The JFreeChart Class Library

Written by David Gilbert

January 25, 2002

© 2000-2002, Simba Management Limited. All rights reserved.

JFreeChart is free software, but this document is not free.
If you decide to use this document and you are not an open source developer,
you are required to pay a fee.

Contents

1 Introduction

1.1 What is JFreeChart?

JFreeChart is a class library for drawing charts. It is written entirely in Java, and should run on any implementation of the Java 2 platform (SDK 1.2.2 or later). JFreeChart makes use of the Java 2D API for drawing charts, so it won't work with earlier versions of Java. The latest version, plus any bug reports or other announcements, can be found at:

`http://www.jrefinery.com/jfreechart`

JFreeChart is licensed using the open-source GNU Lesser General Public Licence (LGPL). See Appendix A for details.

1.2 This Document

This document has been written for version 0.7.1 of JFreeChart.

The JFreeChart software is free, but this document is not. If you are using JFreeChart for closed-source development work, and want to use this documentation, then you are required to pay a registration fee of £30 (credit card payments are accepted via a link on the JFreeChart web page). Companies are encouraged to register multiple copies. Payment includes access to any updates to the documentation for one year from the date of registration.

You are welcome to evaluate this document before purchasing it. I rely on your honesty to pay for the documentation if you use it, as this makes life simpler for everyone. Don't be a freeloader.

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

1.3 Acknowledgements

JFreeChart contains code and ideas from many people. At the risk of missing someone out, I would like to thank the following people for their contributions: Andrzej Porebski, Bill Kelemen, David Berry, Matthew Wright, David Li, Sylvain Vieujot, Serge V. Grachov, Jonathan Nash, Hans-Jurgen Greiner, Joao Guilherme Del Valle, Mark Watson, Søren Caspersen, Laurence Vanhelsuwé, Martin Cordova, Wolfgang Irler and Craig MacFarlane.

1.4 Comments and Suggestions

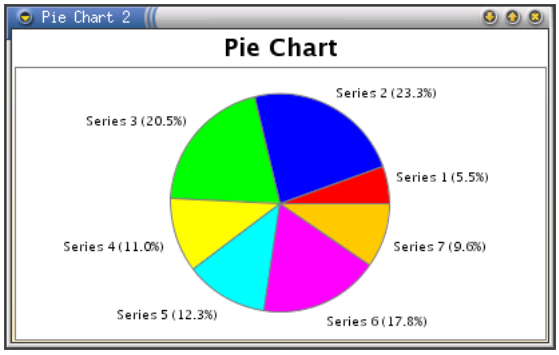
If you have any comments or suggestions regarding this document, please send e-mail to: `david.gilbert@jrefinery.com`

2 Sample Charts

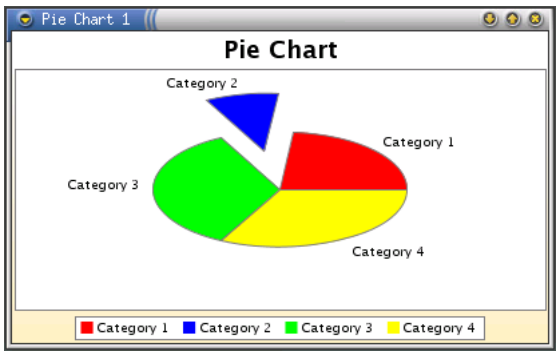
This section shows some sample charts created using the JFreeChart demonstration application. It is intended to give a reasonable overview of the types of charts that JFreeChart can generate.

2.1 Pie Charts

JFreeChart can create *pie charts* using any data that conforms to the `PieDataset` interface:



Individual pie sections can be "exploded", and the chart can take on an elliptical shape, as shown in the next example:

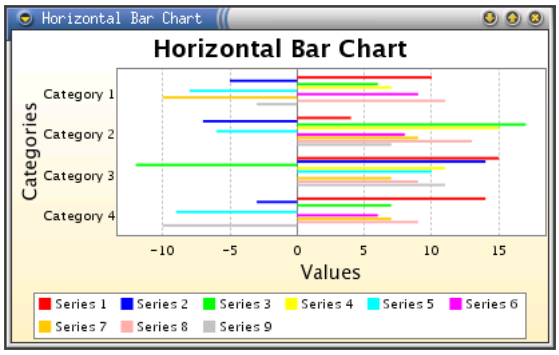


The original pie chart implementation was contributed by Andrzej Porebski.

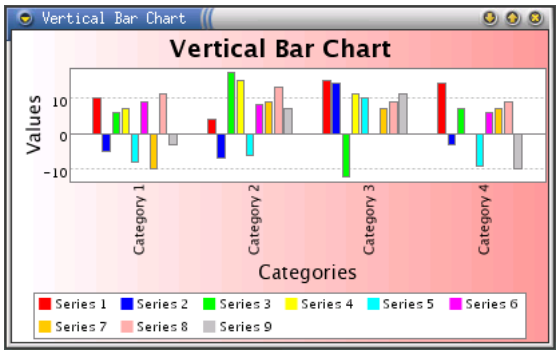
2.2 Bar Charts

A range of bar charts can be created with JFreeChart, using any data that conforms to the `CategoryDataset` interface.

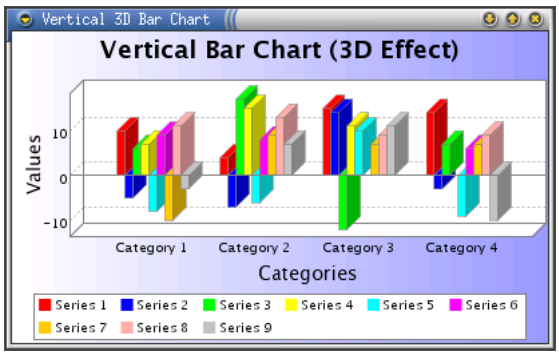
The first example is a *horizontal bar chart*:



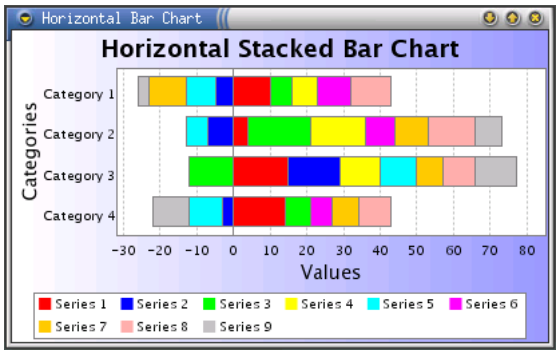
Using exactly the same data, but changing the orientation, we can generate a *vertical bar chart*:



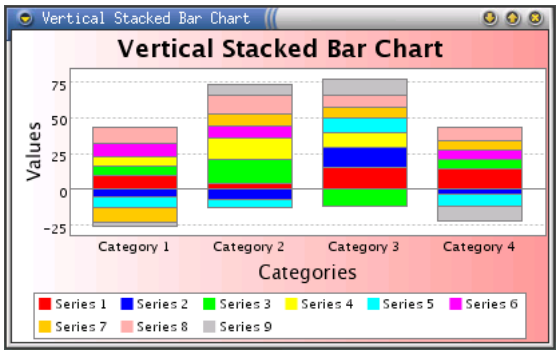
Vertical bar charts can be displayed with a 3D effect (thanks to Serge Grachov):



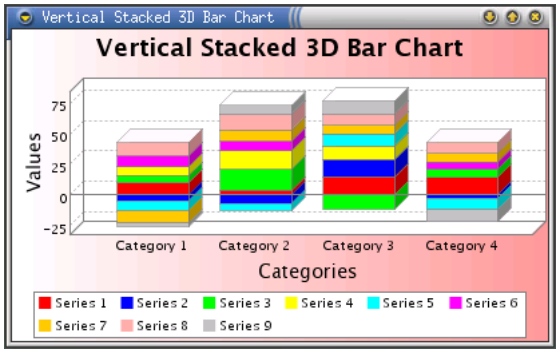
The bars can be stacked in a *stacked horizontal bar chart*:



...and similarly a *stacked vertical bar chart*:

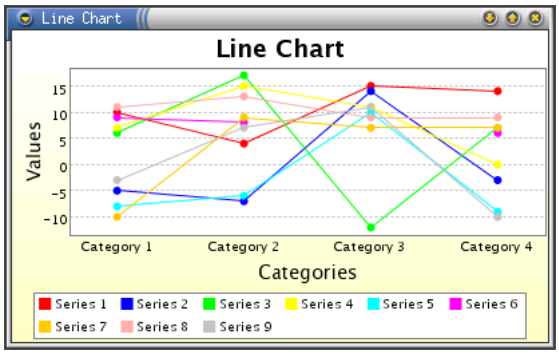


The stacked vertical bar chart can be displayed with a 3D effect (again thanks to Serge Grachov):



2.3 Line Chart

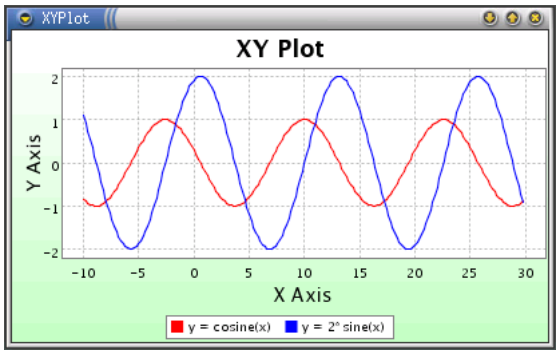
The *line chart* is generated using the same `CategoryDataset` that is used for the bar charts:



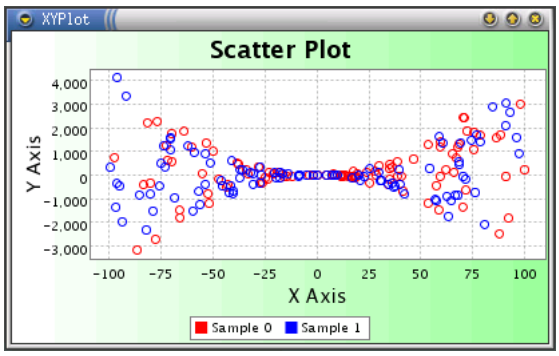
The data is the same, but the *line chart* gives you another presentation option.

2.4 XY Plots

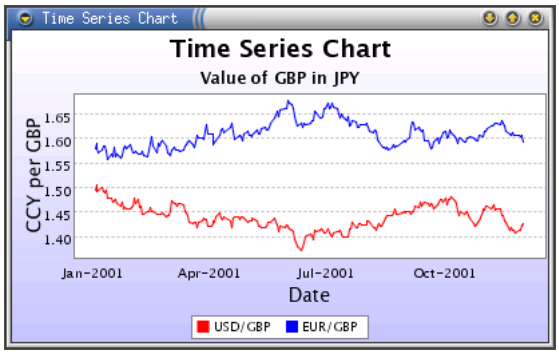
A third type of dataset, the `XYDataset`, is used to generate further chart types. The standard *XY plot* has numerical x and y axes. By default, lines are drawn between each data point:



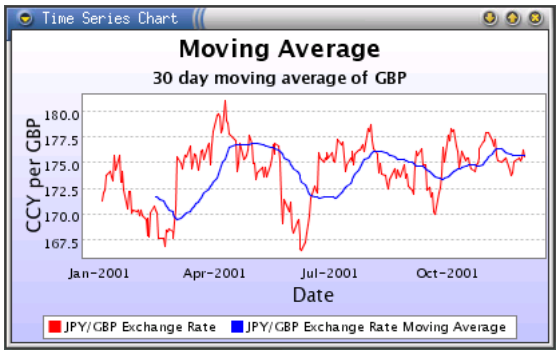
Shapes can be drawn at data points, rather than drawing lines between data points, for a *scatter plot*:



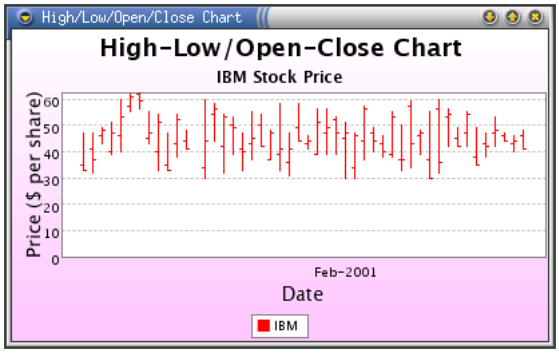
JFreeChart supports *time series charts*:



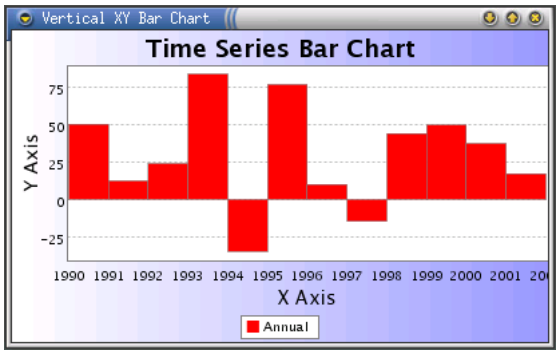
It is possible to add a moving average line to a time series plot:



You can display *high-low-open-close* data (thanks to Andrzej Porebski), using `HighLowDataset` (an extension of `XYDataset`):

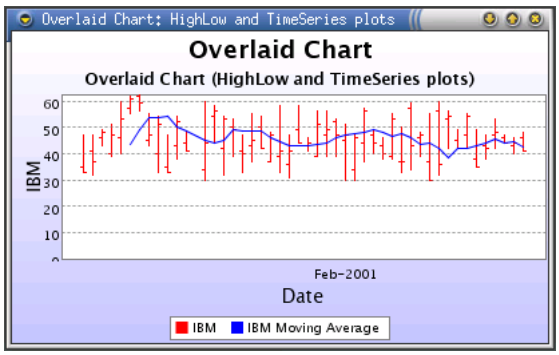


Bar charts over a numerical domain can be drawn using `IntervalXYDataset` (another extension of `XYDataset`):

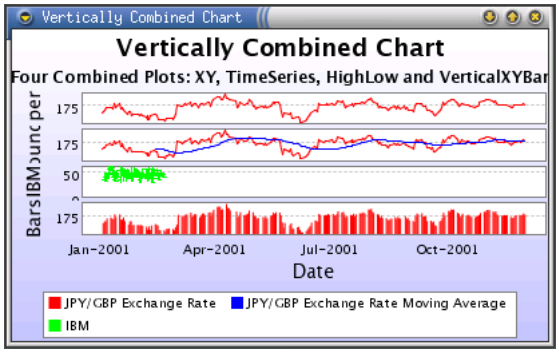


2.5 Combined Charts

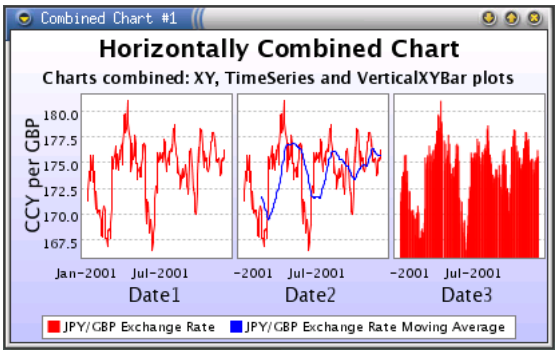
Bill Kelemen has extended JFreeChart to allow for combined charts, including overlaid charts:



...horizontally combined charts:



...and vertically combined charts:



2.6 Future Development

Given the open development model of JFreeChart, it is likely that many more chart types will be developed in the future as developers modify JFreeChart to meet their requirements. Check the JFreeChart web-page for updates:

`http://www.jrefinery.com/jfreechart`

3 Downloading and Installing JFreeChart

3.1 Download

You can download the latest version of JFreeChart from:

```
http://www.jrefinery.com/jfreechart
```

There are two versions of the JFreeChart download:

File:	Description:
`jfreechart-0.7.1.tar.gz`	JFreeChart for Linux/Unix.
`jfreechart-0.7.1.zip`	JFreeChart for Windows.

The two files contain the same source code. All the text files in the Windows download have been recoded into DOS format (both carriage return *and* linefeed at the end of each line).

JFreeChart uses the JCommon Class Library (currently version `0.5.3`). The JCommon runtime jar file is included in the JFreeChart download, but if you require the source code (recommended) then you should also download JCommon from:

```
http://www.jrefinery.com/jcommon
```

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

3.2 Unpacking the Files

After downloading JFreeChart, you need to unpack the files. You should move the download file to a convenient directory—when you unpack JFreeChart, a new subdirectory will be created in the same location as the download file.

3.2.1 Unpacking on Linux/Unix

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

```
tar xvzf jfreechart-0.7.1.tar.gz
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-0.7.1`.

3.2.2 Unpacking on Windows

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

```
jar -xvf jfreechart-0.7.1.zip
```

This will extract all the source, run-time and documentation files for JFreeChart into a new directory called `jfreechart-0.7.1`.

3.2.3 The Files

The top-level directory (`jfreechart-0.7.1`) contains two files and four sub-directories, as described in the following table:

File/Directory:	Description:
`jars`	A directory containing the JFreeChart and JCommon run-time jar files.
`javadoc`	A directory containing the Javadoc HTML files.
`licence-LGPL.txt`	The licence for JFreeChart.
`README`	Important information - *read this first!*
`servlet`	A directory containing files required for the servlet demonstration.
`source`	A directory containing the source code for JFreeChart.

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

3.3 Running the Demonstration Application

A demonstration application is included with JFreeChart, to give you some idea of what the class library can do. It is not necessary to recompile the library to run the demonstration application. All the classes are precompiled in the jar files.

To run the demo, type the following command[1] all on one line:

```
java -classpath jfreechart-0.7.1.jar:jcommon-0.5.3.jar
com.jrefinery.chart.demo.JFreeChartDemo
```

Depending on your system setup, you may need to specify the full path for the `java` executable. You may also need to type the full path to the JFreeChart and JCommon jar files.

3.4 Compiling the Source

You can recompile the source files (contained in the `source` folder) using the `javac` tool, although it's recommended that you set up a project in your favourite development environment.

At the command line, change to the source directory, then type the following command:

```
javac -g:none -O -verbose -classpath .:../jars/jcommon-0.5.3.jar
com/jrefinery/chart/demo/JFreeChartDemo.java
```

This compiles the demonstration application *and all the JFreeChart classes* (`javac` compiles each class for which it cannot find a `.class` file provided that it can find the corresponding `.java` source file). The class files are written to the same directories as the source files.

[1] If you are using Windows, you should use a semi-colon rather than a colon to separate the jar files.

3.5 Generating the Javadoc Documentation

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

To generate the documentation, enter the following command:

```
javadoc -sourcepath <your-source-directory> -d <your-output-directory>
com.jrefinery.chart com.jrefinery.chart.event com.jrefinery.chart.ui
```

4 Developing with JFreeChart

This section presents a tutorial on how to use the *JFreeChart* class library in your own projects.

4.1 Overview

The `JFreeChart` class coordinates the entire process of drawing charts. One method:

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

...instructs the `JFreeChart` object to draw a chart onto a specific area on a *graphics device*.[2]

In broad terms, `JFreeChart` achieves this by obtaining data from a `Dataset`, and delegating the drawing to a `Plot` object.

The `JFreeChart` class can work with many different dataset implementations, and even more `Plot` subclasses. The following table summarises the combinations that are currently available:

Dataset:	**Compatible Plot Types:**
`PieDataset`	`PiePlot`.
`CategoryDataset`	`HorizontalBarPlot`, `VerticalBarPlot`, `LinePlot`, `StackedHorizontalBarPlot`, `StackedVerticalBarPlot`.
`XYDataset`	`XYPlot` with various renderers.
`IntervalXYDataset`	`VerticalXYBarPlot`.
`HighLowDataset`	`XYPlot` with a `HighLowRenderer`.
`CandleStickDataset`	`XYPlot` with a `CandleStickRenderer`.

There are a lot of combinations, but don't worry, just keep in mind that a chart usually has one `Dataset` and one `Plot`.

4.2 Creating Your First Chart

To illustrate, let's create a *pie chart* using JFreeChart. First, we need to create a dataset. The `DefaultPieDataset` class in the JCommon Class Library is designed just for this purpose:

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

Next, we need to create a chart. A convenient way to do this in JFreeChart is to use the `ChartFactory` class:

```
// create a chart...
JFreeChart chart = ChartFactory.createPieChart("Sample Pie Chart", data, true);
```

[2] Java supports several graphics devices—including the screen, the printer, and buffered images—via different implementations of `java.awt.Graphics2D`. Thanks to this abstraction, JFreeChart can generate charts on any of these target devices, as well as others implemented by third parties (for example, the SVG Generator of the Batik Project).

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

Now we have a chart, but we don't yet have anywhere to draw it. Let's create a frame to display the chart in. The `JFreeChartFrame` class already knows how to display charts:

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

And that's all there is to it...here is the complete program, so that you know which packages you need to import:

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

import com.jrefinery.data.DefaultPieDataset;
import com.jrefinery.chart.ChartFactory;
import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.JFreeChartFrame;

public class First {

    public static void main(String[] args) {

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

        // create a chart...
        JFreeChart chart = ChartFactory.createPieChart("Sample Pie Chart", data, true);

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

    }

}
```

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

4.3 More about Datasets

In the previous section, we used the `DefaultPieDataset` class to supply data for our chart. JFreeChart can work with this class, because it implements the `PieDataset` interface. Take a look at this interface now, by looking at the source code[3] or the Javadoc HTML pages for the JCommon Class Library, or in the reference section towards the end of this document.

All of the datasets used by JFreeChart are defined by interfaces. This allows you to implement your own dataset using whatever data structures make sense for your own project. Of course, there are default classes available (in the JCommon Class Library) that implement each of the interfaces used by JFreeChart. You are free to use these default implementations.

[3]One of the many advantages of open source software is that you can always refer to the source code to find out how things work.

4.4 Customising Charts

After you create a chart, you may want to change some of its attributes. In this section I describe some common chart customisations.

4.4.1 Adding Chart Titles

Charts are created with only one title (or sometimes no title at all). It is relatively simple, however, to add more titles and subtitles to the chart, using the following method in the `JFreeChart` class:

```
public void addTitle(AbstractTitle title);
```
Adds a title to the chart.

The title is some concrete subclass of `AbstractTitle`, for example `TextTitle`. The placement of the title at the top, bottom, left or right of the chart is controlled by a property of the title itself.

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

4.4.2 Modifying Chart Titles

To modify a title that has already been added to a chart, you can make use of the following methods in the JFreeChart class:

```
public AbstractTitle getTitle(int index);
```
Returns a title from the chart's list of titles. Note that the index is zero-based.

```
public List getTitles();
```
Returns the complete list of titles for the chart.

You will need to cast the title to an appropriate class before you can change its properties.

4.4.3 Changing Plot Properties

You can change many properties associated with the chart's plot and axes. First, you need to get a reference to the `Plot` using the following method in the `JFreeChart` class:

```
public Plot getPlot();
```
Returns a reference to the chart's plot.

Properties that are common to all plot types can be changed easily. For example, to change the colors used for the series in a plot:

```
Plot plot = myChart.getPlot();
plot.setSeriesPaint(myPaintArray);
```

Some properties can only be changed after you have cast the result of the `getPlot()` method to an appropriate subclass of `Plot`. For example, if you want to set the gaps between the bars in a bar plot, you will need to use something like this:

```
BarPlot plot = (BarPlot)myChart.getPlot();
plot.setCategoryGapsPercent(0.10);
```

Refer to the documentation for the individual `Plot` subclasses for more information about the properties that you can change.

4.4.4 Changing Axis Properties

You can change the properties of the axes by getting a reference to the axis from the `Plot` as follows:

```
Plot plot = myChart.getPlot();
Axis hAxis = plot.getHorizontalAxis();
hAxis.setLabel("Categories");
hAxis.setLabelFont(someFont);
```

As with the plot, some properties can only be changed after you have cast the result of the `getHorizontalAxis()` method to some appropriate subclass of `Axis`.

5 Special Topics

This section presents special topics related using the *JFreeChart* class library in your own projects.

5.1 Scalable Vector Graphics

5.1.1 Introduction

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

Thanks to some excellent work by developers in The Apache XML Project, you can use the Batik SVG Toolkit to write charts, created using `JFreeChart`, to SVG format. The Batik SVG Toolkit is an open source toolkit written in Java, available from:

`http://xml.apache.org/batik`

At the time of writing, the latest version of Batik is `1.1.1`.

5.1.2 Batik and JFreeChart

Getting JFreeChart to work with Batik is relatively painless. I've only spent a limited amount of time working with Batik, so I'm no expert, but here I will describe a simple program that creates a chart and saves it in SVG format in a file. Hopefully this will be enough to get you started.

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

Next, create a project in your favourite Java development environment, and type in the following program:

```
package svgtest;

import com.jrefinery.data.DefaultPieDataset;
import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.ChartFactory;

import java.awt.geom.Rectangle2D;
import java.io.*;

import org.apache.batik.svggen.SVGGraphics2D;
import org.apache.batik.dom.GenericDOMImplementation;
import org.w3c.dom.Document;
import org.w3c.dom.DOMImplementation;

public class Application {

    public static void main(String[] args) throws IOException {

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

        // create a chart
        JFreeChart chart = ChartFactory.createPieChart("Sample Pie Chart", data, true);

        // THE FOLLOWING CODE BASED ON THE EXAMPLE IN THE BATIK DOCUMENTATION...
```

```
        // Get a DOMImplementation
        DOMImplementation domImpl = GenericDOMImplementation.getDOMImplementation();

        // Create an instance of org.w3c.dom.Document
        Document document = domImpl.createDocument(null, "svg", null);

        // Create an instance of the SVG Generator
        SVGGraphics2D svgGenerator = new SVGGraphics2D(document);

        // Ask the chart to render into the SVG Graphics2D implementation
        chart.draw(svgGenerator, new Rectangle2D.Double(0, 0, 400, 300));

        // Finally, stream out SVG to a file using UTF-8
        // character to byte encoding
        boolean useCSS = true; // we want to use CSS style attribute
        Writer out = new OutputStreamWriter(new FileOutputStream(new File("test.svg")),
                                            "UTF-8");
        svgGenerator.stream(out, useCSS);

    }

}
```

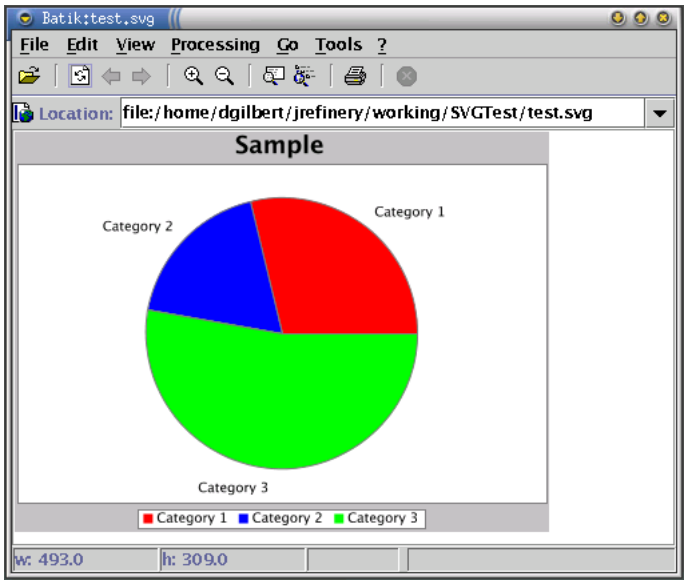
To compile the program, you need to ensure that the following jar files are on your classpath:

File:	Description:
`jcommon-0.5.3.jar`	Common classes from JRefinery.
`jfreechart-0.7.1.jar`	The JFreeChart class library.
`batik-awt-util.jar`	Batik runtime files.
`batik-dom.jar`	Batik runtime files.
`batik-ext.jar`	Batik runtime files.
`batik-svggen.jar`	Batik runtime files.
`batik-util.jar`	Batik runtime files.
`batik-xml.jar`	Batik runtime files.

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

5.1.3 Viewing the SVG

Batik includes a viewer application which you can use to open the SVG file:



If you play about with the viewer, zooming in and out and transforming the chart, you will begin to appreciate the power of the SVG format.

6 Package Overview

The following sections contain reference information for the packages that make up JFreeChart.

Package:	Description:
com.jrefinery.chart	The main chart classes.
com.jrefinery.chart.combination	Combination charts.
com.jrefinery.chart.data	Some data fitting classes (to be moved).
com.jrefinery.chart.event	The event classes.
com.jrefinery.chart.tooltips	The tooltip classes.
com.jrefinery.chart.ui	User interface classes.
com.jrefinery.chart.demo	The demonstration application.

Additional information can be found in the Javadoc-generated HTML documentation for JFreeChart.

7 Package: com.jrefinery.chart

This package contains the major classes and interfaces in the JFreeChart class library.

7.1 AbstractTitle

7.1.1 Overview

The base class for all chart titles. Several concrete sub-classes have been implemented, including: `TextTitle`, `DateTitle` and `ImageTitle`.

The `JFreeChart` class maintains a list of titles, which can hold zero, one or many titles.

7.1.2 Constructors

The default constructor:

`protected AbstractTitle(int position, int horizontalAlignment, int verticalAlignment, Insets insets);`
Creates a new `AbstractTitle`.

7.1.3 Notes

The original version of this class was written by David Berry. I've since made a few changes to the original version, but the idea for allowing a chart to have multiple titles came from David.

This class implements `Cloneable`, which is useful when editing title properties because you can edit a copy of the original, and then either apply the changes or cancel the changes.

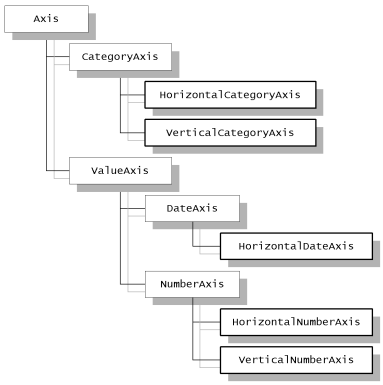
See Also
`ImageTitle`, `TextTitle`.

7.2 Axis

7.2.1 Overview

An abstract class representing an axis (horizontal or vertical). The `Plot` class is responsible for maintaining references to axes.

The following diagram shows all the subclasses of `Axis`:



7.2.2 Constructors

To create a new `Axis`:

`protected Axis(String label);`
Creates a new `Axis`, with the specified label.

7.2.3 Attributes

The `Axis` class has the following attributes:

Attribute:	Description:
`Plot`	The plot that the axis belongs to.
`Label`	The axis label.
`LabelFont`	The font for the axis label.
`LabelPaint`	The color for the axis label.
`LabelInsets`	The space to leave blank around the axis label.
`TickLabelsVisible`	A flag controlling the visibility of tick labels.
`TickLabelFont`	The font for the tick labels.
`TickLabelPaint`	The color for the tick labels.
`TickLabelInsets`	The space to leave around the tick labels.
`TickMarksVisible`	A flag controlling the visibility of tick marks.
`TickMarkStroke`	The *stroke* used to draw the tick marks.

The following default values are used for attributes wherever necessary:

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

7.2.4 Notes

The `Axis` class implements a notification mechanism that informs registered listeners that a change has been made to an axis. The following methods are used:

`public void addChangeListener(AxisChangeListener listener);`
Registers an object to receive notification whenever the axis changes.

`public void removeChangeListener(AxisChangeListener listener);`
Deregisters an object, so that it no longer receives notification when the axis changes.

`public void notifyListeners(AxisChangeEvent event);`
Notifies all registered listeners that a change has been made to the axis.

The axis class also implements a utility method for drawing vertical text[4]:

`protected void drawVerticalString(String text, Graphics2D g2, float x, float y);`
Draws text at the specified location, running from the bottom to the top of the screen. This method is used to draw axis labels.

See Also
`AxisChangeEvent`, `AxisChangeListener`, `AxisNotCompatibleException`.

7.3 AxisNotCompatibleException

7.3.1 Overview

An exception that indicates that an attempt has been made to assign an axis to a `Plot` where the axis is not compatible with the plot type (for example, a `VerticalCategoryAxis` will not work with an `XYPlot`).

7.3.2 Constructors

To create a new exception:

`public AxisNotCompatibleException(String message);`
Creates a new exception.

7.3.3 Notes

The `AxisNotCompatibleException` is a subclass of `RuntimeException`.

See Also
`PlotNotCompatibleException`.

7.4 Bar

7.4.1 Overview

A utility class for representing a bar in a `BarPlot`.

7.4.2 Notes

Used temporarily during the drawing process only.

[4]This method may get moved elsewhere as it has uses beyond drawing labels for axes.

See Also

`BarPlot`.

7.5 BarPlot

7.5.1 Overview

A base class for generating bar plots—extends `Plot` and implements `CategoryPlot`. In the current implementation, there are two concrete subclasses: `HorizontalBarPlot` and `VerticalBarPlot`.

This class implements the `CategoryPlot` interface, which enables the category axis to query the bar plot for the positioning of categories.

7.5.2 Constructors

There are two constructors for `BarPlot`—the first requires all properties to be specified, the second assumes default values for most properties. Refer to the Javadoc or the source code for details.

```
protected BarPlot(Axis horizontalAxis, Axis verticalAxis);
```

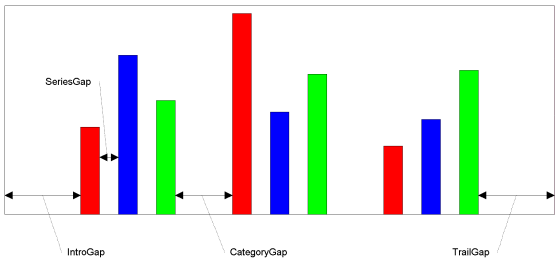
Creates a new `BarPlot`.

7.5.3 Attributes

The `BarPlot` class has the following attributes:

Attribute:	Description:
`IntroGapPercent`	The space *before* the first bar.
`TrailGapPercent`	The space *after* the last bar.
`CategoryGapsPercent`	The space between the last bar in one category, and the first bar in the next category.
`ItemGapsPercent`	The space between two bars in the same category.
`ToolTipGenerator`	The tooltip generator (optional).

This diagram illustrates the purpose of the "gap" attributes:



The following default values are used for attributes wherever necessary:

Name:	Value:
`DEFAULT_INTRO_GAP_PERCENT`	`0.05` (5 percent)
`DEFAULT_TRAIL_GAP_PERCENT`	`0.05` (5 percent)
`DEFAULT_CATEGORY_GAPS_PERCENT`	`0.20` (20 percent)
`DEFAULT_ITEM_GAPS_PERCENT`	`0.15` (15 percent)

7.5.4 Notes

The `BarPlot` subclasses are tooltip enabled.

See Also
`HorizontalBarPlot`, `VerticalBarPlot`.

7.6 CandlestickRenderer

7.6.1 Overview

Renders data for an `XYPlot` in the form of *candlesticks*. Implements the `XYItemRenderer` interface.

7.6.2 Constructors

To create a new renderer:

```
public CandlestickRenderer(double candleWidth);
```
Creates a new renderer.

7.6.3 Notes

This renderer requires a `HighLowDataset`.

See Also
`XYItemRenderer`.

7.7 CategoryAxis

7.7.1 Overview

An abstract base class for axes that display labels for categorical data.

`CategoryAxis` extends `Axis`. Known subclasses include `HorizontalCategoryAxis` and `VerticalCategoryAxis`.

7.7.2 Notes

Note that this class doesn't add anything to `Axis`—it occupies its place in the class hierarchy purely for descriptive purposes.

See Also
`Axis`.

7.8 CategoryPlot

7.8.1 Overview

An interface that should be implemented by any subclass of `Plot` that displays categorical data. It allows the axis to determine the position of each category, so that it can place the category labels correctly. Plots can vary in the way they distribute categories along an axis.

7.8.2 Methods

The interface defines the following methods:

`public List getCategories();`
Returns an ordered list of the categories

`public double getCategoryCoordinate(int category, Rectangle2D area);`
Returns the coordinate (horizontal or vertical depending on the plot) of the center of the specified category.

`public CategoryDataset getDataset();`
A convenience method that returns the dataset as a `CategoryDataset`.

7.8.3 Notes

The `CategoryDataset` interface is part of the JCommon Class Library.

See Also
`CategoryAxis`.

7.9 ChartFactory

7.9.1 Overview

This class provides a range of static methods for constructing charts. These methods make it easier to create charts with default properties.

7.9.2 Methods

```
public static JFreeChart createPieChart(String title, PieDataset data,
boolean legend);
```
Creates a *pie chart* for the given `PieDataset`.

```
public static JFreeChart createVerticalBarChart(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```
Creates a *vertical bar chart* for the given `CategoryDataset`.

```
public static JFreeChart createVerticalBarChart3D(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```
Creates a *vertical bar chart with 3D effect* for the given `CategoryDataset`.

```
public static JFreeChart createStackedVerticalBarChart(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```
Creates a *stacked vertical bar chart* for the given `CategoryDataset`.

```
public static JFreeChart createStackedVerticalBarChart3D(String title,
String categoryAxisLabel, String valueAxisLabel, CategoryDataset data,
boolean legend);
```
Creates a *stacked vertical bar chart with 3D effect* for the given `CategoryDataset`.

`public static JFreeChart createHorizontalBarChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset data, boolean legend);`
Creates a *horizontal bar chart* for the given `CategoryDataset`.

`public static JFreeChart createStackedHorizontalBarChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset data, boolean legend);`
Creates a *stacked horizontal bar chart* for the given `CategoryDataset`.

`public static JFreeChart createLineChart(String title, String categoryAxisLabel, String valueAxisLabel, CategoryDataset data, boolean legend);`
Creates a *line chart* for the given `CategoryDataset`.

`public static JFreeChart createXYChart(String title, String xAxisLabel, String yAxisLabel, XYDataset data, boolean legend)`
Creates an *XY plot* for the given `XYDataset`.

`public static JFreeChart createScatterPlot(String title, String xAxisLabel, String yAxisLabel, XYDataset data, boolean legend)`
Creates a *scatter plot* for the given `XYDataset`.

`public static JFreeChart createTimeSeriesChart(String title, String timeAxisLabel, String valueAxisLabel, XYDataset data, boolean legend)`
Creates a *time series chart* for the given `XYDataset`.

`public static JFreeChart createVerticalXYBarChart(String title, String xAxisLabel, String yAxisLabel, IntervalXYDataset data, boolean legend)`
Creates a *vertical XY bar chart* for the given `IntervalXYDataset`.

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

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

7.9.3 Notes

These methods are provided for convenience only. You are not required to use them.

See Also
`JFreeChart`.

7.10 ChartUtilities

7.10.1 Overview

This class contains some useful methods for use with charts.

7.10.2 Methods

The methods include:

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

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

7.10.3 Notes

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

See Also
`JFreeChart`.

7.11 DateAxis

7.11.1 Overview

The base class for axes that display date/time values—extends `Axis`. This class is designed to be flexible about the range of dates/times that it can display—anything from several milliseconds to several decades should be handled.

7.11.2 Constructors

This class has two constructors—the first requires all properties to be specified, while the second assumes default values for many properties.

7.11.3 Attributes

`DateAxis` defines the following properties:

Attribute:	Description:
`minimumDate`	The minimum date (or time) visible on the axis.
`maximumDate`	The maximum date (or time) visible on the axis.
`tickUnits`	The `DateUnit` used for tick marks.
`tickLabelFormatter`	The `DateFormat` object used to format the tick labels.

7.11.4 Notes

In the current implementation, there is one subclass: `HorizontalDateAxis`, which can be used with an `XYPlot` to present time series data.

See Also
`HorizontalDateAxis`, `DateUnit`.

7.12 DateTitle

7.12.1 Overview

A chart title that displays the current date. Since charts can have multiple titles, this class enables the current date to be added in various positions relative to the chart (often at the bottom).

7.12.2 Notes

The original version of this class was written by David Berry (`dberry@dallas.net`).

See Also

`AbstractTitle`.

7.13 DateUnit

7.13.1 Overview

Represents a fixed unit of time. This class is used to specify the tick units on a `DateAxis`.

7.13.2 Constructors

The constructor:

```
public DateUnit(int field, int count);
```

Creates a new `DateUnit`.

The `field` attribute is one of the following:

Time Unit:	Constant:
Year	`Calendar.YEAR`
Month	`Calendar.MONTH`
Day	`Calendar.DATE`
Hour	`Calendar.HOUR_OF_DAY`
Minute	`Calendar.MINUTE`
Second	`Calendar.SECOND`
Millisecond	`Calendar.MILLISECOND`

7.13.3 Notes

Refer to the Javadoc HTML files and source code for details.

See Also

`DateAxis`.

7.14 HighLow

7.14.1 Overview

Not yet documented.

7.14.2 Notes

Refer to Javadoc HTML files and source code for details.

7.15 HighLowRenderer

7.15.1 Overview

A renderer that can be used with the `XYPlot` class and a `HighLowDataset` to create high-low-open-close charts.

7.15.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`XYPlot`.

7.16 HorizontalAxis

7.16.1 Overview

An interface that *must* be implemented by all horizontal axes. The methods defined by this interface are used by the `Plot` that owns the axis, for layout purposes.

7.16.2 Methods

The interface defines two methods. The plot will call one of these two methods, depending on the implementation.

```
public Rectangle2D reserveAxisArea(Graphics2D g2,
Plot plot, Rectangle2D drawArea, double reservedWidth);
```
Calculates the area that the horizontal axis requires to draw itself. If this method is used, it will be called after the vertical axis has determined the width that it requires—the argument `reservedWidth` contains this value.

```
public double reserveHeight(Graphics2D g2, Plot plot,
Rectangle2D drawArea);
```
Estimates the height that the horizontal axis requires to draw itself. If this method is used, it will be called before the vertical axis is asked to calculate the area that it requires—the height returned by this method will be passed to the vertical axis.

See Also
`VerticalAxis`.

7.17 HorizontalBarPlot

7.17.1 Overview

This plot draws a bar chart using data from a `CategoryDataset`, where the categories are plotted against the vertical axis and the numerical data is plotted against the vertical axis.

7.17.2 Constructors

This class provides two constructors—one that requires all the attributes for the plot to be specified, the other assumes a number of default values. Refer to the Javadoc or the source code for details.

7.17.3 Methods

Some notes on the methods for `HorizontalBarPlot`:

> `public double getCategoryCoordinate(...);`
> This method returns the y-coordinate of the center of the specified category. The category axis will call this method to determine where to place the category labels, because it has no knowledge of the distribution of categories (these could vary, depending on the nature of the plot).

See Also
`BarPlot`, `HorizontalValuePlot`, `VerticalBarPlot`.

7.18 HorizontalBarRenderer

7.18.1 Overview

Not yet documented.

7.18.2 Notes

Refer to Javadoc HTML files and source code for details.

7.19 HorizontalCategoryAxis

7.19.1 Overview

A horizontal axis that displays labels for categorical data. This class extends `CategoryAxis` and implements `HorizontalAxis`.

7.19.2 Constructors

There are two constructors defined, one that sets up the axis with mostly default properties, and another that requires the caller to specify all the properties for the axis. Refer to the Javadoc or the source code for details.

> `public HorizontalCategoryAxis(String label);`
> Creates a new axis, using default values where necessary.

7.19.3 Attributes

The axis can display category labels with a horizontal or vertical orientation—this is controlled by the `VerticalCategoryLabels` attribute. The remaining properties for this class are inherited from `CategoryAxis`.

7.19.4 Notes

In the current implementation, this class can be used with `LinePlot` and `VerticalBarPlot`.

This class relies on the `Plot` to implement the `CategoryPlot` interface. This is because the axis has no control over the visual presentation of the data—in particular, the axis cannot know how the categories are to be distributed along the axis, so it must query the `Plot` via the defined interface.

See Also
`CategoryAxis`, `VerticalCategoryAxis`.

7.20 HorizontalCategoryItemRenderer

7.20.1 Overview

Not yet documented.

7.20.2 Notes

Refer to Javadoc HTML files and source code for details.

7.21 HorizontalDateAxis

7.21.1 Overview

An axis that displays numerical data in *date format*—this class extends `DateAxis` and implements `HorizontalAxis`.

7.21.2 Attributes

The axis can display category labels with a horizontal or vertical orientation—this is controlled by the `verticalTickLabels` property.

The remaining properties for this class are inherited from `DateAxis`. Although the axis displays dates for tick labels, it is still working with `Number` objects. The numbers are interpreted as the number of milliseconds since 1 January 1970 (that is, the encoding used by `java.util.Date`).

See Also
`DateAxis`.

7.22 HorizontalNumberAxis

7.22.1 Overview

An horizontal axis that displays numerical data—this class extends `NumberAxis` and implements `HorizontalAxis`.

7.22.2 Constructors

There are three constructors for this class. One requires the caller to specify all the axis properties, while the other two use some default properties. Refer to the Javadoc or the source code for details.

7.22.3 Methods

Some notes on the methods in `HorizontalNumberAxis`:

`public void autoAdjustRange();`
Obtains the minimum and maximum data values from the `Plot`, provided that it implements `HorizontalValueRange`, and adjusts the axis range accordingly. Note that the `autoRangeIncludesZero` flag is checked in this method.

`public void refreshTicks(...);`
A utility method for calculating the positions of the ticks on an axis, just prior to drawing the axis. This method checks the `autoTickUnits` flag, and automatically determines a suitable "standard" tick size if required.

7.22.4 Notes

Refer to the Javadoc HTML files and the source code for details.

See Also
`NumberAxis`, `HorizontalNumberAxis`.

7.23 HorizontalValuePlot

7.23.1 Overview

An interface that returns the minimum and maximum values in the "horizontal direction" for a two-dimensional plot. The values could be from the dataset's domain or range, depending on the orientation of the plot.

This interface is known to be implemented by `HorizontalBarPlot`.

7.23.2 Methods

This interface has two methods:

`public Number getMinimumHorizontalDataValue();`
Returns the minimum data value in the horizontal direction for the plot;

`public Number getMaximumHorizontalDataValue();`
Returns the maximum data value in the horizontal direction for the plot;

7.23.3 Notes

Refer to the Javadoc HTML files and source code for details.

See Also
`VerticalValuePlot`.

7.24 ImageTitle

7.24.1 Overview

A chart title that displays an image.

7.24.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`AbstractTitle`.

7.25 JFreeChart

7.25.1 Overview

The `JFreeChart` class is central to the entire chart generation process. This class co-ordinates a collection of other classes with the aim of producing attractive charts in a variety of situations.

JFreeChart has been designed to draw charts onto a Java 2D graphics device (`java.awt.Graphics2D`) which means that charts can be drawn on any device supported by Java. Usually, developers are interested in drawing charts on the screen, but you have the option to also output charts to the printer, an offscreen image buffer, a scalable vector graphics (SVG) generator or whatever. Thanks to `Graphics2D` the same drawing code is used in all cases.

7.25.2 Constructors

All constructors require you to supply a `Dataset` and a `Plot` instance. You need to make sure that the plot and the dataset are compatible (refer to the plot's documentation for details).

The simplest constructor is:

> `public JFreeChart(Dataset data, Plot plot);`
> Creates a new `JFreeChart` instance. The chart will have no title, and no legend.

For greater control, a more complete constructor is available:

> `public JFreeChart(Dataset data, Plot plot, String title, Font titleFont, boolean createLegend);`
> Creates a new `JFreeChart` instance. This constructor allows you to specify a single title (you can add additional titles, later, if necessary).

7.25.3 Attributes

The `JFreeChart` class has the following attributes:

Attribute:	Description:
`titles`	A list of the titles for the chart.
`legend`	The chart legend.
`data`	The dataset.
`plot`	The plot.
`antialias`	A flag that indicates whether or not the chart should be drawn with anti-aliasing.
`chartBackgroundPaint`	The background paint for the chart.
`seriesPaint`	An array of paints used to identify series within a plot.
`seriesStroke`	An array of pen strokes used to plot the data for each series.
`seriesOutlinePaint`	An array of paints used to draw the outlines of series.
`seriesOutlineStroke`	An array of pen strokes used to draw the outlines of series.

7.25.4 Methods

To add a title to the chart:

`public void addTitle(AbstractTitle title);`
Adds a title to the chart.

To set the legend for a chart:

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

You can change the chart's dataset at any time, but you need to be sure that the new dataset is compatible with the type of plot currently set for the chart:

`public void setDataset(Dataset data);`
Changes the dataset for a chart.

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

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

Some plot types have the ability to generate tooltips, which can be useful if the chart is being displayed on-screen. To enable this function, you need to register a *tooltip manager* with the chart:

`public void setToolTips(ToolTips tooltips);`
Sets a tooltip manager with the chart.

To draw the chart:

`public void draw(Graphics2D g2, Rectangle2D chartArea, DrawInfo info);`
Draws the chart on the `Graphics2D` device. If the info argument is not `null`, it will be populated with information about the dimensions of the chart drawing.

To receive notification of any change to a chart, a listener object should register via this method:

`public void addChangeListener(ChartChangeListener listener);`
Register to receive chart change events.

To stop receiving change notifications, a listener object should deregister via this method:

`public void removeChangeListener(ChartChangeListener listener);`
Deregister to stop receiving chart change events.

7.25.5 Notes

JFreeChart works with `Dataset` interfaces that are defined in the JCommon Class Library. You can download JCommon from:

`http://www.jrefinery.com/jcommon`

The `ChartFactory` class provides some convenient methods for creating "ready-made" charts.

The Java2D API is used throughout JFreeChart, so JFreeChart does not work with JDK1.1 (a common question from applet developers, although hopefully less of an issue as browser support for Java2 improves).

A chart can have multiple titles (see `AbstractTitle`), although often you will require just one title or no title at all.

See Also
`ChartFactory`, `JFreeChartPanel`, `Plot`.

7.26 JFreeChartFrame

7.26.1 Overview

A frame containing chart within a `JFreeChartPanel`.

7.26.2 Constructors

There are two constructors:

`public JFreeChartFrame(String title, JFreeChart chart);`
Creates a new `JFreeChartFrame` containing the specified chart.

The second constructor gives you the opportunity to request that the chart is contained within a `JScrollPane`:

`public JFreeChartFrame(String title, JFreeChart chart, boolean scrollPane);`
Creates a new `JFreeChartFrame` containing the specified chart.

7.26.3 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`JFreeChartPanel`.

7.27 JFreeChartPanel

7.27.1 Overview

Provides a wrapper around `JFreeChart` to provide a convenient means of including a chart in a Swing-based user-interface. Extends `javax.swing.JComponent`.

7.27.2 Constructors

The standard constructor accepts a `JFreeChart` as the only parameter, and creates a panel that displays the chart. By default, the panel is automatically updated whenever the chart changes:

`public JFreeChartPanel(JFreeChart chart);`
Creates a new `JFreeChartPanel`.

7.27.3 Notes

The panel includes support for displaying tooltips for a chart.

See Also
`JFreeChart`.

7.28 Legend

7.28.1 Overview

Not yet documented.

7.28.2 Notes

Refer to Javadoc HTML files and source code for details.

7.29 Line

7.29.1 Overview

Not yet documented.

7.29.2 Notes

Refer to Javadoc HTML files and source code for details.

7.30 LineAndShapeRenderer

7.30.1 Overview

Not yet documented.

7.30.2 Notes

Refer to Javadoc HTML files and source code for details.

7.31 LinePlot

7.31.1 Overview

Not yet documented.

7.31.2 Notes

Refer to Javadoc HTML files and source code for details.

7.32 NumberAxis

7.32.1 Overview

The base class for axes (both horizontal and vertical) that display numerical data—extends `ValueAxis`.

7.32.2 Constructors

The `NumberAxis` class is `abstract`. Therefore you cannot instantiate this class directly—you must use a subclass (for example, `HorizontalNumberAxis` or `VerticalNumberAxis`).

Subclasses can call one of two constructors for the `NumberAxis` class. The simpler version requires only the axis label to be specified, with all other attributes taking default values:

```
protected NumberAxis(String label);
```
Creates a new `NumberAxis`.

The other constructor takes an extensive list of parameters, allowing much greater control over the construction of the axis. Refer to the Javadoc HTML pages or the source code for details.

7.32.3 Attributes

The following table lists the properties defined by `NumberAxis`[5]:

Attribute:	Description:
`MinimumAxisValue`	The lowest value displayed on the axis.
`MaximumAxisValue`	The highest value displayed on the axis.
`AutoRangeIncludesZero`	A flag that indicates whether or not zero is always included when the axis range is determined automatically.
`AutoRangeMinimumSize`	If the axis range is determined automatically, it is guaranteed never to be less that this value.
`UpperMargin`	The margin to allow at the upper end of the axis scale (expressed as a percentage of the total axis range).
`LowerMargin`	The margin to allow at the lower end of the axis scale (expressed as a percentage of the total axis range).
`TickUnit`	The spacing between ticks on the axis.
`StandardTickUnits`	A collection of standard tick units. If *auto-tick-selection* is on, one of these tick units will be selected automatically.

[5]Keep in mind that many other attributes are inherited from `ValueAxis`.

The following default values are used for attributes wherever necessary:

Name:	Value:
DEFAULT_MINIMUM_AXIS_VALUE	0.0
DEFAULT_MAXIMUM_AXIS_VALUE	1.0
DEFAULT_UPPER_MARGIN	0.05 (5 percent)
DEFAULT_LOWER_MARGIN	0.05 (5 percent)
DEFAULT_MINIMUM_AUTO_RANGE	new Double(0.0000001);
DEFAULT_TICK_UNIT	new NumberTickUnit(new Double(1.0), new DecimalFormat("0"));

7.32.4 Methods

To set the lower bound for the axis:

`public void setMinimumAxisValue(double value);`
Sets the lower bound for the axis. If the `AutoRange` attribute is `true` it is automatically switched to `false`. Registered listeners are notified of the change.

To set the upper bound for the axis:

`public void setMaximumAxisValue(double value);`
Sets the upper bound for the axis. If the `AutoRange` attribute is `true` it is automatically switched to `false`. Registered listeners are notified of the change.

If you have set the `AutoRange` flag to `true` (so that the axis range automatically adjusts to fit the current data), you may also want to set the `AutoRangeIncludesZero` flag to ensure that the axis range always includes zero:

`public void setAutoRangeIncludesZero(boolean flag);`
Sets the `AutoRangeIncludesZero` flag.

When the `AutoTickUnit` property is set to `true`, the axis will select a tick unit from a set of standard tick units. You can define your own standard tick units for an axis with the following method:

`public void setStandardTickUnits(TickUnits units);`
Sets the standard tick units for the axis.

7.32.5 Notes

This class defines a default set of standard tick units. You can override the default settings by calling the `setStandardTickUnits(...)` method.

See Also
`TickUnits`, `ValueAxis`.

7.33 NumberTickUnit

7.33.1 Overview

A numerical tick unit. The `NumberAxis` class creates a collection of standard tick units from which it can choose an appropriate tick unit for the range of data it is trying to display.

7.33.2 Constructors

The standard constructor:

```
public NumberTickUnit(Number value, NumberFormat formatter);
```
Creates a new number tick unit.

7.33.3 Notes

Extends the `TickUnit` class.

See Also
`TickUnit`.

7.34 PiePlot

7.34.1 Overview

The `PiePlot` class draws pie charts, using data obtained through the `PieDataset` interface (part of the JCommon Class Library).

7.34.2 Constructors

The default constructor:

```
protected PiePlot();
```
Creates a pie plot with default attributes.

7.34.3 Attributes

The `PiePlot` class has the following attributes:

Attribute:	Description:
`InteriorGapPercent`	The space to leave blank around the outside of the pie.
`Circular`	Circular or elliptical pie.
`RadiusPercent`	Controls the radius of the unexploded pie.
`SectionLabelType`	The type of labels for the pie sections.
`SectionLabelFont`	The font for the section labels.
`SectionLabelPaint`	The color for the section labels.
`SectionLabelGapPercent`	The gap for the section labels.
`ExplodePercentages[]`	The amount to 'explode' each pie section.
`PercentFormatter`	A formatter for the percentage labels.
`ToolTipGenerator`	A plug-in tooltip generator.

The following default values are used where necessary:

Name:	Value:
`DEFAULT_INTERIOR_GAP`	`0.20` (20 percent)
`DEFAULT_RADIUS`	`1.00` (100 percent)
`DEFAULT_SECTION_LABEL_FONT`	`new Font("SansSerif", Font.PLAIN, 10);`
`DEFAULT_SECTION_LABEL_PAINT`	`Color.black;`
`DEFAULT_SECTION_LABEL_GAP`	`0.10` (10 percent)

7.34.4 Methods

A pie plot is drawn with this method:

> `public void draw(Graphics2D g2, Rectangle2D drawArea, ToolTips tooltips);`
> Draws the pie plot within the specified drawing area.
>
> If `tooltips` is not `null`, then tooltips will be generated for each pie section as the pie plot is drawn.

The `JFreeChart` class usually calls the `draw(...)` method for you.

To set the tooltip generator (optional) for the pie plot:

> `public void setToolTipGenerator(PieToolTipGenerator generator) ;`
> Registers a tooltip generator with the pie plot. If you write your own generator, you can have full control over the tooltip text that is generated for each pie section.

7.34.5 Notes

`PiePlot` inherits axes from the `Plot` class. You should leave these set to `null`.

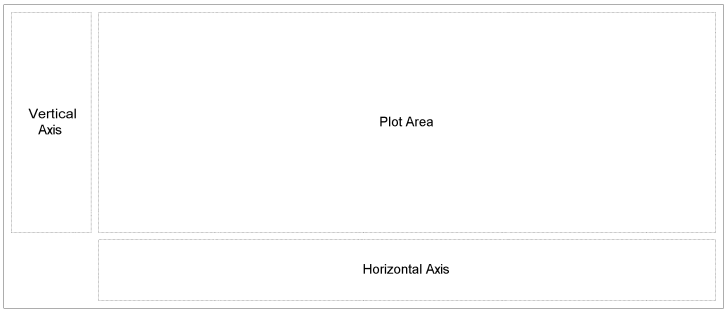
See Also
`Plot`.

7.35 Plot

7.35.1 Overview

An abstract base class that controls the visual representation of data in a chart. The `Plot` manages a horizontal axis and a vertical axis.

The plot is given an area (the *draw area*) by the `JFreeChart` object, into which it must draw the axes and the data. The following diagram shows how this area is divided:



Determining the dimensions of these regions is an awkward problem. The plot area can be resized arbitrarily, but the vertical axis and horizontal axis sizes are more difficult. Note that the height of the vertical axis is related to the height of the horizontal axis, and, likewise, the width of the vertical axis is related to the width of the horizontal axis. This results in a "chicken and egg" problem, because changing the width of an axis can affect its height (especially if the tick

units change with the resize) and changing its height can affect the width (for the same reason).

There are a number of concrete subclasses of `Plot`, including: `HorizontalBarPlot`, `VerticalBarPlot`, `PiePlot`, `XYPlot` and `HighLowPlot`.

7.35.2 Constructors

To create a new `Plot`:

```
protected Plot(Axis horizontalAxis, Axis verticalAxis);
```

Creates a new `Plot`, with the specified axes.

7.35.3 Attributes

The `Plot` class has the following attributes:

Attribute:	Description:
`Chart`	The chart that the plot belongs to.
`VerticalAxis`	The vertical axis.
`HorizontalAxis`	The horizontal axis.
`Insets`	The amount of space to leave around the outside of the plot.
`BackgroundPaint`	The color used to draw the background of the plot area.
`OutlineStroke`	The pen/brush used to draw an outline around the plot area.
`OutlinePaint`	The color used to draw an outline around the plot area.
`SeriesPaint`	An array of `Paint` objects used for the series colors.
`SeriesStroke`	An array of `Stroke` objects used for drawing series.
`SeriesOutlinePaint`	An array of `Paint` objects used for the series outline colors.

7.35.4 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`JFreeChart`.

7.36 PlotException

7.36.1 Overview

Not yet documented.

7.36.2 Notes

Refer to Javadoc HTML files and source code for details.

7.37 PlotNotCompatibleException

7.37.1 Overview

An exception that indicates that an attempt has been made to assign a plot to a chart where the plot is not compatible with the chart's current `Dataset`. For example, an `XYPlot` will not work with a `CategoryDataset`.

7.37.2 Constructors

To create a new exception:

```
public AxisNotCompatibleException(String message);
```
Creates a new exception.

7.37.3 Notes

The `PlotNotCompatibleException` class is a subclass of `RuntimeException`.

See Also
`AxisNotCompatibleException`.

7.38 StackedHorizontalBarRenderer

7.38.1 Overview

Not yet documented.

7.38.2 Notes

Refer to Javadoc HTML files and source code for details.

7.39 StackedVerticalBarRenderer

7.39.1 Overview

Not yet documented.

7.39.2 Notes

Refer to Javadoc HTML files and source code for details.

7.40 StackedVerticalBarRenderer3D

7.40.1 Overview

Not yet documented.

7.40.2 Notes

Refer to Javadoc HTML files and source code for details.

7.41 StandardLegend

7.41.1 Overview

This class is soon to be replaced by `LegendTitle`.

7.42 StandardTitle

7.42.1 Overview

Not yet documented.

7.42.2 Notes

Refer to Javadoc HTML files and source code for details.

7.43 StandardXYItemRenderer

7.43.1 Overview

The default renderer for the `XYPlot` class. This renderer represents data by drawing lines between *(x, y)* data points and/or drawing shapes at each *(x, y)* data point.

7.43.2 Constructors

To create a `StandardXYItemRenderer`:

> `public StandardXYItemRenderer(int type);`
> Creates a new renderer. The `type` argument should be one of: `LINES`, `SHAPES` or `SHAPES_AND_LINES`.

7.43.3 Notes

This class implements the `XYItemRenderer` interface.

The `XYPlot` class will use an instance of this class as its default renderer.

See Also
`XYPlot`, `XYItemRenderer`.

7.44 TextTitle

7.44.1 Overview

A standard chart title—extends `AbstractTitle`.

7.44.2 Notes

The original version of this class was written by David Berry.

See Also
`AbstractTitle`.

7.45 Tick

7.45.1 Overview

A utility class representing a tick on an axis. Used temporarily during the drawing process only.

7.45.2 Constructors

The standard constructor:

```
public Tick(Object value, String text, float x, float y)
```
Creates a tick.

See Also
`TickUnit`.

7.46 TickUnit

7.46.1 Overview

An abstract class representing a tick unit. Subclasses include `NumberTickUnit`.

7.46.2 Constructors

The standard constructor:

```
public TickUnit(Number value);
```
Creates a new tick value.

7.46.3 Notes

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

See Also
`NumberTickUnit`.

7.47 TickUnits

7.47.1 Overview

A collection of tick units. Used by the `Number` axis class to store a list of "standard" tick units, from which an appropriate tick unit is selected as the chart is being redrawn.

7.47.2 Constructors

The default constructor:

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

7.47.3 Methods

To add a new tick unit to the collection:

`public void add(TickUnit unit);`
Adds the tick unit to the collection.

To find the tick unit in the collection that is closest in size to another tick unit:

`public TickUnit getNearestTickUnit(TickUnit unit);`
Returns the tick unit that is closest in size to the specified unit.

7.47.4 Notes

The `NumberAxis` class has a private method `createStandardTickUnits()` that generates a tick unit collection (of standard tick sizes) for use by numerical axes.

See Also
`TickUnit`.

7.48 Title

7.48.1 Overview

Not yet documented.

7.49 ValueAxis

7.49.1 Overview

The base class for all (horizontal and vertical) axes that display "values". Ultimately, values are represented as `double` primitives, but subclasses of `ValueAxis` have been implemented that give the appearance of working with `Number` and `Date` objects.

Known subclasses of `ValueAxis` include `DateAxis` and `NumberAxis`.

7.49.2 Constructors

To construct a `ValueAxis`:

`public ValueAxis(String label);`
Creates a `ValueAxis` with the specified label. All other attributes take default values.

If you want more control over the settings for the axis, use this constructor:

```
protected ValueAxis(String label, Font labelFont, Paint labelPaint,
Insets labelInsets, boolean tickLabelsVisible, Font tickLabelFont, Paint
tickLabelPaint, Insets tickLabelInsets, boolean tickMarksVisible, Stroke
tickMarkStroke, boolean autoRange, boolean autoTickUnit,
boolean gridLinesVisible, Stroke gridStroke, Paint gridPaint);
```
Creates a `ValueAxis`.

7.49.3 Attributes

The `ValueAxis` class has the following attributes:

Attribute:	Description:
`AutoRange`	A flag controlling whether or not the axis automatically adjusts its range to reflect the range of data values.
`AutoTickUnit`	A flag controlling whether or not the tick units are selected automatically.
`GridLinesVisible`	A flag controlling whether or not grid lines are displayed.
`GridLineStroke`	The *stroke* used to draw the grid lines.
`GridLinePaint`	The color for the grid lines.

The following default values are used for attributes wherever necessary:

Name:	Value:
`DEFAULT_GRID_LINE_STROKE`	`new BasicStroke(0.5f, BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL, 0.0f, new float[] 2.0f, 2.0f, 0.0f);`
`DEFAULT_GRID_LINE_PAINT`	`Color.grey;`

7.49.4 Methods

A key function for a `ValueAxis` is to convert a data value to an output coordinate for plotting purposes. The output coordinate will be dependent on the area into which the data is being drawn:

`public double translateValueToJava2D(double value, Rectangle2D dataArea);`
Converts a data value into a co-ordinate within the `dataArea`. The `dataArea` is the rectangle inside the plot's axes.

See Also
`Axis`, `DateAxis`, `NumberAxis`.

7.50 VerticalAxis

7.50.1 Overview

An interface that *must* be implemented by all vertical axes. The methods defined by this interface are used by the `Plot` that owns the axis, for layout purposes.

7.50.2 Methods

The interface defines two methods. The plot chooses which of these two methods to call when laying out the axes.

`public Rectangle2D reserveAxisArea(Graphics2D g2, Plot plot, Rectangle2D drawArea, double reservedHeight);`
Calculates the area that the vertical axis requires to draw itself. If this method is used, it will be called after the horizontal axis has estimated the height that it requires—the argument `reservedHeight` contains this value.

`public double reserveWidth(Graphics2D g2, Plot plot, Rectangle2D drawArea);`
Estimates the width that the vertical axis requires to draw itself. If this method is used, it will be called before the horizontal axis is asked to calculate the area that it requires—the width returned by this method will be passed to the horizontal axis.

See Also
`HorizontalAxis`.

7.51 VerticalBarPlot

7.51.1 Overview

This plot draws a standard bar chart using data from a `CategoryDataset`, where the categories are plotted against the horizontal axis and the numerical data is plotted against the vertical axis.

7.51.2 Constructors

The simplest constructor requires only the axes to be specified:

`public VerticalBarPlot(CategoryAxis horizontalAxis, ValueAxis verticalAxis);`
Creates a vertical bar plot. Default values are assumed for most attributes.

For more complete control, use the following constructor:

`public VerticalBarPlot(CategoryAxis horizontalAxis, ValueAxis verticalAxis, Insets insets, double introGapPercent, double trailGapPercent, double categoryGapPercent, double itemGapPercent, CategoryToolTipGenerator toolTipGenerator);`
Creates a vertical bar plot.

7.51.3 Methods

The category axis will need to ask the plot for the coordinate of a particular category, since the plot controls the distribution of the categories. This method is used:

`public double getCategoryCoordinate(...);`
This method returns the x-coordinate of the center of the specified category. The category axis will call this method to determine where to place the category labels, because it has no knowledge of the distribution of categories (these could vary depending on the nature of the plot).

7.51.4 Notes

The bar widths cannot be controlled directly. Instead, you set the amount (percentage) of the total space that should be allocated to the gaps *between* the bars, and then the bar widths are determined automatically.

See Also
`BarPlot`, `HorizontalBarPlot`, `VerticalValuePlot`.

7.52 VerticalBarPlot3D

7.52.1 Overview

Not yet documented.

7.52.2 Notes

Refer to Javadoc HTML files and source code for details.

7.53 VerticalBarRenderer

7.53.1 Overview

Not yet documented.

7.53.2 Notes

Refer to Javadoc HTML files and source code for details.

7.54 VerticalBarRenderer3D

7.54.1 Overview

Not yet documented.

7.54.2 Notes

Refer to Javadoc HTML files and source code for details.

7.55 VerticalCategoryAxis

7.55.1 Overview

A vertical axis that displays categorical data. This class extends `CategoryAxis`. As for the other category axes, this class relies on the plot to provide information about how the categories are distributed along the axis (this information is obtained via the `CategoryPlot` interface).

7.55.2 Constructors

There are two constructors for this class. One requires all the attributes for the axis to be specified, the other provides for default values on some attributes. Refer to the Javadoc or source code for details.
The default constructor:

```
public VerticalCategoryAxis(String label);
```
Creates a new `VerticalCategoryAxis`.

See Also
`CategoryAxis`, `HorizontalCategoryAxis`.

7.56 VerticalNumberAxis

7.56.1 Overview

A vertical axis that displays numerical data—this class extends `NumberAxis`.

7.56.2 Constructors

There are three constructors for this class. One requires all the attributes for the axis to be specified, the other two provide for default values on some attributes. Refer to the Javadoc or source code for details.

7.56.3 Methods

A list of important methods:

`public void autoAdjustRange();`
This method obtains the maximum and minimum data values from the `Plot`, provided that it implements `VerticalValueRange`, and adjusts the axis range accordingly. Note that the `autoRangeIncludesZero` flag is checked in this method.

`public void refreshTicks(...);`
A utility method for calculating the positions of the ticks on an axis, just prior to drawing the axis. This method checks the `autoTickUnits` flag, and automatically determines a suitable "standard" tick size if required.

`private void calculateAutoTickUnits(...);`
This method is used to pick a standard tick size from the array defined in `NumberAxis`. The approach used is to find the smallest tick units such that the tick labels do not overlap.

See Also
`NumberAxis`, `HorizontalNumberAxis`.

7.57 VerticalNumberAxis3D

7.57.1 Overview

Not yet documented.

7.57.2 Notes

Refer to Javadoc HTML files and source code for details.

7.58 VerticalValuePlot

7.58.1 Overview

An interface that returns minimum and maximum data values in the "vertical direction" for a two-dimensional plot. The values could be from the dataset domain or range, depending on the orientation of the plot.

7.58.2 Methods

This interface has two methods:

`public Number getMinimumVerticalDataValue();`
Returns the minimum data value in the vertical direction for the plot.

`public Number getMaximumVerticalDataValue();`
Returns the maximum data value in the vertical direction for the plot.

7.58.3 Notes

This interface is known to be implemented by `LinePlot`, `VerticalBarPlot` and `XYPlot`.

See Also
`HorizontalValuePlot`.

7.59 VerticalXYBarPlot

7.59.1 Overview

Not yet documented.

7.59.2 Notes

Refer to Javadoc HTML files and source code for details.

7.60 VerticalXYBarRenderer

7.60.1 Overview

Not yet documented.

7.60.2 Notes

Refer to Javadoc HTML files and source code for details.

7.61 XYItemRenderer

7.61.1 Overview

An interface that must be implemented by a *renderer* so that it can work with an `XYPlot`. By changing the renderer for an `XYPlot`, you can change the appearance of the plot.

Several implementations of this interface are available:

- `StandardXYItemRenderer`;
- `CandlestickRenderer`;
- `HighLowRenderer`;
- `VerticalXYBarRenderer`.

7.61.2 Methods

There is just one method in this interface:

```
public Shape drawItem(Graphics2D g2, Rectangle2D plotArea, Plot plot,
ValueAxis horizontalAxis, ValueAxis verticalAxis, XYDataset data, int series,
int item, double translatedRangeZero);
```
Draws a single data item on behalf of `XYPlot`.

7.61.3 Notes

Some renderers require a specific subclass of `XYDataset`.

See Also
`XYPlot`.

7.62 XYPlot

7.62.1 Overview

Draws a visual representation of data from an `XYDataset`, where the horizontal axis measures the x-values and the vertical axis measures the y-values.

It is possible to display time series data with `XYPlot` by employing a `HorizontalDateAxis` in place of the usual `HorizontalNumberAxis`. In this case, the x-values are interpreted as milliseconds as used in `java.util.Date`.

7.62.2 Constructors

The simplist constructor requires just the axes to be specified:

```
public XYPlot(ValueAxis horizontalAxis, ValueAxis verticalAxis);
```
Creates an XY plot. Default values are used where necessary.

```
public XYPlot(ValueAxis horizontalAxis, ValueAxis verticalAxis, Insets
insets, Paint background, Stroke outlineStroke, Paint outlinePaint);
```
Creates an XY plot.

7.62.3 Methods

To get the current renderer for the plot:

```
public XYItemRenderer getItemRenderer();
```
Returns the current renderer.

To set a new renderer for the plot:

```
public void setItemRenderer(XYItemRenderer renderer);
```
Sets a new renderer.

7.62.4 Notes

The `XYPlot` class works with a *renderer* to control the visual representation of the data. By default, a renderer is installed that draws lines between each of the data points.

`XYPlot` implements both `HorizontalValuePlot` and `VerticalValuePlot`, enabling the axes to automatically determine the range of data that is available for the plot.

See Also

`Plot`, `StandardXYItemRenderer`.

8 Package: com.jrefinery.chart.combination

This package includes most of the classes—written by Bill Kelemen—for creating combination charts.

8.1 AbstractAxisRange

8.1.1 Overview

A base class for representing an axis range.

8.1.2 Constructors

To create a new range:

`public AbstractAxisRange(Object min, Object max);`
Creates a new range.

8.1.3 Methods

A list of important methods:

`public void combine(AxisRange range);`
Extends this range (if necessary) to cover another range.

See Also
`AxisRange`.

8.2 AxisRange

8.2.1 Overview

An interface for an axis range.

8.2.2 Methods

The interface defines three methods:

`public Object getMin();`
Returns the minimum value in the range.

`public Object getMax();`
Returns the maximum value in the range.

`public void combine(AxisRange range);`
Extends this range (if necessary) to incorporate another range.

See Also
`AbstractAxisRange`.

8.3 CombinableAxis

8.3.1 Overview

Interface for an axis in a combined chart.

See Also
`CombinedPlot`.

8.4 CombinedChart

8.4.1 Overview

An extension of `JFreeChart` for displaying combined charts.

See Also
`JFreeChart`.

8.5 CombinedHorizontalDateAxis

8.5.1 Overview

An extension of `HorizontalDateAxis` for use in combined charts.

See Also
`HorizontalDateAxis`.

8.6 CombinedHorizontalNumberAxis

8.6.1 Overview

An extension of HorizontalNumberAxis for use in combined charts.

See Also
`HorizontalNumberAxis`.

8.7 CombinedPlot

8.7.1 Overview

A plot for creating combined charts.

See Also
`CombinedChart`.

8.8 CombinedVerticalNumberAxis

8.8.1 Overview

An extension of `VerticalNumber` axis used for creating combined charts.

See Also
`CombinedPlot`.

8.9 DateAxisRange

8.9.1 Overview

Not yet documented.

8.9.2 Notes

Refer to Javadoc HTML files and source code for details.

8.10 NumberAxisRange

8.10.1 Overview

Not yet documented.

8.10.2 Notes

Refer to Javadoc HTML files and source code for details.

8.11 OverlaidHorizontalDateAxis

8.11.1 Overview

Not yet documented.

8.11.2 Notes

Refer to Javadoc HTML files and source code for details.

8.12 OverlaidHorizontalNumberAxis

8.12.1 Overview

Not yet documented.

8.12.2 Notes

Refer to Javadoc HTML files and source code for details.

8.13 OverlaidPlot

8.13.1 Overview

Not yet documented.

8.13.2 Notes

Refer to Javadoc HTML files and source code for details.

8.14 OverlaidVerticalNumberAxis

8.14.1 Overview

Not yet documented.

8.14.2 Notes

Refer to Javadoc HTML files and source code for details.

9 Package: com.jrefinery.chart.data

This package contains some classes for data fitting. These will eventually be rewritten and moved into another package.

9.1 LinearPlotFitAlgorithm

9.1.1 Overview

Not yet documented.

9.1.2 Notes

Refer to Javadoc HTML files and source code for details.

9.2 MovingAveragePlotFitAlgorithm

9.2.1 Overview

Not yet documented.

9.2.2 Notes

Refer to Javadoc HTML files and source code for details.

9.3 PlotFit

9.3.1 Overview

Not yet documented.

9.3.2 Notes

Refer to Javadoc HTML files and source code for details.

9.4 PlotFitAlgorithm

9.4.1 Overview

Not yet documented.

9.4.2 Notes

Refer to Javadoc HTML files and source code for details.

10 Package: com.jrefinery.chart.event

This package contains classes and interfaces that are used to broadcast and receive events relating to changes in chart properties. By default, some of the classes in the library will automatically register themselves with other classes, so that they receive notification of any changes and can react accordingly. For the most part, you can simply rely on this default behaviour.

10.1 AxisChangeEvent

10.1.1 Overview

An event that is used to provide information about changes to axes.

See Also
`AxisChangeListener`.

10.2 AxisChangeListener

10.2.1 Overview

An interface through which axis change event notifications are posted. If a class needs to receive notification of changes to an axis, then it needs to implement this interface and register itself with the axis.

10.2.2 Methods

The interface defines a single method:

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

See Also
`AxisChangeEvent`.

10.3 ChartChangeEvent

10.3.1 Overview

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

See Also
`ChartChangeListener`.

10.4 ChartChangeListener

10.4.1 Overview

An interface through which chart change event notifications are posted. If a class needs to receive notification of changes to a chart, then it needs to implement this interface and register itself with the chart.

10.4.2 Methods

The interface defines a single method:

`public void chartChanged(ChartChangeEvent event);`
Receives notification of a change to a chart.

See Also
`ChartChangeEvent`.

10.5 LegendChangeEvent

10.5.1 Overview

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

See Also
`LegendChangeListener`.

10.6 LegendChangeListener

10.6.1 Overview

An interface through which legend change event notifications are posted. If a class needs to receive notification of changes to a legend, then it needs to implement this interface and register itself with the legend.

10.6.2 Methods

The interface defines a single method:

`public void legendChanged(LegendChangeEvent event);`
Receives notification of a change to a legend.

See Also
`LegendChangeEvent`.

10.7 PlotChangeEvent

10.7.1 Overview

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

See Also
`PlotChangeListener`.

10.8 PlotChangeListener

10.8.1 Overview

An interface through which plot change event notifications are posted. If a class needs to receive notification of changes to a plot, then it needs to implement this interface and register itself with the plot.

10.8.2 Methods

The interface defines a single method:

`public void plotChanged(PlotChangeEvent event);`
Receives notification of a change to a plot.

See Also
`PlotChangeEvent`.

10.9 TitleChangeEvent

10.9.1 Overview

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

See Also
`TitleChangeListener`.

10.10 TitleChangeListener

10.10.1 Overview

An interface through which title change event notifications are posted. If a class needs to receive notification of changes to a title, then it needs to implement this interface and register itself with the title.

10.10.2 Methods

The interface defines a single method:

`public void titleChanged(TitleChangeEvent event);`
Receives notification of a change to a title.

See Also
`TitleChangeEvent`.

11 Package: com.jrefinery.chart.tooltips

This package contains some classes for generating tooltips.

11.1 CategoryToolTipGenerator

11.1.1 Overview

The interface that should be implemented by a *category tooltip generator*. The idea is that you can develop your own tooltip generator, register it with a plot, and take full control over the tooltip text that is generated.

11.1.2 Methods

This interface defines a single method:

> `public String generateToolTip(CategoryDataset data, int series, Object category);`
> This method is called whenever the plot needs to generate a tooltip. It should return the tooltip text (which can be anything you want to make it).

11.1.3 Notes

The `StandardCategoryToolTipGenerator` is one implementation of this interface, but you are free to write your own implementation to suit your requirements.

11.2 PieToolTipGenerator

11.2.1 Overview

The interface that should be implemented by a *pie tooltip generator*. The idea is that you can develop your own tooltip generator, register it with a `PiePlot`, and take full control over the tooltip text that is generated.

11.2.2 Methods

This interface defines a single method:

> `public String generateToolTip(PieDataset data, Object category);`
> This method is called whenever the `PiePlot` needs to generate a tooltip. It should return a `String` that will be used as the tooltip text.

11.2.3 Notes

The `StandardPieToolTipGenerator` is one implementation of this interface, but you are free to write your own implementation to suit your requirements.

See Also
`StandardPieToolTipGenerator`.

11.3 StandardCategoryToolTipGenerator

11.3.1 Overview

A default implementation of the `CategoryToolTipGenerator` interface.

11.3.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`CategoryToolTipGenerator`.

11.4 StandardHighLowToolTipGenerator

11.4.1 Overview

A default implementation of the `HighLowToolTipGenerator` interface.

11.4.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`HighLowToolTipGenerator`.

11.5 StandardPieToolTipGenerator

11.5.1 Overview

A default implementation of the `PieToolTipGenerator` interface.

11.5.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`PieToolTipGenerator`.

11.6 StandardToolTips

11.6.1 Overview

An implementation of the `ToolTips` interface, this class can be registered with a chart via the `setToolTips(...)` method and will collect tooltips as the chart is being drawn.

11.6.2 Constructors

Use the default constructor to create a new tooltip manager:

`public StandardToolTips();`
Creates a new tooltip manager.

11.6.3 Methods

This class provides implementations for all the methods in the `ToolTips` interface.

11.6.4 Notes

This implementation is not highly optimised. If you are using generating charts with large numbers of data items, you should either stop using tooltips, or write a more efficient implementation.

See Also
`ToolTips`.

11.7 StandardXYToolTipGenerator

11.7.1 Overview

A default implementation of the `XYToolTipGenerator` interface.

11.7.2 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`XYToolTipGenerator`.

11.8 ToolTip

11.8.1 Overview

A simple class representing a tooltip. It records the tooltip text, and the area that the tooltip applies to.

11.8.2 Notes

This class is immutable.

See Also
`ToolTipGenerator`.

11.9 ToolTipGenerator

11.9.1 Overview

Not yet documented.

11.9.2 Notes

Refer to Javadoc HTML files and source code for details.

11.10 ToolTips

11.10.1 Overview

An interface defining the methods to be supported by a *tooltip manager*.

If you set a tooltip manager for a chart, then it will collect tooltips as the chart is being drawn (provided that the `Plot` subclass is capable of generating tooltips). The `JFreeChartPanel` class makes use of this facility to provide chart tooltips.

11.10.2 Notes

By default, there is no tooltip manager set for a chart.

See Also
`StandardToolTips`.

11.11 XYToolTipGenerator

11.11.1 Overview

The interface that should be implemented by a *XY tooltip generator*. The idea is that you can develop your own tooltip generator, register it with a plot, and take full control over the tooltip text that is generated.

11.11.2 Methods

This interface defines a single method:

> `public String generateToolTip(XYDataset data, int series, int item);`
> This method is called whenever the `XYPlot` needs to generate a tooltip. It should return a `String` that will be used as the tooltip text.

11.11.3 Notes

Refer to Javadoc HTML files and source code for details.

See Also
`StandardXYToolTipGenerator`.

12 Package: com.jrefinery.chart.ui

This package contains user interface classes that can be used to modify chart properties. These classes are optional—they are used in the demonstration application, but you do not need to include this package in your own projects if you do not want to.

12.1 AxisPropertyEditPanel

12.1.1 Overview

Not yet documented.

12.1.2 Notes

Refer to Javadoc HTML files and source code for details.

12.2 ChartPropertyEditPanel

12.2.1 Overview

A panel that displays all the properties of a chart, and allows the user to edit the properties. The panel uses a `JTabbedPane` to display four sub-panels: a `TitlePropertyPanel`, a `LegendPropertyPanel`, a `PlotPropertyPanel` and a panel containing "other" properties (such as the anti-alias setting and the background paint for the chart).

The constructors for this class require a reference to a `Dialog` or a `Frame`. Whichever one is specified is passed on to the `TitlePropertyPanel` and is used if and when a sub-dialog is required for editing titles.

12.2.2 Notes

Refer to Javadoc HTML files and source code for details.

12.3 LegendPropertyEditPanel

12.3.1 Overview

Not yet documented.

12.3.2 Notes

Refer to Javadoc HTML files and source code for details.

12.4 NumberAxisPropertyEditPanel

12.4.1 Overview

Not yet documented.

12.4.2 Notes

Refer to Javadoc HTML files and source code for details.

12.5 PlotPropertyEditPanel

12.5.1 Overview

Not yet documented.

12.5.2 Notes

Refer to Javadoc HTML files and source code for details.

12.6 TitlePropertyEditPanel

12.6.1 Overview

Not yet documented.

12.6.2 Notes

Refer to Javadoc HTML files and source code for details.

13 Package: com.jrefinery.data

This package is part of the JCommon Class Library, which can be downloaded from:

`http://www.jrefinery.com/jcommon`

The reference documentation for this package is included here, even though it is not strictly part of the JFreeChart Class Library, because JFreeChart makes extensive use of the interfaces and classes in this package.

13.1 AbstractDataset

13.1.1 Overview

A useful base class for implementing the `Dataset` interface (or extensions). This class provides a default implementation of the *change listener* mechanism.

13.1.2 Constructors

The default constructor:

`protected AbstractDataset();`
Allocates storage for the registered change listeners.

13.1.3 Methods

`public void addChangeListener(DatasetChangeListener listener);`
Registers a change listener with the dataset. The listener will be notified whenever the dataset changes.

`public void addChangeListener(DatasetChangeListener listener);`
Deregisters a change listener. The listener will be no longer be notified whenever the dataset changes.

13.1.4 Notes

You can implement a dataset without subclassing `AbstractDataset`. This class is provided simply for convenience to save you having to implement your own change listener mechanism.

See Also

`Dataset`, `DatasetChangeListener`, `AbstractSeriesDataset`.

13.2 AbstractSeriesDataset

13.2.1 Overview

A useful base class for implementing the `SeriesDataset` interface (or extensions). This class extends `AbstractDataset`.

13.2.2 Constructors

The default constructor:

`protected AbstractSeriesDataset();`
Simply calls the constructor of the superclass.

13.2.3 Methods

Implementations are provided for the following methods:

`public String[] getLegendItemLabels();`
Returns an array of series names.

13.2.4 Notes

You can implement a dataset without subclassing `AbstractSeriesDataset`. This class is provided simply for convenience.

See Also

`Dataset`.

13.3 BasicTimeSeries

13.3.1 Overview

A time series is a data structure that associates numeric values with particular time periods. In other words, a collection of data values in the form *(timeperiod, value)*.

The time periods are represented by subclasses of `TimePeriod`. Subclasses include `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second` and `Millisecond`. Different subclasses of `TimePeriod` cannot be mixed in one time series.

A time series may contain zero, one or many time periods with associated data values.

13.3.2 Constructors

There are three constructors:

`public BasicTimeSeries(String name);`
Creates an empty time series for daily data (that is, one value per day).

`public BasicTimeSeries(String name, Class timePeriodClass);`
Creates an empty time series. The caller specifies the time period.

`public BasicTimeSeries(String name, String domain, String range, Class timePeriodClass);`
Creates an empty time series. The caller specifies the time period, plus strings describing the domain and range.

13.3.3 Methods

`public void add(TimePeriod period, Number value) throws SeriesException;`
Adds a new value to the time series. Throws an exception if the time period is not unique within the series.

13.3.4 Notes

The class name was formerly `TimeSeries`, but this has been changed to avoid confusion with the subclass in the `com.jrefinery.finance` package.

See Also

`TimeSeriesCollection`

13.4 CategoryDataset

13.4.1 Overview

An interface (extending `SeriesDataset`) that defines the structure of a *category dataset*. The dataset consists of a table of series and categories. A value is associated with each combination of series and category (`null` values are permitted).

13.4.2 Methods

To obtain the number of categories:

```
public int getCategoryCount();
```
Returns the number of categories in the dataset.

To get a list of the categories in the dataset:

```
public List getCategories();
```
Returns a list of the categories in the dataset.

To get the value for a series/category combination:

```
public Number getValue(int series, Object category);
```
Returns the value associated with a particular series and category. The value may be `null`.

13.4.3 Notes

You can use any `Object` instance to represent a category. Using `String` is convenient, as the `toString()` method is used whenever a label is required for a category.

This interface is intended for reading data, not updating it.

See Also

`DefaultCategoryDataset`, `SeriesDataset`.

13.5 CombinationDataset

13.5.1 Overview

An interface for combining datasets. Written by Bill Kelemen.

13.5.2 Notes

This interface is used to create combined charts with the JFreeChart class library.

See Also

`CombinedDataset`.

13.6 CombinedDataset

13.6.1 Overview

An implementation of the `CombinationDataset` interface. Written by Bill Kelemen.

13.6.2 Notes

This class is used to create combined charts with the JFreeChart class library.

See Also

`CombinationDataset`.

13.7 Dataset

13.7.1 Overview

The base interface for datasets. Not useful in its own right, this interface is further extended by `PieDataset`, `CategoryDataset` and `SeriesDataset`.

13.7.2 Methods

A couple of methods relate to the use of datasets for drawing charts (see JFreeChart):

```
public int getLegendItemCount();
```
Returns the number of items to display in the legend.

```
public String[] getLegendItemLabels();
```
Returns an array of strings to use as labels in the legend.

Two further methods are used for registering change listeners with the dataset:

```
public void addChangeListener(DatasetChangeListener listener);
```
Registers a change listener with the dataset.

```
public void removeChangeListener(DatasetChangeListener listener);
```
Deregisters a change listener.

13.7.3 Notes

This interface is not intended to be used directly, you should use an extension of this interface such as `PieDataset`, `CategoryDataset` or `XYDataset`.

See Also

`PieDataset`, `SeriesDataset`.

13.8 DatasetChangeEvent

13.8.1 Overview

An event that is used to provide information about changes to datasets.

See Also

`DatasetChangeListener`.

13.9 DatasetChangeListener

13.9.1 Overview

An interface through which dataset change event notifications are posted. If a class needs to receive notification of changes to a dataset, then it needs to implement this interface and register itself with the dataset.

13.9.2 Methods

The interface defines a single method:

`public void datasetChanged(DatasetChangeEvent event);`
Receives notification of a change to a dataset.

See Also
`DatasetChangeEvent`.

13.10 Datasets

13.10.1 Overview

A collection of utility methods for working with datasets.

13.10.2 Methods

To get the minimum and maximum domain values in a dataset:

`public static Number getMinimumDomainValue(Dataset data);`
Returns the minimum domain value for the dataset.

`public static Number getMaximumDomainValue(Dataset data);`
Returns the maximum domain value for the dataset.

To get the minimum and maximum range values in a dataset:

`public static Number getMinimumRangeValue(Dataset data);`
Returns the minimum range value for the dataset.

`public static Number getMaximumRangeValue(Dataset data);`
Returns the maximum range value for the dataset.

To create a `PieDataset` from a `CategoryDataset`:

`public static PieDataset createPieDataset(CategoryDataset data, Object category);`
Returns a pie dataset by taking all the values in the category dataset for the specified category.

`public static PieDataset createPieDataset(CategoryDataset data, int series) ;`
Returns a pie dataset by taking all the values in the category dataset for the specified series.

See Also
`DomainInfo`, `RangeInfo`.

13.11 Day

13.11.1 Overview

A subclass of `TimePeriod` that represents one day. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

13.11.2 Constructor

To construct a `Day` instance:

`public Day(int day, int month, int year);`
Creates a new `Day` instance. The `year` argument should be in the range 1900 to 9999.

To create a `Day` instance based on a `SerialDate`:

`public Day(SerialDate day);`
Creates a new `Day` instance.

To create a `Day` instance based on a `Date`:

`public Day(Date time);`
Creates a new `Day` instance.

The default constructor creates a `Day` instance based on the current system date:

`public Day();`
Creates a new `Day` instance for the current system date.

13.11.3 Methods

To access the day:

`public SerialDate getDay();`
Returns the day as a `SerialDate`.

There is no method to *set* the day, because this class is immutable.

Given a `Day` object, you can create an instance representing the previous day or the next day:

`public TimePeriod previous();`
Returns the previous day, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next day, or null if the upper limit of the range is reached.

To convert a `Day` object to a `String` object:

`public String toString();`
Returns a string representing the day.

`public static Day parseDay(String s) throws TimePeriodFormatException;`
Parses the string and, if possible, returns a `Day` object.

13.11.4 Notes

In the current implementation, the day can be in the range 1-Jan-1900 to 31-Dec-9999.

The `Day` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `SerialDate`.

13.12 DefaultCategoryDataset

A quick and dirty implementation of the `CategoryDataset` interface. This class is due for a rewrite.

See Also
`CategoryDataset`

13.13 DefaultPieDataset

13.13.1 Overview

A convenient implementation of the `PieDataset` interface.

13.13.2 Constructors

The default constructor creates an empty pie dataset:

`public DefaultPieDataset();`
Creates a new dataset, initially empty.

`public DefaultPieDataset(Collection values);`
Creates a new dataset containing the values supplied. Section names are automatically generated.

13.13.3 Methods

To get the value for a particular category:

`public Number getValue(Object category);`
Returns the number associated with a category. This method can return `null`.

To set the value for a particular category:

`public void setValue(Object category, Number value);`
Sets the number associated with a category.

13.13.4 Notes

The dataset can contain `null` values.

See Also
`PieDataset`.

13.14 DefaultXYDataset

A quick and dirty implementation of the `XYDataset` interface. This class is in the process of being replaced by `XYSeriesCollection`.

See Also

`XYDataset`

13.15 DomainInfo

13.15.1 Overview

An interface that provides information about the minimum and maximum values in a dataset's domain.

13.15.2 Methods

To get the minimum value in the dataset's domain:

`public Number getMinimumDomainValue();`
Returns the minimum value in the dataset's domain.

To get the maximum value in the dataset's domain:

`public Number getMaximumDomainValue();`
Returns the maximum value in the dataset's domain.

13.15.3 Notes

It is not mandatory for a dataset to implement this interface. However, sometimes it is necessary to calculate the minimum and maximum values in a dataset. Without knowing the internal structure of a dataset, the only means of determining this information is iteration over the entire dataset. If there is a more efficient way to determine the values for your data structures, then you can implement this interface and provide the values directly.

See Also

`RangeInfo`.

13.16 HighLowDataset

An extension of the `XYDataset` interface, that supplies data in the form of "high/low, open/close" items.

`public Number getHighValue(int series, int item);`
Returns the high value for an item within a series.

`public Number getLowValue(int series, int item);`
Returns the low value for an item within a series.

`public Number getOpenValue(int series, int item);`
Returns the open value for an item within a series.

`public Number getCloseValue(int series, int item);`
Returns the close value for an item within a series.

This interface is used in the JFreeChart library.

13.17 Hour

13.17.1 Overview

A subclass of `TimePeriod` that represents one hour in a particular day. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

13.17.2 Constructor

To construct an `Hour` instance:

`public Hour(int hour, Day day);`
Creates a new `Hour` instance. The `hour` argument should be in the range 1 to 24.

To construct an `Hour` instance based on a `java.util.Date`:

`public Hour(Date time);`
Creates a new `Hour` instance.

A default constructor is provided:

`public Hour();`
Creates a new `Hour` instance based on the current system time.

13.17.3 Methods

To access the hour:

`public int getHour();`
Returns the hour (in the range 1 to 24).

To access the day:

`public Day getDay();`
Returns the day.

There is no method to *set* the hour or the day, because this class is immutable.

Given a `Hour` object, you can create an instance representing the previous hour or the next hour:

`public TimePeriod previous();`
Returns the previous hour, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next hour, or null if the upper limit of the range is reached.

13.17.4 Notes

The `Hour` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Day`.

13.18 IntervalXYZDataset

A natural extension of the `IntervalXYDataset` interface.

13.19 Millisecond

13.19.1 Overview

A subclass of `TimePeriod` that represents one millisecond. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

13.19.2 Constructors

To construct a `Millisecond` instance:

> `public Millisecond(long millisecond);`
> Creates a new `Millisecond` instance. The `millisecond` argument uses the same encoding as `java.util.Date`.

You can construct a a `Millisecond` instance based on a `java.util.Date` instance:

> `public Millisecond(Date time);`
> Creates a new `Millisecond` instance represnting the same millisecond as the `time` argument.

A default constructor is provided, which creates a `Millisecond` instance based on the current system time:

> `public Millisecond();`
> Creates a new `Millisecond` instance based on the current system time.

13.19.3 Methods

To access the millisecond:

> `public int getMillisecond();`
> Returns the millisecond.

There is no method to *set* the millisecond, because this class is immutable.

Given a `Millisecond` object, you can create an instance representing the previous millisecond or the next millisecond:

> `public TimePeriod previous();`
> Returns the previous millisecond, or null if the lower limit of the range is reached.
>
> `public TimePeriod next();`
> Returns the next millisecond, or null if the upper limit of the range is reached.

13.19.4 Notes

The `Millisecond` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `java.util.Date`.

13.20 Minute

13.20.1 Overview

A subclass of `TimePeriod` that represents one minute in a particular day. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

13.20.2 Constructors

To construct a `Minute` instance:

> `public Minute(int minute, Day day);`
> Creates a new `Minute` instance. The `minute` argument should be in the range 1 to 24*60.

To construct a `Minute` instance based on a `java.util.Date`:

> `public Minute(Date time);`
> Creates a new `Minute` instance.

A default constructor is provided:

> `public Minute();`
> Creates a new `Minute` instance, based on the current system time.

13.20.3 Methods

To access the minute:

> `public int getMinute();`
> Returns the minute (in the range 1 to 24*60).

To access the day:

> `public Day getDay();`
> Returns the day.

There is no method to *set* the minute or the day, because this class is immutable.

Given a `Minute` object, you can create an instance representing the previous minute or the next minute:

> `public TimePeriod previous();`
> Returns the previous minute, or null if the lower limit of the range is reached.
>
> `public TimePeriod next();`
> Returns the next minute, or null if the upper limit of the range is reached.

13.20.4 Notes

The `Minute` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Day`.

13.21 Month

13.21.1 Overview

A subclass of `TimePeriod` that represents one month in a particular year. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

13.21.2 Constructors

To construct a `Month` instance:

`public Month(int month, Year year);`
Creates a new `Month` instance. The `month` argument should be in the range 1 to 12.

`public Month(int month, int year);`
Creates a new `Month` instance.

To construct a `Month` instance based on a `java.util.Date`:

`public Month(Date time);`
Creates a new `Month` instance.

A default constructor is provided:

`public Month();`
Creates a new `Month` instance, based on the current system time.

13.21.3 Methods

To access the month:

`public int getMonth();`
Returns the month (in the range 1 to 12).

To access the year:

`public Year getYear();`
Returns the year.

There is no method to *set* the month or the year, because this class is immutable.

Given a `Month` object, you can create an instance representing the previous month or the next month:

`public TimePeriod previous();`
Returns the previous month, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next month, or null if the upper limit of the range is reached.

To convert a `Month` object to a `String` object:

`public String toString();`
Returns a string representing the month.

13.21.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Month` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Year`.

13.22 PieDataset

13.22.1 Overview

The interface for a dataset that associates values with categories.

13.22.2 Methods

Three methods are defined in the interface:

`public int getCategoryCount();`
Returns the number of categories in the dataset.

`public Set getCategories();`
Returns the set of categories.

`public Number getValue(Object category);`
Returns the value associated with a particular category.

13.22.3 Notes

The name of the interface is derived from a common usage for this type of dataset—the creation of pie charts.

There are some convenient methods for creating a `PieDataset` object by slicing a `CategoryDataset`. Refer to the `Datasets` class for more details.

See Also
`DefaultPieDataset`.

13.23 Quarter

13.23.1 Overview

A subclass of `TimePeriod` that represents one quarter in a particular year. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

13.23.2 Constructor

To construct a `Quarter` instance:

`public Quarter(int quarter, Year year);`
Creates a new `Quarter` instance. The `quarter` argument should be in the range 1 to 4.

`public Quarter(int quarter, int year);`
Creates a new `Quarter` instance.

To construct a `Quarter` instance based on a `java.util.Date`:

`public Quarter(Date time);`
Creates a new `Quarter` instance.

A default constructor is provided:

`public Quarter();`
Creates a new `Quarter` instance based on the current system time.

13.23.3 Methods

To access the quarter:

`public int getQuarter();`
Returns the quarter (in the range 1 to 4).

To access the year:

`public Year getYear();`
Returns the year.

There is no method to *set* the quarter or the year, because this class is immutable.

Given a `Quarter` object, you can create an instance representing the previous quarter or the next quarter:

`public TimePeriod previous();`
Returns the previous quarter, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next quarter, or null if the upper limit of the range is reached.

To convert a `Quarter` object to a `String` object:

`public String toString();`
Returns a string representing the quarter.

13.23.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Quarter` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Year`.

13.24 RangeInfo

13.24.1 Overview

An interface that provides information about the minimum and maximum values in a dataset's range.

13.24.2 Methods

To get the minimum value in the dataset's range:

`public Number getMinimumRangeValue();`
Returns the minimum value in the dataset's range.

To get the maximum value in the dataset's range:

`public Number getMaximumRangeValue();`
Returns the maximum value in the dataset's range.

13.24.3 Notes

It is not mandatory for a dataset to implement this interface. However, sometimes it is necessary to calculate the minimum and maximum values in a dataset. Without knowing the internal structure of a dataset, the only means of determining this information is iteration over the entire dataset. If there is a more efficient way to determine the values for your data structures, then you can implement this interface and provide the values directly.

See Also
`DomainInfo`.

13.25 Second

13.25.1 Overview

A subclass of `TimePeriod` that represents one second in a particular day. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

13.25.2 Constructors

To construct a `Second` instance:

`public Second(int second, Day day);`
Creates a new `Second` instance. The `second` argument should be in the range 1 to 24*60*60.

To construct a `Second` instance based on a `java.util.Date`:

`public Second(Date date);`
Creates a new `Second` instance.

A default constructor is provided:

`public Second();`
Creates a new `Second` instance based on the current system time.

13.25.3 Methods

To access the second:

`public int getSecond();`
Returns the second (in the range 1 to 24*60*60).

To access the day:

`public Day getDay();`
Returns the day.

There is no method to *set* the second or the day, because this class is immutable.

Given a `Second` object, you can create an instance representing the previous second or the next second:

`public TimePeriod previous();`
Returns the previous second, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next second, or null if the upper limit of the range is reached.

13.25.4 Notes

The `Second` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Day`.

13.26 SeriesChangeListener

The interface through which series change notifications are posted.

13.27 SeriesDataset

13.27.1 Overview

A base interface that defines a dataset containing zero, one or many data series.

13.27.2 Methods

The methods in the interface are:

`public int getSeriesCount();`
Returns the number of series in the dataset.

`public String getSeriesName(int series);`
Returns the name of the series with the specified index (zero based).

13.27.3 Notes

This interface is extended by `CategoryDataset` and `XYDataset`.

See Also:
`CategoryDataset`, `XYDataset`.

13.28 SeriesException

An exception generated by a series. For example, a time series will not allow duplicate time periods—attempting to add a duplicate time period will throw a `SeriesException`.

13.29 Statistics

13.29.1 Overview

Provides some static utility methods for calculating statistics.

13.29.2 Methods

To calculate the average of an array of `Number` objects:

`public static double getAverage(Number[] data);`
Returns the average of an array of numbers.

To calculate the standard deviation of an array of `Number` objects:

`public static double getStdDev(Number[] data);`
Returns the standard deviation of an array of numbers.

To calculate a least squares regression line through an array of data:

`public static double[] getLinearFit(Number[] x_data, Number[] y_data);`
Returns the intercept (`double[0]`) and slope (`double[1]`) of the linear regression line.

To calculate the slope of a least squares regression line:

`public static double getSlope(Number[] x_data, Number[] y_data);`
Returns the slope of the linear regression line.

To calculate the slope of a least squares regression line:

`public static double getCorrelation(Number[] data1, Number[] data2);`
Returns the correlation between two sets of numbers.

13.29.3 Notes

This class was written by Matthew Wright.

13.30 SubseriesDataset

A specialised dataset implementation written by Bill Kelemen. To be documented.

13.31 TimePeriod

13.31.1 Overview

An abstract class that represents a fixed period of time. This class and its subclasses are designed to be used with the `BasicTimeSeries` class.

13.31.2 Methods

Given a `TimePeriod` instance, you can create another instance representing the previous time period, or the next time period:

`public abstract TimePeriod previous();`
Returns the previous time period, or `null`.

`public abstract TimePeriod next();`
Returns the next time period, or `null`.

To get the millisecond at the start, middle and end of the time period (using the same encoding convention as `java.util.Date`):

`public long getStart();`
The first millisecond of the time period.

`public long getMiddle();`
The middle millisecond of the time period.

`public long getEnd();`
The last millisecond of the time period.

13.31.3 Notes

All `TimePeriod` subclasses are required to be immutable.

Known subclasses include: `Year`, `Quarter`, `Month`, `Week`, `Day`, `Hour`, `Minute`, `Second` and `Millisecond`.

See Also:
`BasicTimeSeries`.

13.32 TimePeriodFormatException

An exception that can be thrown by the methods used to convert time periods to strings, and vice versa.

See Also
`TimePeriod`

13.33 TimeSeriesCollection

13.33.1 Overview

A collection of `TimeSeries` objects. The collection may contain zero, one or many time series.

`TimeSeriesCollection` extends `AbstractSeriesDataset` to provide some of the basic series information.

The collection implements the `IntervalXYDataset` interface (and, therefore, the `XYDataset` interface) and can be used as a convenient dataset for the JFreeChart library.

13.33.2 Constructors

You can construct a `TimeSeriesCollection` in several different ways:

`public TimeSeriesCollection();`
Creates a new time series collection, initially empty.

`public TimeSeriesCollection(BasicTimeSeries series);`
Creates a new time series collection, containing a single time series.

Once a collection has been constructed, you are free to add additional time series to the collection. There are not yet any methods for removing a series from a collection (possibly to be implemented in the future).

13.33.3 Methods

To find out how many time series objects are in the collection:

`public int getSeriesCount();`
Returns the number of time series objects in the collection.

To get a reference to a particular series:

`public BasicTimeSeries getSeries(int series);`
Returns a reference to a series in the collection.

To get the name of a series:

`public String getSeriesName(int series);`
Returns the name of a series in the collection. This method is provided for convenience.

To add a series to the collection:

`public void addSeries(BasicTimeSeries series);`
Adds the series to the collection. Registered listeners are notified that the collection has changed.

To get the number of items in a series:

`public int getItemCount(int series);`
Returns the number of items in a series. This method is part of the `XYDataset` interface.

13.33.4 Notes

This class implements the `XYDataset` and `IntervalXYDataset` interfaces.

See Also:
`AbstractSeriesDataset`, `BasicTimeSeries`, `XYDataset` and `IntervalXYDataset`.

13.34 TimeSeriesDataPair

Associates a numerical value with a time period. This class is used by the `TimeSeries` class.

There are a number of important features. First, the class implements the `Comparable` interface, allowing data items to be sorted into time order using standard Java API calls. Second, the instances of this class can be easily cloned. Third, the time period element is immutable, so that when a collection of objects is held in sorted order, the sorted property cannot inadvertently be broken.

See Also
`TimeSeries`

13.35 TimeSeriesTableModel

An initial attempt to display a time series in a `JTable`.

13.36 Values

An interface for accessing a set of values. This hasn't been used for anything yet...but the idea was to create a simple data structure that could be passed to a variety of statistical methods (for example, a method that calculates frequency distributions, returning an appropriate dataset for constructing a histogram). More work to be done...

13.37 Week

13.37.1 Overview

A subclass of `TimePeriod` that represents one week in a particular year. This class is designed to be used with the `BasicTimeSeries` class, but (hopefully) is general enough to be used in other situations.

13.37.2 Constructors

To construct a `Week` instance:

`public Week(int week, Year year);`
Creates a new `Week` instance. The `week` argument should be in the range 1 to 52.

`public Week(int week, int year);`
Creates a new `Week` instance.

To construct a `Week` instance based on a `java.util.Date`:

`public Week(Date time);`
Creates a new `Week` instance.

A default constructor is provided:

`public Week();`
Creates a new `Week` instance based on the current system time.

13.37.3 Methods

To access the week:

`public int getWeek();`
Returns the week (in the range 1 to 52).

To access the year:

`public Year getYear();`
Returns the year.

There is no method to *set* the week or the year, because this class is immutable.

Given a `Week` object, you can create an instance representing the previous week or the next week:

`public TimePeriod previous();`
Returns the previous week, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next week, or null if the upper limit of the range is reached.

To convert a `Week` object to a `String` object:

`public String toString();`
Returns a string representing the week.

13.37.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Week` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `BasicTimeSeries`, `Year`.

13.38 XYDatapair

Associates a numerical value with another numerical value. This class is analagous to the `TimeSeriesDataPair` class.

13.39 XYDataset

13.39.1 Overview

An interface that defines a collection of data in the form of *(x, y)* values. The dataset can consist of zero, one or many data series. Each series can have *(x, y)* values that are completely independent of the other series in the dataset.

13.39.2 Methods

The methods in the interface are:

`public int getItemCount(int series);`
Returns the number of data items in a series.

`public Number getXValue(int series, int item);`
Returns an x-value for a series.

`public Number getYValue(int series, int item);`
Returns a y-value for a series (possibly `null`).

13.39.3 Notes

JFreeChart uses this interface to obtain data for drawing charts.

See Also:
`SeriesDataset`, `DefaultXYDataset`, `IntervalXYDataset`.

13.40 XYSeries

A series of (x, y) data items. Analagous to the `TimeSeries` class.

13.41 XYSeriesCollection

A collection of `XYSeries` objects. This class implements the `XYDataset` interface, so can be used very conveniently with JFreeChart.

13.42 XYZDataset

A natural extension of the `XYDataset` interface.

13.43 Year

13.43.1 Overview

A subclass of `TimePeriod` that represents one year. This class is designed to be used with the `TimeSeries` class, but is (hopefully) general enough to be used in other situations.

13.43.2 Constructors

To construct a `Year` instance:

`public Year(int year);`
Creates a new `Year` instance. The `year` argument should be in the range 1900 to 9999.

To construct a `Year` instance based on a `java.util.Date`:

`public Year(Date time);`
Creates a new `Year` instance.

A default constructor is provided:

`public Year();`
Creates a new `Year` instance based on the current system time.

13.43.3 Methods

To access the year:

`public int getYear();`
Returns the year.

There is no method to *set* the year, because this class is immutable.

Given a `Year` object, you can create an instance representing the previous year or the next year:

`public TimePeriod previous();`
Returns the previous year, or null if the lower limit of the range is reached.

`public TimePeriod next();`
Returns the next year, or null if the upper limit of the range is reached.

To convert a `Year` object to a `String` object, or vice versa:

`public String toString();`
Returns a string representing the year.

`public static Year parseYear(String s) throws TimePeriodFormatException;`
Parses the string and, if possible, returns a `Year` object.

13.43.4 Notes

In the current implementation, the year can be in the range 1900 to 9999.

The `Year` class is immutable. This is a requirement for all `TimePeriod` subclasses.

See Also:
`TimePeriod`, `TimeSeries`.

A The GNU Lesser General Public Licence

A.1 Introduction

JFreeChart is licensed under the terms of the GNU Lesser General Public Licence (LGPL). The full text of this licence is reproduced in this appendix. You should read and understand this licence before using JFreeChart in your own projects.

If you are not familiar with the concept of open-source software, you can find out more at the following web-sites:

Organisation:	Description:
The Free Software Foundation	`http://www.fsf.org`
The Open Source Initiative	`http://www.opensource.org`

A.2 The Licence

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and

passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source

code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

* a) The modified work must itself be a software library. * b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change. * c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License. * d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law. If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

* a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

* b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

* c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

* d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

* e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

* b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS