTitleText property specifies the text that will appear as the x1 axis title. Default value is "" (empty string).
Y1AxisAnnotationMethod	The Y1AxisAnnotationMethod property determines how axis annotations are generated. Valid values include VALUE (annotation is generated by Chart, with possible callbacks to a label generator); VALUE_LABELS (annotation is taken from a list of value labels provided by the user -- a value label is a label that appears at a particular axis value); POINT_LABELS (annotation comes from the data source's point labels that are associated with particular data points); and TIME_LABELS (Chart generates time/date labels based on the TimeUnit, TimeBase and TimeFormat properties). Default value is VALUE.
Y1AxisAnnotationRotation	The Y1AxisAnnotationRotation property specifies the rotation of each axis label. Valid values are ROTATE_90, ROTATE_180, ROTATE_270 or ROTATE_NONE. Default value is ROTATE_NONE.
Y1AxisGridIsShowing	The Y1AxisGridIsShowing property determines whether a grid is drawn for the axis.

Name	Description
`Y1AxisGridSpacing`	The `Y1AxisGridSpacing` property controls the spacing between grid lines relative to the axis.
`Y1AxisGridSpacingIsDefault`	The `Y1AxisGridSpacingIsDefault` property determines whether Chart is responsible for calculating the grid spacing value. If `true`, Chart will calculate the grid spacing. If `false`, Chart will use the provided grid spacing.
`Y1AxisIsLogarithmic`	The `Y1AxisIsLogarithmic` property determines whether the first `y` axis will be logarithmic (`true`) or linear (`false`). Default value is `false`.
`Y1AxisIsReversed`	The `Y1AxisIsReversed` property determines whether the first `y` axis will be reversed in direction. Default value is `false`.
`Y1AxisIsShowing`	The `Y1AxisIsShowing` property determines whether the first `y` axis is currently visible. Default value is `true`.
`Y1AxisMax`	The `Y1AxisMax` property controls the maximum value shown on the axis. The data max is determined by Chart. Default value is calculated.
`Y1AxisMaxIsDefault`	The `Y1AxisMaxIsDefault` property determines whether Chart is responsible for calculating the maximum axis value. If `true`, Chart will calculate the axis max. If `false`, Chart will use the provided axis max. Default value is `true`.
`Y1AxisMin`	The `Y1AxisMin` property controls the minimum value shown on the axis. The data max is determined by Chart. Default value is calculated.
`Y1AxisMinIsDefault`	The `Y1AxisMinIsDefault` property determines whether Chart is responsible for calculating the minimum axis value. If `true`, Chart will calculate the axis min. If `false`, Chart will use the provided axis min. Default value is `true`.
`Y1AxisNumSpacing`	The `Y1AxisNumSpacing` property controls the interval between axis labels. Default value is calculated.
`Y1AxisNumSpacingIsDefault`	The `Y1AxisNumSpacingIsDefault` property determines whether Chart is responsible for calculating the numbering spacing. If `true`, Chart will calculate the spacing. If `false`, Chart will use the provided numbering spacing. Default value is `true`.
`Y1AxisOrigin`	The `Y1AxisOrigin` property controls location of the origin along the axis. Default value is calculated.
`Y1AxisOriginIsDefault`	The `Y1AxisOriginIsDefault` property determines whether Chart is responsible for positioning the axis origin. If `true`, Chart calculates the axis origin. If `false`, Chart uses the provided axis origin value. Default value is `true`.

Name	Description
Y1AxisOriginPlacement	The Y1AxisOriginPlacement property specifies where the origin is placed. Note that the Y1AxisOriginPlacement property is only active if the Y1AxisOrigin property has not been set. Valid values include AUTOMATIC (places origin at minimum value or at zero if there are negative and positive values), ZERO (places origin at zero), MIN (places origin at minimum value on axis) or MAX (places origin at maximum value on axis). Default value is AUTOMATIC.
Y1AxisPlacement	The Y1AxisPlacement property determines the method used to place the first y axis. Valid values include AUTOMATIC (Chart chooses an appropriate location), ORIGIN (appears at the origin of another axis), MIN (appears at the minimum axis value), MAX (appears at the maximum axis value) or VALUE_ANCHORED (appears at a particular value along another axis). Default value is AUTOMATIC.
Y1AxisPrecision	The Y1AxisPrecision property controls the number of zeros that appear after the decimal place in chart-generated axis labels. Default value is calculated.
Y1AxisPrecisionIsDefault	The Y1AxisPrecisionIsDefault determines whether Chart is responsible for calculating the numbering precision. If true, Chart will calculate the precision. If false, Chart will use the precision provided in Y1AxisPrecision. Default value is true.
Y1AxisTickSpacing	The Y1AxisTickSpacing property controls the interval between tick lines on the axis. Note: if the Y1AxisAnnotationMethod property is set to POINT_LABELS, tick lines appear at point values. Default value is calculated.
Y1AxisTickSpacingIsDefault	The Y1AxisTickSpacingIsDefault property determines whether Chart is responsible for calculating the tick spacing. If true, Chart will calculate the tick spacing. If false, Chart will use the provided tick spacing. Default value is true.
Y1AxisTimeFormat	The Y1AxisTimeFormat property controls the format used to generate time labels for time labelled axes. The formats supported are similar to those supported by the C function strftime(). Default value is calculated based on the value of Y1AxisTimeUnit.
Y1AxisTimeFormatIsDefault	The Y1AxisTimeFormatIsDefault property determines whether a time label format is generated automatically, or the user value for Y1AxisTimeFormat is used. Default value is true.

Name	Description
Y1AxisTimeUnit	The Y1AxisTimeUnit property controls the unit of time used for labelling a time labelled axis. Valid Y1AxisTimeUnit values include SECONDS, MINUTES, HOURS, DAYS, WEEKS, MONTHS and YEARS. Default value is SECONDS.
Y1AxisTitleIsShowing	The Y1AxisTitleIsShowing property determines whether the Y1 axis is visible. Default value is false. Default value is false.
Y1AxisTitlePlacement	The Y1AxisTitlePlacement property controls where the JCAxis title is placed relative to the "opposing" axis. Valid values include JCLegend.NORTH or JCLegend.SOUTH for horizontal axes, and JCLegend.EAST, JCLegend.WEST, JCLegend.NORTHEAST, JCLegend.SOUTHEAST, JCLegend.NORTHWEST or JCLegend.SOUTHEAST for vertical axes. Default value is calculated.
Y1AxisTitlePlacementIsDefault	The Y1AxisTitlePlacementIsDefault property determines whether Chart is responsible for calculating a reasonable default placement for the axis title. Default value is true.
Y1AxisTitleText	The Y1AxisTitleText property specifies the text that will appear as the Y1 axis title. Default value is " " (empty string).

A.14 JCChartLabel

Name	Description
Adjust	The Adjust property determines how text is justified (positioned) in the label. Valid values include ChartText.LEFT, ChartText.CENTER and ChartText.RIGHT. Default value is ChartText.LEFT.
Anchor	Specifies how the label is to be positioned relative to the specified point. Valid values are JCChartLabel.NORTHEAST, JCChartLabel.NORTHWEST, JCChartLabel.NORTH, JCChartLabel.EAST, JCChartLabel.WEST, JCChartLabel.SOUTHEAST, JCChartLabel.SOUTHWEST and JCChartLabel.SOUTH.
AttachMethod	Specifies how the label is attached to the chart. Valid values are JCChartLabel.ATTACH_COORD (attach label to an absolute point anywhere on the chart), JCChartLabel.ATTACH_DATACOORD (attach label to a point in the data space on the chart area), and JCChartLabel.ATTACH_DATAINDEX (attach the label to a specific point/bar/slice on the chart)
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent JCTitle.

Name	Description
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT.
BorderWidth	The BorderWidth property determines the width of the border drawn around the chart label. Default value is 0.
Coord	The coordinate in the chart's space where the label is to be attached.
DataCoord	The coordinate in the chart area's data space where the label is to be attached.
DataIndex	A data index representing the point/bar/slice in the chart to which the label is to be attached.
DataView	For labels using ATTACH_DATACOORD, this property specifies which ChartDataView's axes should be used.
DwellDelay	Value in milliseconds representing the delay after the mouse arrives at a data point to which a Dwell Label is attached before showing the label. Default is 0, to display the label immediately.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent JCTitle.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent JCTitle.
Height	The Height property determines the height of the label. Default value is generated.
HeightIsDefault	The HeightIsDefault property determines whether the height of the label is calculated by Chart (true) or taken from the Height property (false). Default value is true.
Insets	The Insets property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.
IsDwellLabel	When IsDwellLabel is set to true, the label is only displayed when the cursor is over the point/bar/slice that the label is attached to. This property is only used when the label is attached using ATTACH_DATAINDEX. When set to false (the default), the label is always displayed.
IsShowing	The IsShowing property determines whether the associated label is currently visible. Default value is true.
Left	The Left property determines the location of the left of the label. Default value is generated.
LeftIsDefault	The LeftIsDefault property determines whether the left position of the label is calculated by Chart (true) or taken from the Left property (false). Default value is true.
Name	The Name property specifies a string identifier for the JCChartLabel object.

Name	Description
Offset	Offset specifies where the label should be positioned relative to the position the labels thinks it should be, depending on what the label's attachMethod is.
Parent	The Parent property assures the connection to the chart on which the JCChartLabel appears. Default value is null.
ParentRegion	The ParentRegion property is the ChartRegion parent. Default value is null.
Rotation	The Rotation property specifies the label's rotation (counterclockwise). Valid values are ChartText.DEG_90, ChartText.DEG_180, ChartText.DEG_270 and ChartText.DEG_0. The default value is ChartText.DEG_0.
Text	The Text property controls the text displayed inside the label.
Top	The Top property determines the location of the top of the label. The default value is calculated.
TopIsDefault	The TopIsDefault property determines whether the top position of the label is calculated by Chart (true) or taken from the Top property (false). Default value is true.
Width	The Width property determines the width of the label. Default value is generated.
WidthIsDefault	The WidthIsDefault property determines whether the width of the label is calculated by Chart (true) or taken from the Width property (false). Default value is true.

A.15 JCChartStyle

Name	Description
FillColor	The FillColor property determines the color used to fill regions in chart. Default value is generated.
FillImage	The FillImage property determines the image used to paint the fill region of bar charts. Default value is null.
FillPattern	The FillPattern property determines the fill pattern used to fill regions in chart. *Note: Since Java does not support patterned fills, this property is not supported.* Default value is JCFillStyle.SOLID.
FillStyle	The FillStyle property controls the appearance of filled areas in chart. See JCFillStyle for additional properties. Note that all JCChartStyle properties of the format Fill* are virtual properties that map to properties of JCFillStyle.
LineColor	The LineColor property determines the color used to draw a line. Default value is generated.

Name	Description
`LinePattern`	The `LinePattern` property dictates the pattern used to draw a line. Valid values include `JCLineStyle.NONE`, `JCLineStyle.SOLID`, `JCLineStyle.LONG_DASH`, `JCLineStyle.SHORT_DASH`, `JCLineStyle.LSL_DASH` and `JCLineStyle.DASH_DOT`. *Note: Since Java does not support line patterns, this property is currently not supported.* Default value is `JCLineStyle.SOLID`.
`LineStyle`	The `LineStyle` property controls the appearance of lines in chart. See `JCLineStyle` for additional properties.
`LineWidth`	The `LineWidth` property controls the line width. *Note: Since Java only supports width-1 lines, this property is currently not supported.* Default value is 1.
`SymbolColor`	The `SymbolColor` property determines the color used to paint the symbol. Default value is generated.
`SymbolCustomShape`	The `SymbolCustomShape` property contains an object derived from JCShape that is used to draw points. See `JCShape` for details. Default value is `null`.
`SymbolShape`	The `SymbolShape` property determines the type of symbol that will be drawn. Valid values include `JCSymbolStyle.NONE`, `JCSymbolStyle.DOT`, `JCSymbolStyle.BOX`, `JCSymbolStyle.TRIANGLE`, `JCSymbolStyle.DIAMOND`, `JCSymbolStyle.STAR`, `JCSymbolStyle.VERT_LINE`, `JCSymbolStyle.HORIZ_LINE`, `JCSymbolStyle.CROSS`, `JCSymbolStyle.CIRCLE` and `JCSymbolStyle.SQUARE`. Default value is generated.
`SymbolSize`	The `SymbolSize` property determines the size of the symbol. Default value is 6.
`SymbolStyle`	The `SymbolStyle` property controls the symbol that represents an individual point. See `JCSymbolStyle` for additional properties. Note that all `JCChartStyle` properties of the format Symbol* are virtual properties that map to properties of `JCSymbolStyle`.

A.16 JCFillStyle

Name	Description
`Color`	The `Color` property determines the color used to fill regions in chart. The default value is generated.
`Image`	The `Image` property determines the image used to paint the fill region. Only bar charts use this property. The default value is `null`.
`Pattern`	The `Pattern` property determines the fill pattern used to fill regions in chart. *Note: Since Java does not support patterned fills, this property is not supported.* The default value is `JCFillStyle.SOLID`.

A.17 JCGridLegend

Name	Description
Anchor	The Anchor property determines the position of the legend relative to the ChartArea. Valid values include JCLegend.NORTH, JCLegend.SOUTH, JCLegend.EAST, JCLegend.WEST, JCLegend.NORTHWEST, JCLegend.SOUTHWEST, JCLegend.NORTHEAST and JCLegend.SOUTHEAST. The default value is JCLegend.EAST.
Background	The Background property determines the background color used to draw inside the legend. Note that the Background property is inherited from the parent ChartRegion.
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT.
BorderWidth	The BorderWidth property determines the width of the border drawn around the legend. The default value is 0.
Font	The Font property determines what font is used to render text inside the legend. Note that the Font property is inherited from the parent ChartRegion.
Foreground	The Foreground property determines the foreground color used to draw inside the legend. Note that the Foreground property is inherited from the parent ChartRegion.
GroupGap	The GroupGap property determines the gap between groups of items in the chart legend (e.g. the columns/rows associated with a data view).
Height	The Height property determines the height of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
HeightIsDefault	The HeightIsDefault property determines whether the height of the legend is calculated by Chart (true) or taken from the Height property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.
Insets	The Insets property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.
InsideItemGap	The InsideItemGap property determines the gap between the symbol and text portions of a legend item.
IsShowing	The IsShowing property determines whether the legend is currently visible. Default value is false.
ItemGap	The ItemGap property determines the gap between the legend items in the same group.
Left	The Left property determines the location of the left of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.

Name	Description
LeftIsDefault	The LeftIsDefault property determines whether the left position of the legend is calculated by Chart (true) or taken from the Left property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.
MarginGap	The MarginGap property determines the gap between the edge of the legend and the start of the item layout.
Name	The Name property specifies a string identifier for the JCLegend object. Note that this property is inherited from ChartRegion.
Orientation	The Orientation property determines how legend information is laid out. Valid values include JCLegend.VERTICAL and JCLegend.HORIZONTAL. The default value is JCLegend.VERTICAL.
Parent	The Parent property assures the connection to the chart on which the JCLegend appears. Default value is null. Note that this property is inherited from ChartRegion.
ParentRegion	The ParentRegion property is the JCLegend's ChartRegion parent. Default value is null. Note that this property is inherited from ChartRegion.
SymbolSize	The SymbolSize property determines the size of the symbol. Default value is 6.
Top	The Top property determines the location of the top of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
TopIsDefault	The TopIsDefault property determines whether the top position of the legend is calculated by Chart (true) or taken from the Top property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.
Width	The Width property determines the width of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
WidthIsDefault	The WidthIsDefault property determines whether the width of the legend is calculated by Chart (true) or taken from the Width property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.

A.18 JCHLOCChartFormat

Name	Description
IsShowingClose	The IsShowingClose property indicates whether the close tick indication is shown or not. The tick appears to the right of the Hi-Lo line. The default value is true.
IsShowingOpen	The IsShowingOpen property indicates whether the open tick indication is shown or not. The tick appears to the left of the Hi-Lo line. The default value is true.

Name	Description
IsOpenCloseFullWidth	The IsOpenCloseFullWidth property indicates whether the open and close tick indications are drawn across the full width of the Hi-Lo bar or just on one side. The default value is false.

A.19 JCLegend

Name	Description
Anchor	The Anchor property determines the position of the legend relative to the ChartArea. Valid values include JCLegend.NORTH, JCLegend.SOUTH, JCLegend.EAST, JCLegend.WEST, JCLegend.NORTHWEST, JCLegend.SOUTHWEST, JCLegend.NORTHEAST and JCLegend.SOUTHEAST. The default value is JCLegend.EAST.
Background	The Background property determines the background color used to draw inside the legend. Note that the Background property is inherited from the parent ChartRegion.
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT.
BorderWidth	The BorderWidth property determines the width of the border drawn around the legend. The default value is 0.
Font	The Font property determines what font is used to render text inside the legend. Note that the Font property is inherited from the parent ChartRegion.
Foreground	The Foreground property determines the foreground color used to draw inside the legend. Note that the Foreground property is inherited from the parent ChartRegion.
Height	The Height property determines the height of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
HeightIsDefault	The HeightIsDefault property determines whether the height of the legend is calculated by Chart (true) or taken from the Height property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.
IsShowing	The IsShowing property determines whether the legend is currently visible. Default value is false.
Left	The Left property determines the location of the left of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.

Name	Description
LeftIsDefault	The LeftIsDefault property determines whether the left position of the legend is calculated by Chart (true) or taken from the Left property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.
Name	The Name property specifies a string identifier for the JCLegend object. Note that this property is inherited from ChartRegion.
Orientation	The Orientation property determines how legend information is laid out. Valid values include JCLegend.VERTICAL and JCLegend.HORIZONTAL. The default value is JCLegend.VERTICAL.
Parent	The Parent property assures the connection to the chart on which the JCLegend appears. Default value is null. Note that this property is inherited from ChartRegion.
ParentRegion	The ParentRegion property is the JCLegend's ChartRegion parent. Default value is null. Note that this property is inherited from ChartRegion.
Top	The Top property determines the location of the top of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
TopIsDefault	The TopIsDefault property determines whether the top position of the legend is calculated by Chart (true) or taken from the Top property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.
Width	The Width property determines the width of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
WidthIsDefault	The WidthIsDefault property determines whether the width of the legend is calculated by Chart (true) or taken from the Width property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.

A.20 JCLineStyle

Name	Description
Color	The Color property determines the color used to draw a line. The default value is generated.
Pattern	The Pattern property dictates the pattern used to draw a line. Valid values include JCLineStyle.NONE, JCLineStyle.SOLID, JCLineStyle.LONG_DASH, JCLineStyle.SHORT_DASH, JCLineStyle.LSL_DASH and JCLineStyle.DASH_DOT. *Note: Since Java does not support line patterns, this property is currently not supported.* The default value is JCLineStyle.SOLID.
Width	The Width property controls the line width. The default value is 1.

A.21 JCMultiColLegend

Name	Description
`Anchor`	The `Anchor` property determines the position of the legend relative to the ChartArea. Valid values include `JCLegend.NORTH`, `JCLegend.SOUTH`, `JCLegend.EAST`, `JCLegend.WEST`, `JCLegend.NORTHWEST`, `JCLegend.SOUTHWEST`, `JCLegend.NORTHEAST` and `JCLegend.SOUTHEAST`. The default value is `JCLegend.EAST`.
`Background`	The `Background` property determines the background color used to draw inside the legend. Note that the `Background` property is inherited from the parent ChartRegion.
`BorderType`	The `BorderType` property determines the style of border drawn around the ChartRegion. Valid values come from `jclass.base.Border`, and include `NONE`, `ETCHED_IN`, `ETCHED_OUT`, `IN`, `OUT`, `PLAIN`, `FRAME_IN`, `FRAME_OUT`, `CONTROL_IN` and `CONTROL_OUT`.
`BorderWidth`	The `BorderWidth` property determines the width of the border drawn around the legend. The default value is 0.
`Font`	The `Font` property determines what font is used to render text inside the legend. Note that the `Font` property is inherited from the parent ChartRegion.
`Foreground`	The `Foreground` property determines the foreground color used to draw inside the legend. Note that the `Foreground` property is inherited from the parent ChartRegion.
`GroupGap`	The `GroupGap` property determines the gap between groups of items in the chart legend (e.g. the columns/rows associated with a data view).
`Height`	The `Height` property determines the height of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
`HeightIsDefault`	The `HeightIsDefault` property determines whether the height of the legend is calculated by Chart (`true`) or taken from the `Height` property (`false`). The default value is `true`. Note that the property is inherited from the parent ChartRegion.
`Insets`	The Insets property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.
`InsideItemGap`	The `InsideItemGap` property determines the gap between the symbol and text portions of a legend item.
`IsShowing`	The `IsShowing` property determines whether the legend is currently visible. Default value is `false`.
`ItemGap`	The `ItemGap` property determines the gap between the legend items in the same group.

Name	Description
Left	The Left property determines the location of the left of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
LeftIsDefault	The LeftIsDefault property determines whether the left position of the legend is calculated by Chart (true) or taken from the Left property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.
MarginGap	The MarginGap property determines the gap between the edge of the legend and the start of the item layout.
Name	The Name property specifies a string identifier for the JCLegend object. Note that this property is inherited from ChartRegion.
NumColumns	The NumColumns property determines the number of columns in this legend. If the number of columns is set to zero (the default), then the NumColumns will be adjusted automatically.
NumRows	The NumRows property determines the number of rows in this legend. If the number of rows is set to zero (the default), the number of rows will be adjusted automatically.
Orientation	The Orientation property determines how legend information is laid out. Valid values include JCLegend.VERTICAL and JCLegend.HORIZONTAL. The default value is JCLegend.VERTICAL.
Parent	The Parent property assures the connection to the chart on which the JCLegend appears. Default value is null. Note that this property is inherited from ChartRegion.
ParentRegion	The ParentRegion property is the JCLegend's ChartRegion parent. Default value is null. Note that this property is inherited from ChartRegion.
SymbolSize	The SymbolSize property determines the size of the symbol. Default value is 6.
Top	The Top property determines the location of the top of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
TopIsDefault	The TopIsDefault property determines whether the top position of the legend is calculated by Chart (true) or taken from the Top property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.
Width	The Width property determines the width of the legend. The default value is calculated. Note that the property is inherited from the parent ChartRegion.
WidthIsDefault	The WidthIsDefault property determines whether the width of the legend is calculated by Chart (true) or taken from the Width property (false). The default value is true. Note that the property is inherited from the parent ChartRegion.

A.22 JCPieChartFormat

Name	Description
ExplodeList	The ExplodeList property specifies a list of exploded pie slices in the pie charts. Default value is an empty list.
ExplodeOffset	The ExplodeOffset property specifies the distance a slice is exploded from the center of a pie chart. Default value is 10.
MinSlices	The MinSlices property represents the minimum number of pie slices that Chart will try to display before grouping slices into the other slice. Default value is 5.
OtherLabel	The OtherLabel property represents used on the "other" pie slice. As with other point labels, the "other" label is a ChartText instance. Default value is " " (empty string).
OtherStyle	The OtherStyle property specifies the style used to render the "other" pie slice.
SortOrder	The SortOrder property determines the order in which pie slices will be displayed. Note that the other slice is always last in any ordering. Valid values include JCPieChartFormat.ASCENDING_ORDER, JCPieChartFormat.DESCENDING_ORDER and JCPieChartFormat.DATA_ORDER. Default value is JCPieChartFormat.DATA_ORDER.
ThresholdMethod	The ThresholdMethod property determines how the ThresholdValue property is used. If the method is SLICE_CUTOFF, the ThresholdValue is used as a cutoff to determine what items are lumped into the other slice. If the method is PIE_PERCENTILE, items are groups into the other slice until it represents "ThresholdValue" percent of the pie. Default value is SLICE_CUTOFF.
ThresholdValue	The ThresholdValue property is a percentage value between 0.0 and 100.0. How this value is used depends on the ThresholdMethod property. Default value is 10.0.

A.23 JCSymbolStyle

Name	Description
Color	The Color property determines the color used to paint the symbol. The default value is generated.
CustomShape	The CustomShape property contains an object derived from JCShape that is used to draw points. See JCShape for details. The default value is null.

Name	Description
Shape	The Shape property determines the shape of symbol that will be drawn. Valid values include JCSymbolStyle.NONE, JCSymbolStyle.DOT, JCSymbolStyle.BOX, JCSymbolStyle.TRIANGLE, JCSymbolStyle.DIAMOND, JCSymbolStyle.STAR, JCSymbolStyle.VERT_LINE, JCSymbolStyle.HORIZ_LINE, JCSymbolStyle.CROSS, JCSymbolStyle.CIRCLE and JCSymbolStyle.SQUARE. The default value is JCSymbolStyle.DOT.
Size	The Size property determines the size of the symbol. The default value is 6.

A.24 JCTitle

Name	Description
Adjust	The Adjust property determines how text is justified or positioned in the title. Valid values include ChartText.LEFT, ChartText.CENTER and ChartText.RIGHT. Default value is ChartText.LEFT.
Background	The Background property determines the background color used to draw inside the chart region. Note that the Background property is inherited from the parent ChartText.
BorderType	The BorderType property determines the style of border drawn around the ChartRegion. Valid values come from jclass.base.Border, and include NONE, ETCHED_IN, ETCHED_OUT, IN, OUT, PLAIN, FRAME_IN, FRAME_OUT, CONTROL_IN and CONTROL_OUT.
BorderWidth	The BorderWidth property determines the width of the border drawn around the title. Default value is 0.
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent ChartText.
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent ChartText.
Height	The Height property determines the height of the JCTitle. Default value is generated.
HeightIsDefault	The HeightIsDefault property determines whether the height of the title is calculated by Chart (true) or taken from the Height property (false). Default value is true.
Insets	The Insets property specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.
IsShowing	The IsShowing property determines whether the associated title is currently visible. Default value is true.
Label	The Label property determines the label that appears inside JCTitle. For backwards-compatability. Default value is calculated.

Name	Description
`Left`	The `Left` property determines the location of the left of the JCTitle. Default value is generated.
`LeftIsDefault`	The `LeftIsDefault` property determines whether the left position of the title is calculated by Chart (`true`) or taken from the `Left` property (`false`). Default value is `true`.
`Name`	The `Name` property specifies a string identifier for the JCTitle object.
`Parent`	The `Parent` property assures the connection to the chart on which the JCTitle appears. Default value is `null`.
`ParentRegion`	The `ParentRegion` property is the ChartRegion parent. Default value is `null`.
`Rotation`	The Rotation property specifies the label's rotation (counterclockwise). Valid values are ChartText.DEG_90, ChartText.DEG_180, ChartText.DEG_270 and ChartText.DEG_0. The default value is ChartText.DEG_0.
`Text`	The `Text` property controls the text displayed inside the title. The default value is " " (empty string).
`Top`	The `Top` property determines the location of the top of the JCTitle. The default value is calculated.
`TopIsDefault`	The `TopIsDefault` property determines whether the top position of the title is calculated by Chart (`true`) or taken from the `Top` property (`false`). Default value is `true`.
`Width`	The `Width` property determines the width of the JCTitle. Default value is generated.
`WidthIsDefault`	The `WidthIsDefault` property determines whether the width of the title is calculated by Chart (`true`) or taken from the `Width` property (`false`). Default value is `true`.

A.25 JCValueLabel

Name	Description
`ChartText`	The `ChartText` property controls the ChartText associated with this Value label. The default value is a ChartText instance.
`Text`	The `Text` property specifies the text displayed inside the label. The default value is " " (empty string).
`Value`	The `Value` property controls the position of a label in data space along a particular axis. The default value is 0.0.

A.26 PlotArea

Name	Description
`Background`	The `Background` property determines the background color used to draw inside the chart region. Note that the `Background` is inherited from the parent ChartRegion.
`Bottom`	The `Bottom` property determines the location of the bottom of the PlotArea
`BottomIsDefault`	The `BottomIsDefault` property determines whether the Bottom of the chart region is calculated by Chart (`true`) or taken from the `Bottom` property (`false`).
`Foreground`	The `Foreground` property property determines the color used to draw the axis bounding box controlled by JCChartArea. Note that the `Foreground` property is inherited from the parent ChartRegion.
`Left`	The `Left` property determines the location of the left of the PlotArea
`LeftIsDefault`	The `LeftIsDefault` property determines whether the left position of the chart region is calculated by Chart (`true`) or taken from the `Left` property (`false`).
`Right`	The `Right` property determines the Right of the PlotArea.
`RightIsDefault`	The `RightIsDefault` property determines whether the Right of the chart region is calculated by Chart (`true`) or taken from the `Right` property (`false`).
`Top`	The `Top` property determines the location of the top of the PlotArea.
`TopIsDefault`	The `TopIsDefault` property determines whether the top position of the chart region is calculated by Chart (`true`) or taken from the `Top` property (`false`).

A.27 SimpleChart

Name	Description
`AxisOrientation`	The `AxisOrientation` property determines if the x and y axis are inverted and reversed.
`Background`	The `Background` property determines the background color used to draw inside the chart region. Note that the `Background` property is inherited from the parent JCComponent.
`ChartType`	The `ChartType` property determines the chart type of the first set of data in the chart.
`Data`	The `Data` property controls the file or URL used for the first set of data in chart.

Name	Description
Font	The Font property determines what font is used to render text inside the chart region. Note that the Font property is inherited from the parent JCComponent.
FooterText	The FooterText property holds the text that is displayed in the footer. The default value is " " (empty string).
Foreground	The Foreground property determines the foreground color used to draw inside the chart region. Note that the Foreground property is inherited from the parent JCComponent.
HeaderText	The HeaderText property holds the text that is displayed in the header. The default value is " " (empty string).
Legend	The Legend property controls the object that controls the display of the legend. The default value is null.
LegendAnchor	The LegendAnchor property determines the position of the legend relative to the ChartArea. Valid values include NORTH, SOUTH, EAST, WEST, NORTHWEST, SOUTHWEST, NORTHEAST and SOUTHEAST. The default value is EAST.
LegendIsShowing	The LegendIsShowing property determines whether the legend is visible. The default value is false.
LegendOrientation	The LegendOrientation property determines how legend information is laid out. Valid values include VERTICAL and HORIZONTAL. The default value is VERTICAL.
Margins	The Margins property controls the margins (an Insets object) on the chart. The default value is 1,1,1,1.
SwingDataModel	Sets the chart’s data source to use a specified Swing TableModel object, instead of using the Data property.
View3D	The View3D property combines the values of the Depth, Elevation, and Rotation properties defined in JCChartArea. Depth controls the apparent depth of a graph. Elevation controls the distance above the X-axis for the 3D effect. Rotation controls the position of the eye relative to the Y-axis for the 3D effect. The default value is "0.0,0.0,0.0".
Version	The Version property specifies the JClass Chart version number. This property is read-only. The default value is calculated.
XAxisAnnotation-Method	The XAxisAnnotationMethod property determines how axis annotations are generated. Valid values include VALUE (annotation is generated by Chart, with possible callbacks to a label generator); VALUE_LABELS (annotation is taken from a list of value labels provided by the user — a value label is a label that appears at a particular axis value); POINT_LABELS (annotation comes from the data source's point labels that are associated with particular data points); and TIME_LABELS (Chart generates time/date labels based on the TimeUnit, TimeBase and TimeFormat properties). The default value is VALUE.
XAxisGridIsShowing	The XAxisGridIsShowing property determines whether a grid is drawn for the axis. The default value is false.

Name	Description
XAxisIsLogarithmic	The XAxisIsLogarithmic property determines whether the first x axis will be logarithmic (true) or linear (false). The default value is false.
XAxisIsShowing	The XAxisIsShowing property determines whether the first x axis is currently visible. The default value is true.
XAxisMinMax	The XAxisMinMax controls both the XAxisMin and XAxisMax properties. The XAxisMin property controls the minimum value shown on the axis. If a null string is used, Chart will calculate the axis min. The data min is determined by Chart. The default value is calculated. The XAxisMax property controls the maximum value shown on the axis. If a null string is used, Chart will calculate the axis max. The data max is determined by Chart. The default value is calculated.
XAxisNumSpacing	The XAxisNumSpacing property controls the interval between axis labels. If a null string is used, Chart will calculate the interval. The default value is calculated.
XAxisTitleText	The XAxisTitleText property specifies the text that will appear as the x axis title. The default value is " " (empty string).
YAxisAnnotation-Method	The YAxisAnnotationMethod property determines how axis annotations are generated. Valid values include VALUE (annotation is generated by Chart, with possible callbacks to a label generator); VALUE_LABELS (annotation is taken from a list of value labels provided by the user — a value label is a label that appears at a particular axis value); POINT_LABELS (annotation comes from the data source's point labels that are associated with particular data points); and TIME_LABELS (Chart generates time/date labels based on the TimeUnit, TimeBase and TimeFormat properties). The default value is VALUE.
YAxisGridIsShowing	The YAxisGridIsShowing property determines whether a grid is drawn for the axis.
YAxisIsLogarithmic	The YAxisIsLogarithmic property determines whether the first y axis will be logarithmic (true) or linear (false). The default value is false.
YAxisIsShowing	The YAxisIsShowing property determines whether the first y axis is currently visible. The default value is true.
YAxisMinMax	The YAxisMinMax controls both the YAxisMin and YAxisMax properties. The YAxisMin property controls the minimum value shown on the axis. If a null string is used, Chart will calculate the axis min. The data min is determined by Chart. The default value is calculated. The YAxisMax property controls the maximum value shown on the axis. If a null string is used, Chart will calculate the axis max. The data max is determined by Chart. The default value is calculated.
YAxisNumSpacing	The YAxisNumSpacing property controls the interval between axis labels. If a null string is used, Chart will calculate the interval. The default value is calculated.
YAxisTitleText	The YAxisTitleText property specifies the text that will appear as the Y axis title. The default value is " " (empty string).

B

JCString Properties

Most JClass Chart components support a rich text format called "JCString", which allows a mixture of hypertext, images and text within Chart components. Text can also appear in a variety of colors, fonts and styles, including underline and strikeout.

The following section describes the types of JCString properties available, and provides examples of their use.

Alignment

If a cell contains an image, the line height can, in some cases, be greater than the height of the text. Text can be aligned vertically using the `ALIGN` property. Valid values include `TOP`, `BOTTOM` and `MIDDLE`. The following example uses all three possible `ALIGN` values:

```
([IMAGE=smiley.gif][ALIGN=TOP]top
[IMAGE=smiley.gif][ALIGN=MIDDLE]middle
[IMAGE=smiley.gif][ALIGN=bottom]bottom)
```

Color

Different text colors can be specified by using the `COLOR` property. The JCString shown below displays text using red, green and blue colors.

```
([COLOR=red]Red, [COLOR=green]Green, [COLOR=blue]Blue,
[DEFAULT_COLOR]Default)
```

In addition to these colors, any color referenced in Appendix C can be used, including RGB color values.

Note: The property `DEFAULT_COLOR` resets the text color in the rest of the table to the browser's regular text color.

Fonts

Different fonts can be specified within a single cell or label by using the FONT property. The following JCString example displays text using a variety of fonts and font styles.

```
([FONT=timesroman-plain-20]TimesRoman-20,
 [FONT=timesroman-bold-12]TimesRoman-12 bold,
 [DEFAULT_FONT]Default)
```

Note: The property DEFAULT_FONT resets the fonts in the rest of the table to the browser's regular font.

Horizontal and Vertical Spacing

Vertical and horizontal spacing can be modified by using the VERT_SPACE and HORIZ_SPACE tags. VERT_SPACE offsets the current line by a number of pixels, and HORIZ_SPACE offsets the line from the margin by a set number of pixels.

The example below makes use of the HORIZ_SPACE and VERT_SPACE. tags.

```
([VERT_SPACE=10]Vertical offset=10
 [HORIZ_SPACE=10] Horizontal offset=10)
```

Hypertext

Hypertext links can be specified within a cell. The link appears underlined, and the browser display will show the target URL when the mouse cursor passes over the linked text. Hypertext uses the HREF property, and trackCursorPosition must be set to true in order for the link to work. The example below links to KL Group's home page:

```
(Click [HREF=http://www.klg.com]here[/HREF] for tech support)
```

Images

Images can be specified in a cell by using the IMAGE property. A URL or file name must be provided. If a relative path is given, the document base for the page is assumed.

The example below mixes an image with text:

```
(Tech Support: [IMAGE=http://www.klg.com/images/technical.gif])
```

Reset

The RESET property resets the font and color to the default value used by the browser. The following example changes the COLOR and FONT values back to the default:

```
([COLOR=green][FONT=timesroman-plain-20]Big text[RESET]Regular Text)
```

Strikethrough Text

Text can be crossed-out using the `STRIKEOUT` property. The following incorporates text that has been stuck through:

```
(This text is [ST]crossed-out[/ST].)
```

Underlined Text

Text can be underlined using the JCString `UNDERLINE` property. The following example incorporates underlined text:

```
(This text is [UL]underlined[/UL].)
```

C

Colors and Fonts

Colorname Values ■ RGB Color Values ■ Fonts

This section provides information on common colorname values, specific rgb color values and fonts applicable to all Java programs.

C.1 Colorname Values

The following lists all the colornames that can be used within Java programs. The majority of these colors will appear the same (or similar) across different computing platforms.

- black
- blue
- cyan
- darkGray
- darkGrey
- gray
- grey
- green
- lightGray
- lightGrey
- lightBlue
- magenta
- orange
- pink
- red
- white
- yellow

C.2 RGB Color Values

The following lists all the main RGB color values that can be used within JClass Chart. RGB color values are specified as three numeric values representing the red, green and blue color components; these values are separated by dashes ("-").

The following RGB color values describe the colors available to Unix systems. It is recommended that you test these color values in a JClass Chart program on a Windows or Macintosh system before utilizing them.

The list begins with all of the variations of white, then blacks and greys, and then describes the full color spectrum ranging from reds to violets.

Example code:

```
<PARAM NAME=backgroundList VALUE="(4, 5 255-255-0)">
```

RGB Value	**Description**
255-250-250	Snow
248-248-255	Ghost White
245-245-245	White Smoke
220-220-220	Gainsboro
255-250-240	Floral White
253-245-230	Old Lace
250-240-230	Linen
250-235-215	Antique White
255-239-213	Papaya Whip
255-235-205	Blanched Almond
255-228-196	Bisque
255-218-185	Peach Puff
255-222-173	Navajo White
255-228-181	Moccasin
255 248-220	Cornsilk
255-255-240	Ivory
255-250-205	Lemon Chiffon
255-245-238	Seashell
240-255-240	Honeydew
245-255-250	Mint Cream
240-255-255	Azure
240-248-255	Alice Blue
230-230-250	Lavender
255-240-245	Lavender Blush
255-228-225	Misty Rose
255-255-255	White
0-0-0	Black
47-79-79	Dark Slate Grey
105-105-105	Dim Gray
112- 128-144	Slate Grey
119- 136-153	Light Slate Grey
190- 190-190	Grey

211- 211-211	Light Gray
25-25-112	Midnight Blue
0-0-128	Navy Blue
100- 149 237	Cornflower Blue
72-61-139	Dark Slate Blue
106-90-205	Slate Blue
123- 104 238	Medium Slate Blue
132-112- 255	Light Slate Blue
0-0-205	Medium Blue
65-105-225	Royal Blue
0-0-255	Blue
30-144-255	Dodger Blue
0-19 -255	Deep Sky Blue
135-206-235	Sky Blue
135-206-250	Light Sky Blue
70-130-180	Steel Blue
176-196- 222	Light Steel Blue
173-216-230	Light Blue
176-224-230	Powder Blue
175-238-238	Pale Turquoise
0-206-209	Dark Turquoise
72-209-204	Medium Turquoise
64-224-208	Turquoise
0-255-255	Cyan
224-255-255	Light Cyan
95-158-160	Cadet Blue
102-205-170	Medium Aquamarine
127-255-212	Aquamarine
0-100-0	Dark Green
85-107-47	Dark Olive Green
143-188-143	Dark Sea Green
46-139-87	Sea Green
60-179-113	Medium Sea Green
32-178-170	Light Sea Green
152-251-152	Pale Green
0-255-127	Spring Green
124-252- 0	Lawn Green
0-255-0	Green
127-255- 0	Chartreuse
0-250-154	Medium Spring Green
173-255-47	Green Yellow
50-205-50	Lime Green
154-205-50	Yellow Green
34-139-34	Forest Green
107-142-35	Olive Drab
189-183-107	Dark Khaki
240-230-140	Khaki
238-232-170	Pale Goldenrod
250-250-210	Light Goldenrod Yellow

255-255-224	Light Yellow
255-255-0	Yellow
255-215-0	Gold
238-221-130	Light Goldenrod
218-165-32	Goldenrod
184-134-11	Dark Goldenrod
188-143-143	Rosy Brown
205-92-92	Indian Red
139-69-19	Saddle Brown
160-82-45	Sienna
205-133-63	Peru
222-184- 135	Burlywood
245-245-220	Beige
245-222-179	Wheat
244-164-96	SandyBrown
210-180-140	Tan
210-105-30	Chocolate
178-34-34	Firebrick
165-42-42	Brown
233-150-122	Dark Salmon
250-128-114	Salmon
255-160-122	Light Salmon
255-165- 0	Orange
255-140-0	Dark Orange
255-127-80	Coral
240-128-128	Light Coral
255-99-71	Tomato
255-69-0	Orange Red
255-0-0	Red
255-105-180	Hot Pink
255-20-147	Deep Pink
255-192-203	Pink
255-182-193	Light Pink
219-112-147	Pale Violet Red
176-48-96	Maroon
199-21-133	Medium Violet Red
208-32-144	Violet Red
255-0-255	Magenta
238-130-238	Violet
221-160-221	Plum
218-112-214	Orchid
186-85-211	Medium Orchid
153-50-204	Dark Orchid
148-0-211	Dark Violet
138-43-226	Blue Violet
160- 32-240	Purple
147-112-219	Medium Purple
216-191-216	Thistle

C.3 Fonts

There are five different font names that can be specified in any Java program. They are:

- `Courier`
- `Dialog`
- `DialogInput`
- `Helvetica`
- `TimesRoman`

Note: Font names are case-sensitive.

There are also four standard font style constants that can be used. The valid Java font style constants are:

- `bold`
- `bold+italic`
- `italic`
- `plain`

These values are strung together with dashes ("`-`") when used with the `VALUE` attribute. You must also specify a point size by adding it to other font elements. To display a text using a 12-point italic Helvetica font, use the following:

```
Helvetica-italic-12
```

All three elements (font name, font style and point size) must be used to specify a particular font display; otherwise, the default font is used.

Note: Font display may vary from system to system. If a font does not exist on a system, the default font is displayed instead.

D

HTML Property Reference

This appendix lists the syntax of JClass Chart properties when specified in an HTML file. For example, the following HTML code sets the X-axis annotation method property:

```
<PARAM NAME="xaxis.annotationMethod" VALUE="POINT_LABELS">
```

D.1 ChartDataView Properties

Java Property	HTML Syntax	Value Type
AutoLabel	*data*.autoLabel	boolean
BufferPlotData	*data*.bufferPlotData	boolean
ChartType	*data*.chartType	(enum)
Data	*data*	AppletDataSource
DataFile	dataFile, data1File, or data2File	URLDataSource, FileDataSource
FastUpdate	*data*.fastUpdate	boolean
HoleValue	*data*.holeValue	double

Java Property	HTML Syntax	Value Type
IsInverted	*data*.isInverted	boolean
IsShowing	*data*.isShowing	boolean
IsShowingInLegend	*data*.isShowingInLegend	boolean
OutlineColor	*data*.outlineColor	Color
pointLabels	dataN.pointLabels	String

Note: `data` is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named `data1`, `data2`, `data`*n*.

D.2 ChartDataViewSeries Properties

Java Property	HTML Syntax	Value Type
Fill Color	*data*.series*n*.fill.color	Color
Fill ColorIndex	*data*.series*n*.fill.colorIndex	int
Fill Image	*data*.series*n*.fill.image	Image
FirstPoint	*data*.series*n*.firstPoint	int
IsIncluded	*data*.series*n*.isIncluded	boolean
IsShowing	*data*.series*n*.isShowing	boolean
IsShowingInLegend	*data*.series*n*.isShowingInLegend	boolean
Label	*data*.series*n*.label	String
LastPoint	*data*.series*n*.lastPoint	int
Line Color	*data*.series*n*.line.color	Color
Line ColorIndex	*data*.series*n*.line.colorIndex	int
Line Width (not supported in JDK 1.0 or 1.1)	*data*.series*n*.line.width	int
Symbol Color	*data*.series*n*.symbol.color	Color
Symbol ColorIndex	*data*.series*n*.symbol.colorIndex	int
Symbol Shape	*data*.series*n*.symbol.shape	(enum)
Symbol ShapeIndex	*data*.series*n*.symbol.shapeIndex	int
Symbol Size	*data*.series*n*.symbol.size	int

Note: `data` is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named `data1`, `data2`, `data`*n*.

D.3 JCAxis X- and Y-axis Properties

Java Property	HTML Syntax	Value Type
AnnotationMethod	*[xy]axis*.annotationMethod	(enum)
AnnotationRotation	*[xy]axis*.annotationRotation	(enum)
Font	*[xy]axis*.font	Font
Foreground	*[xy]axis*.foreground	Color
Formula Constant	*[xy]axis*.formula.constant	double
Formula Multiplier	*[xy]axis*.formula.multiplier	double
Formula Originator	*[xy]axis*.formula.originator	Axis Name, eg. `xaxis1`
GridColor	*[xy]axis*.gridColor	Color
GridIsShowing	*[xy]axis*.gridIsShowing	boolean
GridSpacing	*[xy]axis*.gridSpacing	double
IsEditable	*[xy]axis*.isEditable	boolean
IsLogarithmic	*[xy]axis*.isLogarithmic	boolean
IsReversed	*[xy]axis*.isReversed	boolean
IsShowing	*[xy]axis*.isShowing	boolean
Min	*[xy]axis*.min	double
Max	*[xy]axis*.max	double
NumSpacing	*[xy]axis*.numSpacing	double
Origin	*[xy]axis*.origin	double
OriginPlacement	*[xy]axis*.originPlacement	(enum)
Placement	*[xy]axis*.placement	(enum)
PlacementAxis	*[xy]axis*.placementAxis	Axis Name, eg. `xaxis1`
PlacementLocation	*[xy]axis*.placementLocation	double
Precision	*[xy]axis*.precision	int
TickSpacing	*[xy]axis*.tickSpacing	double
TimeBase	*[xy]axis*.timeBase	Date
TimeFormat	*[xy]axis*.timeFormat	String
TimeUnit	*[xy]axis*.timeUnit	(enum)
Title Adjust	*[xy]axis*.title.adjust	(enum)
Title Background	*[xy]axis*.title.background	Color
Title BorderType	*[xy]axis*.title.borderType	(enum)

Java Property	HTML Syntax	Value Type
Title BorderWidth	*[xy]axis*.title.borderWidth	(enum)
Title Font	*[xy]axis*.title.font	Font
Title Foreground	*[xy]axis*.title.foreground	Color
Title IsShowing	*[xy]axis*.title.isShowing	boolean
Title Placement	*[xy]axis*.title.placement	(enum)
Title Rotation	*[xy]axis*.title.rotation	0, 90, 180, 270
Title Text	*[xy]axis*.title.text	JCString
ValueLabels	*[xy]axis*.valueLabels	String[] (values separated by " ;")

Note: xaxis and yaxis are the names of the first axes, generated when chart properties are saved to an HTML file; additional axes are named [xy]axis1, [xy]axis2, [xy]axis*n*.

D.4 JCBarChartFormat Properties

Java Property	HTML Syntax	Value Type
100Percent	*data*.Bar.100Percent	boolean
ClusterOverlap	*data*.Bar.clusterOverlap	int
ClusterWidth	*data*.Bar.clusterWidth	int

Note: data is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named data1, data2, data*n*.

D.5 JCCandleChartFormat Properties

Java Property	HTML Syntax	Value Type
IsComplex	*data*.Candle.isComplex	boolean

Note: data is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named data1, data2, data*n*.

D.6 JCChart/JCComponent Properties

Java Property	HTML Syntax	Value Type
AllowUserChanges	`allowUserChanges`	boolean
Background	`background`	Color
BorderType	`borderType`	(enum)
BorderWidth	`borderWidth`	int
BottomMargin	`bottomMargin`	int
CancelKey	`cancelKey`	int
CustomizeTrigger	`customizeTrigger`	(enum) (see Note for details)
DepthTrigger	`depthTrigger`	(enum) (see Note for details)
DoubleBuffer	`doubleBuffer`	boolean
EditTrigger	`editTrigger`	(enum) (see Note for details)
Font	`font`	Font
Foreground	`foreground`	Color
HighlightColor	`highlightColor`	Color
HighlightThickness	`highlightThickness`	int
Insets	`insets`	insets
IsBatched	`isBatched`	boolean
LabelName	`label`*n*	String (see Note for details)
LeftMargin	`leftMargin`	int
Offset	`offset`	int
PickTrigger	`PickTrigger`	(enum) (see Note for details)
ResetKey	`resetKey`	int
RightMargin	`rightMargin`	int
RotateTrigger	`RotateTrigger`	(enum) (see Note for details)
ShadowThickness	`shadowThickness`	int
SymbolColorIndex	`symbolColorIndex`	int
SymbolShapeIndex	`symbolShapeIndex`	int
TopMargin	`topMargin`	int

Java Property	HTML Syntax	Value Type
TranslateTrigger	`TranslateTrigger`	(enum) (see Note for details)
Traversable	`traversable`	boolean
ZoomTrigger	`ZoomTrigger`	(enum) (see Note for details)

Notes: `label`*`n`* is the number of Chart Labels when chart properties are saved to an HTML file. Valid values for any Trigger property are `NONE`, `CTRL`, `SHIFT`, `ALT` or `META` (equivalent to right-mouse-click).

D.7 JCChartArea Properties

Java Property	HTML Syntax	Value Type
AngleUnit	`chartArea.angleUnit`	(enum)
AxisBoundingBox	`chartArea.axisBoundingBox`	boolean
Background	`chartArea.background`	Color
BorderType	`chartArea.borderType`	(enum)
BorderWidth	`chartArea.borderWidth`	int
Depth	`chartArea.depth`	int
Elevation	`chartArea.elevation`	int
FastAction	`chartArea.fastAction`	boolean
Font	`chartArea.font`	Font
Foreground	`chartArea.foreground`	Color
Height	`chartArea.height`	int
HorizActionAxis	`chartArea.horizActionAxis`	Axis Name, eg. `xaxis1`
Insets	`chartArea.insets`	Insets
IsShowing	`chartArea.isShowing`	boolean
Left	`chartArea.left`	int
PlotArea Background	`chartArea.plotArea.background`	Color
PlotArea Bottom	`chartArea.plotArea.bottom`	int
PlotArea Foreground	`chartArea.plotArea.foreground`	Color
PlotArea Left	`chartArea.plotArea.left`	int
PlotArea Right	`chartArea.plotArea.right`	int

Java Property	HTML Syntax	Value Type
PlotArea Top	`chartArea.plotArea.top`	int
Rotation	`chartArea.rotation`	int
Top	`chartArea.top`	int
VertActionAxis	`chartArea.vertActionAxis`	Axis Name, eg. `xaxis1`
Width	`chartArea.width`	int

D.8 JCChartLabel Properties

Java Property	HTML Syntax	Value Type
Adjust	`label`*n*`.adjust`	(enum)
Anchor	`label`*n*`.anchor`	(enum)
AttachMethod	`label`*n*`.attachMethod`	(enum)
AttachX	`label`*n*`.attachX`	int
AttachY	`label`*n*`.attachY`	int
Background	`label`*n*`.background`	Color
BorderType	`label`*n*`.borderType`	(enum)
BorderWidth	`label`*n*`.borderWidth`	int
DataAttachX	`label`*n*`.dataAttachX`	int
DataAttachY	`label`*n*`.dataAttachY`	int
DataIndex	`label`*n*`.dataIndex`	DataIndex Name, eg. `AttachIndex1`
DataView	`label`*n*`.dataView`	ChartDataView
DwellDelay	`label`*n*`.dwellDelay`	int
Font	`label`*n*`.font`	Font
Foreground	`label`*n*`.foreground`	Color
Height	`label`*n*`.height`	int
Insets	`label`*n*`.insets`	Insets
IsDwellLabel	`label`*n*`.isDwellLabel`	boolean
IsShowing	`label`*n*`.isShowing`	boolean
Left	`label`*n*`.left`	int
OffsetX	`label`*n*`.offsetX`	int
OffsetY	`label`*n*`.offsetY`	int

Java Property	HTML Syntax	Value Type
Rotation	`label`*n*`.rotation`	(enum)
Text	`label`*n*`.text`	JCString
Top	`label`*n*`.top`	int
Width	`label`*n*`.width`	int

Note: `label1` is the name of the first Chart Label, generated when chart properties are saved to an HTML file; additional labels are named `label2`, `label3`, `label`*n*.

D.9 JCDataIndex Properties

Java Property	HTML Syntax	Value Type
DataView	`AttachIndex`*n*`.dataView`	ChartDataView
Distance	`AttachIndex`*n*`.distance`	int
Point	`AttachIndex`*n*`.point`	int
SeriesIndex	`AttachIndex`*n*`.seriesIndex`	int

Note: `AttachIndex1` is the name of first series index, generated when chart properties are saved to an HTML file; additional series are named `AttachIndex2`, `AttachIndex3`, `AttachIndex`*n*.

D.10 JCHLOCChartFormat Properties

Java Property	HTML Syntax	Value Type
IsOpenCloseFullWidth	*data*`.HLOC.isOpenCloseFullWidth`	boolean
IsShowingClose	*data*`.HLOC.isShowingClose`	boolean
IsShowingOpen	*data*`.HLOC.isShowingOpen`	boolean
Line Color	*data*`.HLOC.series`*n*`.line.color`	Color
Line Width (not supported in JDK 1.0 or 1.1)	*data*`.HLOC.series`*n*`.line.width`	int
TickSize	*data*`.HLOC.series`*n*`.tickSize`	int

Note: `data` is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named `data1`, `data2`, `data`*n*.

D.11 JCHiLoChartFormat Properties

Java Property	HTML Syntax	Value Type
Line Color	*data*.HiLo.series*n*.line.color	Color
Line Width (not supported in JDK 1.0 or 1.1)	*data*.HiLo.series*n*.line.width	int

Note: `data` is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named `data1`, `data2`, `data`*n*.

D.12 JCLegend Properties

Java Property	HTML Syntax	Value Type
Anchor	legend.anchor	(enum)
Background	legend.background	Color
BorderType	legend.borderType	(enum)
BorderWidth	legend.borderWidth	int
Font	legend.font	Font
Foreground	legend.foreground	Color
Height	legend.height	int
Insets	legend.insets	Insets
IsShowing	legend.isShowing	boolean
Left	legend.left	int
Orientation	legend.orientation	(enum)
Top	legend.top	int
Width	legend.width	int

D.13 JCPieChartFormat Properties

Java Property	HTML Syntax	Value Type
ExplodeOffset	*data*.Pie.explodeOffset	int
MinSlices	*data*.Pie.minSlices	int
Other Label	*data*.Pie.other.label	String

Java Property	HTML Syntax	Value Type
Other Fill Color	*data*.Pie.other.fill.color	Color
SortOrder	*data*.Pie.sortOrder	ASCENDING, DESCENDING
ThresholdMethod	*data*.Pie.thresholdMethod	(enum)
ThresholdValue	*data*.Pie.thresholdValue	int

Note: `data` is the name of the first dataset, generated when chart properties are saved to an HTML file; additional datasets are named `data1`, `data2`, `data`*n*.

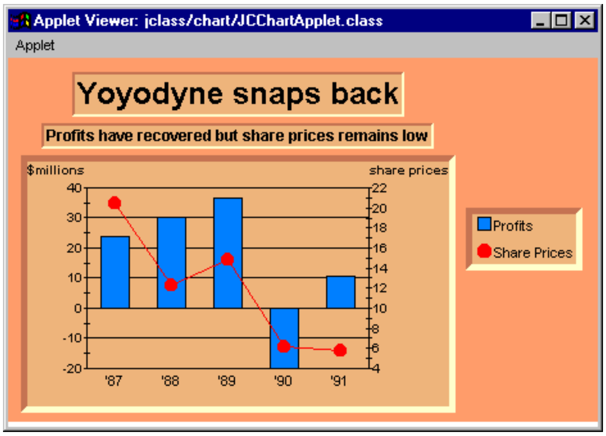
D.14 JCTitle Header and Footer Properties

Java Property	HTML Syntax	Value Type
Adjust	header.adjust footer.adjust	(enum)
Background	header.background footer.background	Color
BorderType	header.borderType footer.borderType	(enum)
BorderWidth	header.borderWidth footer.borderWidth	int
Font	header.font footer.font	Font
Foreground	header.foreground footer.foreground	Color
Height	header.height footer.height	int
Insets	header.insets footer.insets	Insets
IsShowing	header.isShowing footer.isShowing	boolean
Left	header.left footer.left	int
Rotation	header.rotation footer.rotation	(enum)
Text	header.orientation footer.orientation	JCString

Java Property	HTML Syntax	Value Type
Top	`header.top` `footer.top`	int
Width	`header.width` `footer.width`	int

D.15 Example HTML File

The following HTML file defines the chart shown below:



```
<HTML>
<HEAD>
<TITLE>Yoyodyne snaps back</TITLE>
</HEAD>
<BODY>
<CENTER><H2>Yoyodyne snaps back</H2></CENTER>
<APPLET CODE=jclass/chart/JCChartApplet.class CODEBASE="../../.." HEIGHT=472 WIDTH=580>
<PARAM NAME=background VALUE="255-158-107">
<PARAM NAME=foreground VALUE="black">
<PARAM NAME=font VALUE="Dialog-PLAIN-12">
<PARAM NAME=CustomizeTrigger VALUE="Meta">
<PARAM NAME=allowUserChanges VALUE="true">
<PARAM NAME=footer.top VALUE="85">
<PARAM NAME=footer.width VALUE="505">
<PARAM NAME=footer.borderType VALUE="In">
<PARAM NAME=footer.borderWidth VALUE="2">
<PARAM NAME=footer.font VALUE="Helvetica-BOLD-20">
<PARAM NAME=footer.background VALUE="255-175-125">
<PARAM NAME=footer.insets VALUE="0,2,1,2">
<PARAM NAME=footer.text VALUE="Profits have recovered but share prices remains low">
<PARAM NAME=footer.isShowing VALUE="true">
<PARAM NAME=header.width VALUE="375">
<PARAM NAME=header.borderType VALUE="In">
<PARAM NAME=header.borderWidth VALUE="2">
```

```
<PARAM NAME=header.font VALUE="Helvetica-BOLD-35">
<PARAM NAME=header.background VALUE="255-175-125">
<PARAM NAME=header.insets VALUE="0,2,1,2">
<PARAM NAME=header.text VALUE="Yoyodyne snaps back">
<PARAM NAME=header.isShowing VALUE="true">
<PARAM NAME=legend.top VALUE="416">
<PARAM NAME=legend.borderType VALUE="In">
<PARAM NAME=legend.borderWidth VALUE="5">
<PARAM NAME=legend.background VALUE="255-175-125">
<PARAM NAME=legend.isShowing VALUE="true">
<PARAM NAME=legend.anchor VALUE="South">
<PARAM NAME=legend.orientation VALUE="Horizontal">
<PARAM NAME=chartArea.top VALUE="130">
<PARAM NAME=chartArea.height VALUE="280">
<PARAM NAME=chartArea.width VALUE="549">
<PARAM NAME=chartArea.borderType VALUE="In">
<PARAM NAME=chartArea.borderWidth VALUE="5">
<PARAM NAME=chartArea.background VALUE="255-175-125">
<PARAM NAME=xaxis.borderWidth VALUE="3">
<PARAM NAME=xaxis.annotationMethod VALUE="Value_Labels">
<PARAM NAME=xaxis.placement VALUE="Min">
<PARAM NAME=xaxis.placementAxis VALUE="yaxis">
<PARAM NAME=xaxis.gridColor VALUE="black">
<PARAM NAME=xaxis.valueLabels VALUE="1.0; '87; 2.0; '88; 3.0; 89; 4.0; 90; 5.0; '91">
<PARAM NAME=xaxis.title.isShowing VALUE="false">
<PARAM NAME=yaxis.borderWidth VALUE="3">
<PARAM NAME=yaxis.placement VALUE="Min">
<PARAM NAME=yaxis.gridIsShowing VALUE="true">
<PARAM NAME=yaxis.gridColor VALUE="black">
<PARAM NAME=yaxis.title.font VALUE="TimesRoman-BOLD-12">
<PARAM NAME=yaxis.title.text VALUE="$millions">
<PARAM NAME=chartArea.yaxisName1 VALUE="yaxis1">
<PARAM NAME=yaxis1.borderWidth VALUE="3">
<PARAM NAME=yaxis1.placement VALUE="Max">
<PARAM NAME=yaxis1.min VALUE="4.0">
<PARAM NAME=yaxis1.max VALUE="22.0">
<PARAM NAME=yaxis1.gridColor VALUE="black">
<PARAM NAME=yaxis1.title.font VALUE="TimesRoman-BOLD-12">
<PARAM NAME=yaxis1.title.text VALUE="share prices ">
<PARAM NAME=data.chartType VALUE="Bar">
<PARAM NAME=data.outlineColor VALUE="black">
<PARAM NAME=data.series1.line.colorIndex VALUE="0">
<PARAM NAME=data.series1.fill.colorIndex VALUE="0">
<PARAM NAME=data.series1.fill.color VALUE="0-84-255">
<PARAM NAME=data.series1.fill.pattern VALUE="Per_25">
<PARAM NAME=data.series1.symbol.colorIndex VALUE="0">
<PARAM NAME=data.series1.symbol.shapeIndex VALUE="1">
<PARAM NAME=data.series1.label VALUE="Profits">
<PARAM NAME=data.Bar.clusterWidth VALUE="50">
<PARAM NAME=data VALUE="
 ARRAY ' ' 1 5
 1.0 2.0 3.0 4.0 5.0
 24.0 30.2 36.4 -19.8 10.6
 ">
<PARAM NAME=dataName1 VALUE="data1">
<PARAM NAME=data1.outlineColor VALUE="black">
<PARAM NAME=data1.series1.line.colorIndex VALUE="1">
<PARAM NAME=data1.series1.line.color VALUE="red">
<PARAM NAME=data1.series1.fill.colorIndex VALUE="1">
<PARAM NAME=data1.series1.symbol.colorIndex VALUE="1">
```

```
<PARAM NAME=data1.series1.symbol.shapeIndex VALUE="2">
<PARAM NAME=data1.series1.symbol.color VALUE="red">
<PARAM NAME=data1.series1.symbol.shape VALUE="Dot">
<PARAM NAME=data1.series1.symbol.size VALUE="10">
<PARAM NAME=data1.series1.label VALUE="Share Prices">
<PARAM NAME=data1.yaxis VALUE="yaxis1">
<PARAM NAME=data1 VALUE="
 ARRAY ' ' 1 5
 1.0 2.0 3.0 4.0 5.0
 20.5 12.3 14.8 6.2 5.75
 ">
</APPLET>
</BODY>
</HTML>
```

Index