```
/* ===========================================================
 * JFreeChart : a free chart library for the Java(tm) platform
 * ===========================================================
 *
 * (C) Copyright 2000-2006, by Object Refinery Limited and Contributors.
 *
 * Project Info:  http://www.jfree.org/jfreechart/index.html
 *
 * This library is free software; you can redistribute it and/or modify it
 * under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or
 * (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
 * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
 * USA.
 *
 * [Java is a trademark or registered trademark of Sun Microsystems, Inc.
 * in the United States and other countries.]
 *
 * ----------------
 * BarRenderer.java
 * ----------------
 * (C) Copyright 2002-2006, by Object Refinery Limited.
 *
 * Original Author:  David Gilbert (for Object Refinery Limited);
 * Contributor(s):   Christian W. Zuckschwerdt;
 *
 *
 *
 * Changes
 * -------
 * 14-Mar-2002 : Version 1 (DG);
 * 23-May-2002 : Added tooltip generator to renderer (DG);
 * 29-May-2002 : Moved tooltip generator to abstract super-class (DG);
 * 25-Jun-2002 : Changed constructor to protected and removed redundant
 *               code (DG);
 * 26-Jun-2002 : Added axis to initialise method, and record upper and lower
 *               clip values (DG);
 * 24-Sep-2002 : Added getLegendItem() method (DG);
 * 09-Oct-2002 : Modified constructor to include URL generator (DG);
 * 05-Nov-2002 : Base dataset is now TableDataset not CategoryDataset (DG);
 * 10-Jan-2003 : Moved get/setItemMargin() method up from subclasses (DG);
 * 17-Jan-2003 : Moved plot classes into a separate package (DG);
 * 25-Mar-2003 : Implemented Serializable (DG);
 * 01-May-2003 : Modified clipping to allow for dual axes and datasets (DG);
 * 12-May-2003 : Merged horizontal and vertical bar renderers (DG);
 * 12-Jun-2003 : Updates for item labels (DG);
 * 30-Jul-2003 : Modified entity constructor (CZ);
 * 02-Sep-2003 : Changed initialise method to fix bug 790407 (DG);
 * 16-Sep-2003 : Changed ChartRenderingInfo --> PlotRenderingInfo (DG);
 * 07-Oct-2003 : Added renderer state (DG);
 * 27-Oct-2003 : Merged drawHorizontalItem() and drawVerticalItem()
 *               methods (DG);
 * 28-Oct-2003 : Added support for gradient paint on bars (DG);
 * 14-Nov-2003 : Added 'maxBarWidth' attribute (DG);
 * 10-Feb-2004 : Small changes inside drawItem() method to ease cut-and-paste
 *               overriding (DG);
 * 19-Mar-2004 : Fixed bug introduced with separation of tool tip and item
 *               label generators.  Fixed equals() method (DG);
 * 11-May-2004 : Fix for null pointer exception (bug id 951127) (DG);
 * 05-Nov-2004 : Modified drawItem() signature (DG);
```

```
 * 26-Jan-2005 : Provided override for getLegendItem() method (DG);
 * 20-Apr-2005 : Generate legend labels, tooltips and URLs (DG);
 * 18-May-2005 : Added configurable base value (DG);
 * 09-Jun-2005 : Use addItemEntity() method from superclass (DG);
 * 01-Dec-2005 : Update legend item to use/not use outline (DG);
 * ------------: JFreeChart 1.0.0 ---------------------------------------------
 * 06-Dec-2005 : Fixed bug 1374222 (JDK 1.4 specific code) (DG);
 * 11-Jan-2006 : Fixed bug 1401856 (bad rendering for non-zero base) (DG);
 * 04-Aug-2006 : Fixed bug 1467706 (missing item labels for zero value
 *               bars) (DG);
 *
 */

package org.jfree.chart.renderer.category;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
import java.awt.GradientPaint;
import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.Shape;
import java.awt.Stroke;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.io.Serializable;

import org.jfree.chart.LegendItem;
import org.jfree.chart.axis.CategoryAxis;
import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.entity.EntityCollection;
import org.jfree.chart.event.RendererChangeEvent;
import org.jfree.chart.labels.CategoryItemLabelGenerator;
import org.jfree.chart.labels.ItemLabelAnchor;
import org.jfree.chart.labels.ItemLabelPosition;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.PlotRenderingInfo;
import org.jfree.data.Range;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.general.DatasetUtilities;
import org.jfree.text.TextUtilities;
import org.jfree.ui.GradientPaintTransformer;
import org.jfree.ui.RectangleEdge;
import org.jfree.ui.StandardGradientPaintTransformer;
import org.jfree.util.ObjectUtilities;
import org.jfree.util.PublicCloneable;

/**
 * A {@link CategoryItemRenderer} that draws individual data items as bars.
 */
public class BarRenderer extends AbstractCategoryItemRenderer
                         implements Cloneable, PublicCloneable, Serializable {

    /** For serialization. */
    private static final long serialVersionUID = 6000649414965887481L;

    /** The default item margin percentage. */
    public static final double DEFAULT_ITEM_MARGIN = 0.20;

    /**
     * Constant that controls the minimum width before a bar has an outline
     * drawn.
     */
    public static final double BAR_OUTLINE_WIDTH_THRESHOLD = 3.0;

    /** The margin between items (bars) within a category. */
```

```
0137      private double itemMargin;
0138
0139      /** A flag that controls whether or not bar outlines are drawn. */
0140      private boolean drawBarOutline;
0141
0142      /** The maximum bar width as a percentage of the available space. */
0143      private double maximumBarWidth;
0144
0145      /** The minimum bar length (in Java2D units). */
0146      private double minimumBarLength;
0147
0148      /**
0149       * An optional class used to transform gradient paint objects to fit each
0150       * bar.
0151       */
0152      private GradientPaintTransformer gradientPaintTransformer;
0153
0154      /**
0155       * The fallback position if a positive item label doesn't fit inside the
0156       * bar.
0157       */
0158      private ItemLabelPosition positiveItemLabelPositionFallback;
0159
0160      /**
0161       * The fallback position if a negative item label doesn't fit inside the
0162       * bar.
0163       */
0164      private ItemLabelPosition negativeItemLabelPositionFallback;
0165
0166      /** The upper clip (axis) value for the axis. */
0167      private double upperClip;
0168      // TODO:  this needs to move into the renderer state
0169
0170      /** The lower clip (axis) value for the axis. */
0171      private double lowerClip;
0172      // TODO:  this needs to move into the renderer state
0173
0174      /** The base value for the bars (defaults to 0.0). */
0175      private double base;
0176
0177      /**
0178       * A flag that controls whether the base value is included in the range
0179       * returned by the findRangeBounds() method.
0180       */
0181      private boolean includeBaseInRange;
0182
0183      /**
0184       * Creates a new bar renderer with default settings.
0185       */
0186      public BarRenderer() {
0187          super();
0188          this.base = 0.0;
0189          this.includeBaseInRange = true;
0190          this.itemMargin = DEFAULT_ITEM_MARGIN;
0191          this.drawBarOutline = true;
0192          this.maximumBarWidth = 1.0;
0193              // 100 percent, so it will not apply unless changed
0194          this.positiveItemLabelPositionFallback = null;
0195          this.negativeItemLabelPositionFallback = null;
0196          this.gradientPaintTransformer = new StandardGradientPaintTransformer();
0197          this.minimumBarLength = 0.0;
0198      }
0199
0200      /**
0201       * Returns the base value for the bars.
0202       *
0203       * @return The base value for the bars.
0204       */
```



```
0137      private double itemMargin;
0138
0139      /** A flag that controls whether or not bar outlines are drawn. */
0140      private boolean drawBarOutline;
0141
0142      /** The maximum bar width as a percentage of the available space. */
0143      private double maximumBarWidth;
0144
0145      /** The minimum bar length (in Java2D units). */
0146      private double minimumBarLength;
0147
0148      /**
0149       * An optional class used to transform gradient paint objects to fit each
0150       * bar.
0151       */
0152      private GradientPaintTransformer gradientPaintTransformer;
0153
0154      /**
0155       * The fallback position if a positive item label doesn't fit inside the
0156       * bar.
0157       */
0158      private ItemLabelPosition positiveItemLabelPositionFallback;
0159
0160      /**
0161       * The fallback position if a negative item label doesn't fit inside the
0162       * bar.
0163       */
0164      private ItemLabelPosition negativeItemLabelPositionFallback;
0165
0166      /** The upper clip (axis) value for the axis. */
0167      private double upperClip;
0168      // TODO:  this needs to move into the renderer state
0169
0170      /** The lower clip (axis) value for the axis. */
0171      private double lowerClip;
0172      // TODO:  this needs to move into the renderer state
0173
0174      /** The base value for the bars (defaults to 0.0). */
0175      private double base;
0176
0177      /**
0178       * A flag that controls whether the base value is included in the range
0179       * returned by the findRangeBounds() method.
0180       */
0181      private boolean includeBaseInRange;
0182
0183      /**
0184       * Creates a new bar renderer with default settings.
0185       */
0186      public BarRenderer() {
0187          super();
0188          this.base = 0.0;
0189          this.includeBaseInRange = true;
0190          this.itemMargin = DEFAULT_ITEM_MARGIN;
0191          this.drawBarOutline = true;
0192          this.maximumBarWidth = 1.0;
0193              // 100 percent, so it will not apply unless changed
0194          this.positiveItemLabelPositionFallback = null;
0195          this.negativeItemLabelPositionFallback = null;
0196          this.gradientPaintTransformer = new StandardGradientPaintTransformer();
0197          this.minimumBarLength = 0.0;
0198      }
0199
0200      /**
0201       * Returns the base value for the bars.
0202       *
0203       * @return The base value for the bars.
0204       */
```

```
0205      public double getBase() {
0206          return this.base;
0207      }
0208
0209      /**
0210       * Sets the base value for the bars and sends a {@link RendererChangeEvent}
0211       * to all registered listeners.
0212       *
0213       * @param base  the new base value.
0214       */
0215      public void setBase(double base) {
0216          this.base = base;
0217          notifyListeners(new RendererChangeEvent(this));
0218      }
0219
0220      /**
0221       * Returns the item margin as a percentage of the available space for all
0222       * bars.
0223       *
0224       * @return The margin percentage (where 0.10 is ten percent).
0225       */
0226      public double getItemMargin() {
0227          return this.itemMargin;
0228      }
0229
0230      /**
0231       * Sets the item margin and sends a {@link RendererChangeEvent} to all
0232       * registered listeners.  The value is expressed as a percentage of the
0233       * available width for plotting all the bars, with the resulting amount to
0234       * be distributed between all the bars evenly.
0235       *
0236       * @param percent  the margin (where 0.10 is ten percent).
0237       */
0238      public void setItemMargin(double percent) {
0239          this.itemMargin = percent;
0240          notifyListeners(new RendererChangeEvent(this));
0241      }
0242
0243      /**
0244       * Returns a flag that controls whether or not bar outlines are drawn.
0245       *
0246       * @return A boolean.
0247       */
0248      public boolean isDrawBarOutline() {
0249          return this.drawBarOutline;
0250      }
0251
0252      /**
0253       * Sets the flag that controls whether or not bar outlines are drawn and
0254       * sends a {@link RendererChangeEvent} to all registered listeners.
0255       *
0256       * @param draw  the flag.
0257       */
0258      public void setDrawBarOutline(boolean draw) {
0259          this.drawBarOutline = draw;
0260          notifyListeners(new RendererChangeEvent(this));
0261      }
0262
0263      /**
0264       * Returns the maximum bar width, as a percentage of the available drawing
0265       * space.
0266       *
0267       * @return The maximum bar width.
0268       */
0269      public double getMaximumBarWidth() {
0270          return this.maximumBarWidth;
0271      }
0272
```



```
0205      public double getBase() {
0206          return this.base;
0207      }
0208
0209      /**
0210       * Sets the base value for the bars and sends a {@link RendererChangeEvent}
0211       * to all registered listeners.
0212       *
0213       * @param base  the new base value.
0214       */
0215      public void setBase(double base) {
0216          this.base = base;
0217          notifyListeners(new RendererChangeEvent(this));
0218      }
0219
0220      /**
0221       * Returns the item margin as a percentage of the available space for all
0222       * bars.
0223       *
0224       * @return The margin percentage (where 0.10 is ten percent).
0225       */
0226      public double getItemMargin() {
0227          return this.itemMargin;
0228      }
0229
0230      /**
0231       * Sets the item margin and sends a {@link RendererChangeEvent} to all
0232       * registered listeners.  The value is expressed as a percentage of the
0233       * available width for plotting all the bars, with the resulting amount to
0234       * be distributed between all the bars evenly.
0235       *
0236       * @param percent  the margin (where 0.10 is ten percent).
0237       */
0238      public void setItemMargin(double percent) {
0239          this.itemMargin = percent;
0240          notifyListeners(new RendererChangeEvent(this));
0241      }
0242
0243      /**
0244       * Returns a flag that controls whether or not bar outlines are drawn.
0245       *
0246       * @return A boolean.
0247       */
0248      public boolean isDrawBarOutline() {
0249          return this.drawBarOutline;
0250      }
0251
0252      /**
0253       * Sets the flag that controls whether or not bar outlines are drawn and
0254       * sends a {@link RendererChangeEvent} to all registered listeners.
0255       *
0256       * @param draw  the flag.
0257       */
0258      public void setDrawBarOutline(boolean draw) {
0259          this.drawBarOutline = draw;
0260          notifyListeners(new RendererChangeEvent(this));
0261      }
0262
0263      /**
0264       * Returns the maximum bar width, as a percentage of the available drawing
0265       * space.
0266       *
0267       * @return The maximum bar width.
0268       */
0269      public double getMaximumBarWidth() {
0270          return this.maximumBarWidth;
0271      }
0272
```

```
0273      /**
0274       * Sets the maximum bar width, which is specified as a percentage of the
0275       * available space for all bars, and sends a {@link RendererChangeEvent} to
0276       * all registered listeners.
0277       *
0278       * @param percent  the percent (where 0.05 is five percent).
0279       */
0280      public void setMaximumBarWidth(double percent) {
0281          this.maximumBarWidth = percent;
0282          notifyListeners(new RendererChangeEvent(this));
0283      }
0284
0285      /**
0286       * Returns the minimum bar length (in Java2D units).
0287       *
0288       * @return The minimum bar length.
0289       */
0290      public double getMinimumBarLength() {
0291          return this.minimumBarLength;
0292      }
0293
0294      /**
0295       * Sets the minimum bar length and sends a {@link RendererChangeEvent} to
0296       * all registered listeners.  The minimum bar length is specified in Java2D
0297       * units, and can be used to prevent bars that represent very small data
0298       * values from disappearing when drawn on the screen.
0299       *
0300       * @param min  the minimum bar length (in Java2D units).
0301       */
0302      public void setMinimumBarLength(double min) {
0303          this.minimumBarLength = min;
0304          notifyListeners(new RendererChangeEvent(this));
0305      }
0306
0307      /**
0308       * Returns the gradient paint transformer (an object used to transform
0309       * gradient paint objects to fit each bar.
0310       *
0311       * @return A transformer (<code>null</code> possible).
0312       */
0313      public GradientPaintTransformer getGradientPaintTransformer() {
0314          return this.gradientPaintTransformer;
0315      }
0316
0317      /**
0318       * Sets the gradient paint transformer and sends a
0319       * {@link RendererChangeEvent} to all registered listeners.
0320       *
0321       * @param transformer  the transformer (<code>null</code> permitted).
0322       */
0323      public void setGradientPaintTransformer(
0324              GradientPaintTransformer transformer) {
0325          this.gradientPaintTransformer = transformer;
0326          notifyListeners(new RendererChangeEvent(this));
0327      }
0328
0329      /**
0330       * Returns the fallback position for positive item labels that don't fit
0331       * within a bar.
0332       *
0333       * @return The fallback position (<code>null</code> possible).
0334       */
0335      public ItemLabelPosition getPositiveItemLabelPositionFallback() {
0336          return this.positiveItemLabelPositionFallback;
0337      }
0338
0339      /**
0340       * Sets the fallback position for positive item labels that don't fit
```

```
0341         * within a bar, and sends a {@link RendererChangeEvent} to all registered
0342         * listeners.
0343         *
0344         * @param position  the position (<code>null</code> permitted).
0345         */
0346        public void setPositiveItemLabelPositionFallback(
0347                ItemLabelPosition position) {
0348            this.positiveItemLabelPositionFallback = position;
0349            notifyListeners(new RendererChangeEvent(this));
0350        }
0351
0352        /**
0353         * Returns the fallback position for negative item labels that don't fit
0354         * within a bar.
0355         *
0356         * @return The fallback position (<code>null</code> possible).
0357         */
0358        public ItemLabelPosition getNegativeItemLabelPositionFallback() {
0359            return this.negativeItemLabelPositionFallback;
0360        }
0361
0362        /**
0363         * Sets the fallback position for negative item labels that don't fit
0364         * within a bar, and sends a {@link RendererChangeEvent} to all registered
0365         * listeners.
0366         *
0367         * @param position  the position (<code>null</code> permitted).
0368         */
0369        public void setNegativeItemLabelPositionFallback(
0370                ItemLabelPosition position) {
0371            this.negativeItemLabelPositionFallback = position;
0372            notifyListeners(new RendererChangeEvent(this));
0373        }
0374
0375        /**
0376         * Returns the flag that controls whether or not the base value for the
0377         * bars is included in the range calculated by
0378         * {@link #findRangeBounds(CategoryDataset)}.
0379         *
0380         * @return <code>true</code> if the base is included in the range, and
0381         *         <code>false</code> otherwise.
0382         *
0383         * @since 1.0.1
0384         */
0385        public boolean getIncludeBaseInRange() {
0386            return this.includeBaseInRange;
0387        }
0388
0389        /**
0390         * Sets the flag that controls whether or not the base value for the bars
0391         * is included in the range calculated by
0392         * {@link #findRangeBounds(CategoryDataset)}.  If the flag is changed,
0393         * a {@link RendererChangeEvent} is sent to all registered listeners.
0394         *
0395         * @param include  the new value for the flag.
0396         *
0397         * @since 1.0.1
0398         */
0399        public void setIncludeBaseInRange(boolean include) {
0400            if (this.includeBaseInRange != include) {
0401                this.includeBaseInRange = include;
0402                notifyListeners(new RendererChangeEvent(this));
0403            }
0404        }
0405
0406        /**
0407         * Returns the lower clip value.  This value is recalculated in the
0408         * initialise() method.
```



```
0341         * within a bar, and sends a {@link RendererChangeEvent} to all registered
0342         * listeners.
0343         *
0344         * @param position  the position (<code>null</code> permitted).
0345         */
0346        public void setPositiveItemLabelPositionFallback(
0347                ItemLabelPosition position) {
0348            this.positiveItemLabelPositionFallback = position;
0349            notifyListeners(new RendererChangeEvent(this));
0350        }
0351
0352        /**
0353         * Returns the fallback position for negative item labels that don't fit
0354         * within a bar.
0355         *
0356         * @return The fallback position (<code>null</code> possible).
0357         */
0358        public ItemLabelPosition getNegativeItemLabelPositionFallback() {
0359            return this.negativeItemLabelPositionFallback;
0360        }
0361
0362        /**
0363         * Sets the fallback position for negative item labels that don't fit
0364         * within a bar, and sends a {@link RendererChangeEvent} to all registered
0365         * listeners.
0366         *
0367         * @param position  the position (<code>null</code> permitted).
0368         */
0369        public void setNegativeItemLabelPositionFallback(
0370                ItemLabelPosition position) {
0371            this.negativeItemLabelPositionFallback = position;
0372            notifyListeners(new RendererChangeEvent(this));
0373        }
0374
0375        /**
0376         * Returns the flag that controls whether or not the base value for the
0377         * bars is included in the range calculated by
0378         * {@link #findRangeBounds(CategoryDataset)}.
0379         *
0380         * @return <code>true</code> if the base is included in the range, and
0381         *         <code>false</code> otherwise.
0382         *
0383         * @since 1.0.1
0384         */
0385        public boolean getIncludeBaseInRange() {
0386            return this.includeBaseInRange;
0387        }
0388
0389        /**
0390         * Sets the flag that controls whether or not the base value for the bars
0391         * is included in the range calculated by
0392         * {@link #findRangeBounds(CategoryDataset)}.  If the flag is changed,
0393         * a {@link RendererChangeEvent} is sent to all registered listeners.
0394         *
0395         * @param include  the new value for the flag.
0396         *
0397         * @since 1.0.1
0398         */
0399        public void setIncludeBaseInRange(boolean include) {
0400            if (this.includeBaseInRange != include) {
0401                this.includeBaseInRange = include;
0402                notifyListeners(new RendererChangeEvent(this));
0403            }
0404        }
0405
0406        /**
0407         * Returns the lower clip value.  This value is recalculated in the
0408         * initialise() method.
```

```
0409         *
0410         * @return The value.
0411         */
0412        public double getLowerClip() {
0413            // TODO:  this attribute should be transferred to the renderer state.
0414            return this.lowerClip;
0415        }
0416
0417        /**
0418         * Returns the upper clip value.  This value is recalculated in the
0419         * initialise() method.
0420         *
0421         * @return The value.
0422         */
0423        public double getUpperClip() {
0424            // TODO:  this attribute should be transferred to the renderer state.
0425            return this.upperClip;
0426        }
0427
0428        /**
0429         * Initialises the renderer and returns a state object that will be passed
0430         * to subsequent calls to the drawItem method.  This method gets called
0431         * once at the start of the process of drawing a chart.
0432         *
0433         * @param g2  the graphics device.
0434         * @param dataArea  the area in which the data is to be plotted.
0435         * @param plot  the plot.
0436         * @param rendererIndex  the renderer index.
0437         * @param info  collects chart rendering information for return to caller.
0438         *
0439         * @return The renderer state.
0440         */
0441        public CategoryItemRendererState initialise(Graphics2D g2,
0442                                                    Rectangle2D dataArea,
0443                                                    CategoryPlot plot,
0444                                                    int rendererIndex,
0445                                                    PlotRenderingInfo info) {
0446
0447            CategoryItemRendererState state = super.initialise(g2, dataArea, plot,
0448                    rendererIndex, info);
0449
0450            // get the clipping values...
0451            ValueAxis rangeAxis = getRangeAxis(plot, rendererIndex);
0452            this.lowerClip = rangeAxis.getRange().getLowerBound();
0453            this.upperClip = rangeAxis.getRange().getUpperBound();
0454
0455            // calculate the bar width
0456            calculateBarWidth(plot, dataArea, rendererIndex, state);
0457
0458            return state;
0459
0460        }
0461
0462        /**
0463         * Calculates the bar width and stores it in the renderer state.
0464         *
0465         * @param plot  the plot.
0466         * @param dataArea  the data area.
0467         * @param rendererIndex  the renderer index.
0468         * @param state  the renderer state.
0469         */
0470        protected void calculateBarWidth(CategoryPlot plot,
0471                                         Rectangle2D dataArea,
0472                                         int rendererIndex,
0473                                         CategoryItemRendererState state) {
0474
0475            CategoryAxis domainAxis = getDomainAxis(plot, rendererIndex);
0476            CategoryDataset dataset = plot.getDataset(rendererIndex);
```



```
0409         *
0410         * @return The value.
0411         */
0412        public double getLowerClip() {
0413            // TODO:  this attribute should be transferred to the renderer state.
0414            return this.lowerClip;
0415        }
0416
0417        /**
0418         * Returns the upper clip value.  This value is recalculated in the
0419         * initialise() method.
0420         *
0421         * @return The value.
0422         */
0423        public double getUpperClip() {
0424            // TODO:  this attribute should be transferred to the renderer state.
0425            return this.upperClip;
0426        }
0427
0428        /**
0429         * Initialises the renderer and returns a state object that will be passed
0430         * to subsequent calls to the drawItem method.  This method gets called
0431         * once at the start of the process of drawing a chart.
0432         *
0433         * @param g2  the graphics device.
0434         * @param dataArea  the area in which the data is to be plotted.
0435         * @param plot  the plot.
0436         * @param rendererIndex  the renderer index.
0437         * @param info  collects chart rendering information for return to caller.
0438         *
0439         * @return The renderer state.
0440         */
0441        public CategoryItemRendererState initialise(Graphics2D g2,
0442                                                    Rectangle2D dataArea,
0443                                                    CategoryPlot plot,
0444                                                    int rendererIndex,
0445                                                    PlotRenderingInfo info) {
0446
0447            CategoryItemRendererState state = super.initialise(g2, dataArea, plot,
0448                    rendererIndex, info);
0449
0450            // get the clipping values...
0451            ValueAxis rangeAxis = getRangeAxis(plot, rendererIndex);
0452            this.lowerClip = rangeAxis.getRange().getLowerBound();
0453            this.upperClip = rangeAxis.getRange().getUpperBound();
0454
0455            // calculate the bar width
0456            calculateBarWidth(plot, dataArea, rendererIndex, state);
0457
0458            return state;
0459
0460        }
0461
0462        /**
0463         * Calculates the bar width and stores it in the renderer state.
0464         *
0465         * @param plot  the plot.
0466         * @param dataArea  the data area.
0467         * @param rendererIndex  the renderer index.
0468         * @param state  the renderer state.
0469         */
0470        protected void calculateBarWidth(CategoryPlot plot,
0471                                         Rectangle2D dataArea,
0472                                         int rendererIndex,
0473                                         CategoryItemRendererState state) {
0474
0475            CategoryAxis domainAxis = getDomainAxis(plot, rendererIndex);
0476            CategoryDataset dataset = plot.getDataset(rendererIndex);
```

```
0477          if (dataset != null) {
0478              int columns = dataset.getColumnCount();
0479              int rows = dataset.getRowCount();
0480              double space = 0.0;
0481              PlotOrientation orientation = plot.getOrientation();
0482              if (orientation == PlotOrientation.HORIZONTAL) {
0483                  space = dataArea.getHeight();
0484              }
0485              else if (orientation == PlotOrientation.VERTICAL) {
0486                  space = dataArea.getWidth();
0487              }
0488              double maxWidth = space * getMaximumBarWidth();
0489              double categoryMargin = 0.0;
0490              double currentItemMargin = 0.0;
0491              if (columns > 1) {
0492                  categoryMargin = domainAxis.getCategoryMargin();
0493              }
0494              if (rows > 1) {
0495                  currentItemMargin = getItemMargin();
0496              }
0497              double used = space * (1 - domainAxis.getLowerMargin()
0498                                       - domainAxis.getUpperMargin()
0499                                       - categoryMargin - currentItemMargin);
0500              if ((rows * columns) > 0) {
0501                  state.setBarWidth(Math.min(used / (rows * columns), maxWidth));
0502              }
0503              else {
0504                  state.setBarWidth(Math.min(used, maxWidth));
0505              }
0506          }
0507      }
0508
0509      /**
0510       * Calculates the coordinate of the first "side" of a bar.  This will be
0511       * the minimum x-coordinate for a vertical bar, and the minimum
0512       * y-coordinate for a horizontal bar.
0513       *
0514       * @param plot  the plot.
0515       * @param orientation  the plot orientation.
0516       * @param dataArea  the data area.
0517       * @param domainAxis  the domain axis.
0518       * @param state  the renderer state (has the bar width precalculated).
0519       * @param row  the row index.
0520       * @param column  the column index.
0521       *
0522       * @return The coordinate.
0523       */
0524      protected double calculateBarW0(CategoryPlot plot,
0525                                      PlotOrientation orientation,
0526                                      Rectangle2D dataArea,
0527                                      CategoryAxis domainAxis,
0528                                      CategoryItemRendererState state,
0529                                      int row,
0530                                      int column) {
0531          // calculate bar width...
0532          double space = 0.0;
0533          if (orientation == PlotOrientation.HORIZONTAL) {
0534              space = dataArea.getHeight();
0535          }
0536          else {
0537              space = dataArea.getWidth();
0538          }
0539          double barW0 = domainAxis.getCategoryStart(column, getColumnCount(),
0540                  dataArea, plot.getDomainAxisEdge());
0541          int seriesCount = getRowCount();
0542          int categoryCount = getColumnCount();
0543          if (seriesCount > 1) {
0544              double seriesGap = space * getItemMargin()
```

C:/jfreechart-1.0.2/jfreechart-1.0.2/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 545

```
0545                                / (categoryCount * (seriesCount - 1));
0546            double seriesW = calculateSeriesWidth(space, domainAxis,
0547                    categoryCount, seriesCount);
0548            barW0 = barW0 + row * (seriesW + seriesGap)
0549                          + (seriesW / 2.0) - (state.getBarWidth() / 2.0);
0550        }
0551        else {
0552            barW0 = domainAxis.getCategoryMiddle(column, getColumnCount(),
0553                    dataArea, plot.getDomainAxisEdge()) - state.getBarWidth()
0554                    / 2.0;
0555        }
0556        return barW0;
0557    }
0558
0559    /**
0560     * Calculates the coordinates for the length of a single bar.
0561     *
0562     * @param value  the value represented by the bar.
0563     *
0564     * @return The coordinates for each end of the bar (or <code>null</code> if
0565     *         the bar is not visible for the current axis range).
```

C:/jfreechart-1.0.2/jfreechart-1.0.2-modified/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 545

```
0545                                / (categoryCount * (seriesCount - 1));
0546            double seriesW = calculateSeriesWidth(space, domainAxis,
0547                    categoryCount, seriesCount);
0548            barW0 = barW0 + row * (seriesW + seriesGap)
0549                          + (seriesW / 2.0) - (state.getBarWidth() / 2.0);
0550        }
0551        else {
0552            barW0 = domainAxis.getCategoryMiddle(column, getColumnCount(),
0553                    dataArea, plot.getDomainAxisEdge()) - state.getBarWidth()
0554                    / 2.0;
0555        }
0556        return barW0;
0557    }
0558
0559    /**
0560     * Calculates the coordinate of the first "side" of a bar.  This will be
0561     * the minimum x-coordinate for a vertical bar, and the minimum
0562     * y-coordinate for a horizontal bar.
0563     *
0564     * @param plot        the plot.
0565     * @param orientation the plot orientation.
0566     * @param dataArea    the data area.
0567     * @param domainAxis  the domain axis.
0568     * @param state       the renderer state (has the bar width precalculated).
0569     * @param row         the row index.
0570     * @param column      the column index.
0571     * @return The coordinate.
0572     */
0573    protected double calculateSummaryTaskBarW0(CategoryPlot plot,
0574                                               PlotOrientation orientation,
0575                                               Rectangle2D dataArea,
0576                                               CategoryAxis domainAxis,
0577                                               CategoryItemRendererState state,
0578                                               int row,
0579                                               int column) {
0580        // calculate bar width...
0581        double space = 0.0;
0582        if (orientation == PlotOrientation.HORIZONTAL) {
0583            space = dataArea.getHeight();
0584        } else {
0585            space = dataArea.getWidth();
0586        }
0587        double barW0 = domainAxis.getCategoryStart(column, getColumnCount(),
0588                dataArea, plot.getDomainAxisEdge());
0589        int seriesCount = getRowCount();
0590        int categoryCount = getColumnCount();
0591        if (seriesCount > 1) {
0592            double seriesGap = space * getItemMargin()
0593                    / (categoryCount * (seriesCount - 1));
0594            double seriesW = calculateSeriesWidth(space, domainAxis,
0595                    categoryCount, seriesCount);
0596            barW0 = barW0 + row * (seriesW + seriesGap) + seriesW
0597                    - (state.getBarWidth() / 4.0);
0598        } else {
0599            barW0 = domainAxis.getCategoryMiddle(column, getColumnCount(),
0600                    dataArea, plot.getDomainAxisEdge()) - state.getBarWidth()
0601                    / 4.0;
0602        }
0603        return barW0;
0604    }
0605
0606    /**
0607     * Calculates the coordinates for the length of a single bar.
0608     *
0609     * @param value  the value represented by the bar.
0610     *
0611     * @return The coordinates for each end of the bar (or <code>null</code> if
0612     *         the bar is not visible for the current axis range).
```

C:/jfreechart-1.0.2/jfreechart-1.0.2/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 566

```
0566      */
0567     protected double[] calculateBarL0L1(double value) {
0568         double lclip = getLowerClip();
0569         double uclip = getUpperClip();
0570         double barLow = Math.min(this.base, value);
0571         double barHigh = Math.max(this.base, value);
0572         if (barHigh < lclip) {  // bar is not visible
0573             return null;
0574         }
0575         if (barLow > uclip) {   // bar is not visible
0576             return null;
0577         }
0578         barLow = Math.max(barLow, lclip);
0579         barHigh = Math.min(barHigh, uclip);
0580         return new double[] {barLow, barHigh};
0581     }
0582
0583     /**
0584      * Returns the range of values the renderer requires to display all the
0585      * items from the specified dataset.  This takes into account the range
0586      * of values in the dataset, plus the flag that determines whether or not
0587      * the base value for the bars should be included in the range.
0588      *
0589      * @param dataset  the dataset (<code>null</code> permitted).
0590      *
0591      * @return The range (or <code>null</code> if the dataset is
0592      *         <code>null</code> or empty).
0593      */
0594     public Range findRangeBounds(CategoryDataset dataset) {
0595         Range result = DatasetUtilities.findRangeBounds(dataset);
0596         if (result != null) {
0597             if (this.includeBaseInRange) {
0598                 result = Range.expandToInclude(result, this.base);
0599             }
0600         }
0601         return result;
0602     }
0603
0604     /**
0605      * Returns a legend item for a series.
0606      *
0607      * @param datasetIndex  the dataset index (zero-based).
0608      * @param series  the series index (zero-based).
0609      *
0610      * @return The legend item.
0611      */
0612     public LegendItem getLegendItem(int datasetIndex, int series) {
0613
0614         CategoryPlot cp = getPlot();
0615         if (cp == null) {
0616             return null;
0617         }
0618
0619         CategoryDataset dataset;
0620         dataset = cp.getDataset(datasetIndex);
0621         String label = getLegendItemLabelGenerator().generateLabel(dataset,
0622                 series);
0623         String description = label;
0624         String toolTipText = null;
0625         if (getLegendItemToolTipGenerator() != null) {
0626             toolTipText = getLegendItemToolTipGenerator().generateLabel(
0627                     dataset, series);
0628         }
0629         String urlText = null;
0630         if (getLegendItemURLGenerator() != null) {
0631             urlText = getLegendItemURLGenerator().generateLabel(dataset,
0632                     series);
0633         }
```

C:/jfreechart-1.0.2/jfreechart-1.0.2-modified/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 613

```
0613      */
0614     protected double[] calculateBarL0L1(double value) {
0615         double lclip = getLowerClip();
0616         double uclip = getUpperClip();
0617         double barLow = Math.min(this.base, value);
0618         double barHigh = Math.max(this.base, value);
0619         if (barHigh < lclip) {  // bar is not visible
0620             return null;
0621         }
0622         if (barLow > uclip) {   // bar is not visible
0623             return null;
0624         }
0625         barLow = Math.max(barLow, lclip);
0626         barHigh = Math.min(barHigh, uclip);
0627         return new double[] {barLow, barHigh};
0628     }
0629
0630     /**
0631      * Returns the range of values the renderer requires to display all the
0632      * items from the specified dataset.  This takes into account the range
0633      * of values in the dataset, plus the flag that determines whether or not
0634      * the base value for the bars should be included in the range.
0635      *
0636      * @param dataset  the dataset (<code>null</code> permitted).
0637      *
0638      * @return The range (or <code>null</code> if the dataset is
0639      *         <code>null</code> or empty).
0640      */
0641     public Range findRangeBounds(CategoryDataset dataset) {
0642         Range result = DatasetUtilities.findRangeBounds(dataset);
0643         if (result != null) {
0644             if (this.includeBaseInRange) {
0645                 result = Range.expandToInclude(result, this.base);
0646             }
0647         }
0648         return result;
0649     }
0650
0651     /**
0652      * Returns a legend item for a series.
0653      *
0654      * @param datasetIndex  the dataset index (zero-based).
0655      * @param series  the series index (zero-based).
0656      *
0657      * @return The legend item.
0658      */
0659     public LegendItem getLegendItem(int datasetIndex, int series) {
0660
0661         CategoryPlot cp = getPlot();
0662         if (cp == null) {
0663             return null;
0664         }
0665
0666         CategoryDataset dataset;
0667         dataset = cp.getDataset(datasetIndex);
0668         String label = getLegendItemLabelGenerator().generateLabel(dataset,
0669                 series);
0670         String description = label;
0671         String toolTipText = null;
0672         if (getLegendItemToolTipGenerator() != null) {
0673             toolTipText = getLegendItemToolTipGenerator().generateLabel(
0674                     dataset, series);
0675         }
0676         String urlText = null;
0677         if (getLegendItemURLGenerator() != null) {
0678             urlText = getLegendItemURLGenerator().generateLabel(dataset,
0679                     series);
0680         }
```

```
0634            Shape shape = new Rectangle2D.Double(-4.0, -4.0, 8.0, 8.0);
0635            Paint paint = getSeriesPaint(series);
0636            Paint outlinePaint = getSeriesOutlinePaint(series);
0637            Stroke outlineStroke = getSeriesOutlineStroke(series);
0638
0639            return new LegendItem(label, description, toolTipText, urlText,
0640                    true, shape, true, paint,
0641                    isDrawBarOutline(), outlinePaint, outlineStroke,
0642                    false, new Line2D.Float(), new BasicStroke(1.0f),
0643                    Color.black);
0644        }
0645
0646        /**
0647         * Draws the bar for a single (series, category) data item.
0648         *
0649         * @param g2  the graphics device.
0650         * @param state  the renderer state.
0651         * @param dataArea  the data area.
0652         * @param plot  the plot.
0653         * @param domainAxis  the domain axis.
0654         * @param rangeAxis  the range axis.
0655         * @param dataset  the dataset.
0656         * @param row  the row index (zero-based).
0657         * @param column  the column index (zero-based).
0658         * @param pass  the pass index.
0659         */
0660        public void drawItem(Graphics2D g2,
0661                             CategoryItemRendererState state,
0662                             Rectangle2D dataArea,
0663                             CategoryPlot plot,
0664                             CategoryAxis domainAxis,
0665                             ValueAxis rangeAxis,
0666                             CategoryDataset dataset,
0667                             int row,
0668                             int column,
0669                             int pass) {
0670
0671            // nothing is drawn for null values...
0672            Number dataValue = dataset.getValue(row, column);
0673            if (dataValue == null) {
0674                return;
0675            }
0676
0677            double value = dataValue.doubleValue();
0678
0679            PlotOrientation orientation = plot.getOrientation();
0680            double barW0 = calculateBarW0(plot, orientation, dataArea, domainAxis,
0681                    state, row, column);
0682            double[] barL0L1 = calculateBarL0L1(value);
0683            if (barL0L1 == null) {
0684                return;  // the bar is not visible
0685            }
0686
0687            RectangleEdge edge = plot.getRangeAxisEdge();
0688            double transL0 = rangeAxis.valueToJava2D(barL0L1[0], dataArea, edge);
0689            double transL1 = rangeAxis.valueToJava2D(barL0L1[1], dataArea, edge);
0690            double barL0 = Math.min(transL0, transL1);
0691            double barLength = Math.max(Math.abs(transL1 - transL0),
0692                    getMinimumBarLength());
0693
0694            // draw the bar...
0695            Rectangle2D bar = null;
0696            if (orientation == PlotOrientation.HORIZONTAL) {
0697                bar = new Rectangle2D.Double(barL0, barW0, barLength,
0698                        state.getBarWidth());
0699            }
0700            else {
0701                bar = new Rectangle2D.Double(barW0, barL0, state.getBarWidth(),
```



```
0681            Shape shape = new Rectangle2D.Double(-4.0, -4.0, 8.0, 8.0);
0682            Paint paint = getSeriesPaint(series);
0683            Paint outlinePaint = getSeriesOutlinePaint(series);
0684            Stroke outlineStroke = getSeriesOutlineStroke(series);
0685
0686            return new LegendItem(label, description, toolTipText, urlText,
0687                    true, shape, true, paint,
0688                    isDrawBarOutline(), outlinePaint, outlineStroke,
0689                    false, new Line2D.Float(), new BasicStroke(1.0f),
0690                    Color.black);
0691        }
0692
0693        /**
0694         * Draws the bar for a single (series, category) data item.
0695         *
0696         * @param g2  the graphics device.
0697         * @param state  the renderer state.
0698         * @param dataArea  the data area.
0699         * @param plot  the plot.
0700         * @param domainAxis  the domain axis.
0701         * @param rangeAxis  the range axis.
0702         * @param dataset  the dataset.
0703         * @param row  the row index (zero-based).
0704         * @param column  the column index (zero-based).
0705         * @param pass  the pass index.
0706         */
0707        public void drawItem(Graphics2D g2,
0708                             CategoryItemRendererState state,
0709                             Rectangle2D dataArea,
0710                             CategoryPlot plot,
0711                             CategoryAxis domainAxis,
0712                             ValueAxis rangeAxis,
0713                             CategoryDataset dataset,
0714                             int row,
0715                             int column,
0716                             int pass) {
0717
0718            // nothing is drawn for null values...
0719            Number dataValue = dataset.getValue(row, column);
0720            if (dataValue == null) {
0721                return;
0722            }
0723
0724            double value = dataValue.doubleValue();
0725
0726            PlotOrientation orientation = plot.getOrientation();
0727            double barW0 = calculateBarW0(plot, orientation, dataArea, domainAxis,
0728                    state, row, column);
0729            double[] barL0L1 = calculateBarL0L1(value);
0730            if (barL0L1 == null) {
0731                return;  // the bar is not visible
0732            }
0733
0734            RectangleEdge edge = plot.getRangeAxisEdge();
0735            double transL0 = rangeAxis.valueToJava2D(barL0L1[0], dataArea, edge);
0736            double transL1 = rangeAxis.valueToJava2D(barL0L1[1], dataArea, edge);
0737            double barL0 = Math.min(transL0, transL1);
0738            double barLength = Math.max(Math.abs(transL1 - transL0),
0739                    getMinimumBarLength());
0740
0741            // draw the bar...
0742            Rectangle2D bar = null;
0743            if (orientation == PlotOrientation.HORIZONTAL) {
0744                bar = new Rectangle2D.Double(barL0, barW0, barLength,
0745                        state.getBarWidth());
0746            }
0747            else {
0748                bar = new Rectangle2D.Double(barW0, barL0, state.getBarWidth(),
```

C:/jfreechart-1.0.2/jfreechart-1.0.2/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 702

```
0702                        barLength);
0703            }
0704            Paint itemPaint = getItemPaint(row, column);
0705            GradientPaintTransformer t = getGradientPaintTransformer();
0706            if (t != null && itemPaint instanceof GradientPaint) {
0707                itemPaint = t.transform((GradientPaint) itemPaint, bar);
0708            }
0709            g2.setPaint(itemPaint);
0710            g2.fill(bar);
0711
0712            // draw the outline...
0713            if (isDrawBarOutline()
0714                    && state.getBarWidth() > BAR_OUTLINE_WIDTH_THRESHOLD) {
0715                Stroke stroke = getItemOutlineStroke(row, column);
0716                Paint paint = getItemOutlinePaint(row, column);
0717                if (stroke != null && paint != null) {
0718                    g2.setStroke(stroke);
0719                    g2.setPaint(paint);
0720                    g2.draw(bar);
0721                }
0722            }
0723
0724            CategoryItemLabelGenerator generator
0725                = getItemLabelGenerator(row, column);
0726            if (generator != null && isItemLabelVisible(row, column)) {
0727                drawItemLabel(g2, dataset, row, column, plot, generator, bar,
0728                        (value < 0.0));
0729            }
0730
0731            // add an item entity, if this information is being collected
0732            EntityCollection entities = state.getEntityCollection();
0733            if (entities != null) {
0734                addItemEntity(entities, dataset, row, column, bar);
0735            }
0736
0737        }
0738
0739        /**
0740         * Calculates the available space for each series.
0741         *
0742         * @param space  the space along the entire axis (in Java2D units).
0743         * @param axis  the category axis.
0744         * @param categories  the number of categories.
0745         * @param series  the number of series.
0746         *
0747         * @return The width of one series.
0748         */
0749        protected double calculateSeriesWidth(double space, CategoryAxis axis,
0750                                              int categories, int series) {
0751            double factor = 1.0 - getItemMargin() - axis.getLowerMargin()
0752                                - axis.getUpperMargin();
0753            if (categories > 1) {
0754                factor = factor - axis.getCategoryMargin();
0755            }
0756            return (space * factor) / (categories * series);
0757        }
0758
0759        /**
0760         * Draws an item label.  This method is overridden so that the bar can be
0761         * used to calculate the label anchor point.
0762         *
0763         * @param g2  the graphics device.
0764         * @param data  the dataset.
0765         * @param row  the row.
0766         * @param column  the column.
0767         * @param plot  the plot.
0768         * @param generator  the label generator.
0769         * @param bar  the bar.
```

C:/jfreechart-1.0.2/jfreechart-1.0.2-modified/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 749

```
0749                        barLength);
0750            }
0751            Paint itemPaint = getItemPaint(row, column);
0752            GradientPaintTransformer t = getGradientPaintTransformer();
0753            if (t != null && itemPaint instanceof GradientPaint) {
0754                itemPaint = t.transform((GradientPaint) itemPaint, bar);
0755            }
0756            g2.setPaint(itemPaint);
0757            g2.fill(bar);
0758
0759            // draw the outline...
0760            if (isDrawBarOutline()
0761                    && state.getBarWidth() > BAR_OUTLINE_WIDTH_THRESHOLD) {
0762                Stroke stroke = getItemOutlineStroke(row, column);
0763                Paint paint = getItemOutlinePaint(row, column);
0764                if (stroke != null && paint != null) {
0765                    g2.setStroke(stroke);
0766                    g2.setPaint(paint);
0767                    g2.draw(bar);
0768                }
0769            }
0770
0771            CategoryItemLabelGenerator generator
0772                = getItemLabelGenerator(row, column);
0773            if (generator != null && isItemLabelVisible(row, column)) {
0774                drawItemLabel(g2, dataset, row, column, plot, generator, bar,
0775                        (value < 0.0));
0776            }
0777
0778            // add an item entity, if this information is being collected
0779            EntityCollection entities = state.getEntityCollection();
0780            if (entities != null) {
0781                addItemEntity(entities, dataset, row, column, bar);
0782            }
0783
0784        }
0785
0786        /**
0787         * Calculates the available space for each series.
0788         *
0789         * @param space  the space along the entire axis (in Java2D units).
0790         * @param axis  the category axis.
0791         * @param categories  the number of categories.
0792         * @param series  the number of series.
0793         *
0794         * @return The width of one series.
0795         */
0796        protected double calculateSeriesWidth(double space, CategoryAxis axis,
0797                                              int categories, int series) {
0798            double factor = 1.0 - getItemMargin() - axis.getLowerMargin()
0799                                - axis.getUpperMargin();
0800            if (categories > 1) {
0801                factor = factor - axis.getCategoryMargin();
0802            }
0803            return (space * factor) / (categories * series);
0804        }
0805
0806        /**
0807         * Draws an item label.  This method is overridden so that the bar can be
0808         * used to calculate the label anchor point.
0809         *
0810         * @param g2  the graphics device.
0811         * @param data  the dataset.
0812         * @param row  the row.
0813         * @param column  the column.
0814         * @param plot  the plot.
0815         * @param generator  the label generator.
0816         * @param bar  the bar.
```

C:/jfreechart-1.0.2/jfreechart-1.0.2/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 770

```
0770        * @param negative  a flag indicating a negative value.
0771        */
0772       protected void drawItemLabel(Graphics2D g2,
0773                                    CategoryDataset data,
0774                                    int row,
0775                                    int column,
0776                                    CategoryPlot plot,
0777                                    CategoryItemLabelGenerator generator,
0778                                    Rectangle2D bar,
0779                                    boolean negative) {
0780
0781           String label = generator.generateLabel(data, row, column);
0782           if (label == null) {
0783               return;  // nothing to do
0784           }
0785
0786           Font labelFont = getItemLabelFont(row, column);
0787           g2.setFont(labelFont);
0788           Paint paint = getItemLabelPaint(row, column);
0789           g2.setPaint(paint);
0790
0791           // find out where to place the label...
0792           ItemLabelPosition position = null;
0793           if (!negative) {
0794               position = getPositiveItemLabelPosition(row, column);
0795           }
0796           else {
0797               position = getNegativeItemLabelPosition(row, column);
0798           }
0799
0800           // work out the label anchor point...
0801           Point2D anchorPoint = calculateLabelAnchorPoint(
0802                   position.getItemLabelAnchor(), bar, plot.getOrientation());
0803
0804           if (isInternalAnchor(position.getItemLabelAnchor())) {
0805               Shape bounds = TextUtilities.calculateRotatedStringBounds(label,
0806                       g2, (float) anchorPoint.getX(), (float) anchorPoint.getY(),
0807                       position.getTextAnchor(), position.getAngle(),
0808                       position.getRotationAnchor());
0809
0810               if (bounds != null) {
0811                   if (!bar.contains(bounds.getBounds2D())) {
0812                       if (!negative) {
0813                           position = getPositiveItemLabelPositionFallback();
0814                       }
0815                       else {
0816                           position = getNegativeItemLabelPositionFallback();
0817                       }
0818                       if (position != null) {
0819                           anchorPoint = calculateLabelAnchorPoint(
0820                                   position.getItemLabelAnchor(), bar,
0821                                   plot.getOrientation());
0822                       }
0823                   }
0824               }
0825
0826           }
0827
0828           if (position != null) {
0829               TextUtilities.drawRotatedString(label, g2,
0830                       (float) anchorPoint.getX(), (float) anchorPoint.getY(),
0831                       position.getTextAnchor(), position.getAngle(),
0832                       position.getRotationAnchor());
0833           }
0834       }
0835
0836       /**
0837        * Calculates the item label anchor point.
```

C:/jfreechart-1.0.2/jfreechart-1.0.2-modified/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 817

```
0817        * @param negative  a flag indicating a negative value.
0818        */
0819       protected void drawItemLabel(Graphics2D g2,
0820                                    CategoryDataset data,
0821                                    int row,
0822                                    int column,
0823                                    CategoryPlot plot,
0824                                    CategoryItemLabelGenerator generator,
0825                                    Rectangle2D bar,
0826                                    boolean negative) {
0827
0828           String label = generator.generateLabel(data, row, column);
0829           if (label == null) {
0830               return;  // nothing to do
0831           }
0832
0833           Font labelFont = getItemLabelFont(row, column);
0834           g2.setFont(labelFont);
0835           Paint paint = getItemLabelPaint(row, column);
0836           g2.setPaint(paint);
0837
0838           // find out where to place the label...
0839           ItemLabelPosition position = null;
0840           if (!negative) {
0841               position = getPositiveItemLabelPosition(row, column);
0842           }
0843           else {
0844               position = getNegativeItemLabelPosition(row, column);
0845           }
0846
0847           // work out the label anchor point...
0848           Point2D anchorPoint = calculateLabelAnchorPoint(
0849                   position.getItemLabelAnchor(), bar, plot.getOrientation());
0850
0851           if (isInternalAnchor(position.getItemLabelAnchor())) {
0852               Shape bounds = TextUtilities.calculateRotatedStringBounds(label,
0853                       g2, (float) anchorPoint.getX(), (float) anchorPoint.getY(),
0854                       position.getTextAnchor(), position.getAngle(),
0855                       position.getRotationAnchor());
0856
0857               if (bounds != null) {
0858                   if (!bar.contains(bounds.getBounds2D())) {
0859                       if (!negative) {
0860                           position = getPositiveItemLabelPositionFallback();
0861                       }
0862                       else {
0863                           position = getNegativeItemLabelPositionFallback();
0864                       }
0865                       if (position != null) {
0866                           anchorPoint = calculateLabelAnchorPoint(
0867                                   position.getItemLabelAnchor(), bar,
0868                                   plot.getOrientation());
0869                       }
0870                   }
0871               }
0872
0873           }
0874
0875           if (position != null) {
0876               TextUtilities.drawRotatedString(label, g2,
0877                       (float) anchorPoint.getX(), (float) anchorPoint.getY(),
0878                       position.getTextAnchor(), position.getAngle(),
0879                       position.getRotationAnchor());
0880           }
0881       }
0882
0883       /**
0884        * Calculates the item label anchor point.
```

C:/jfreechart-1.0.2/jfreechart-1.0.2/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 838

```
0838        *
0839        * @param anchor  the anchor.
0840        * @param bar  the bar.
0841        * @param orientation  the plot orientation.
0842        *
0843        * @return The anchor point.
0844        */
0845       private Point2D calculateLabelAnchorPoint(ItemLabelAnchor anchor,
0846                                                 Rectangle2D bar,
0847                                                 PlotOrientation orientation) {
0848
0849           Point2D result = null;
0850           double offset = getItemLabelAnchorOffset();
0851           double x0 = bar.getX() - offset;
0852           double x1 = bar.getX();
0853           double x2 = bar.getX() + offset;
0854           double x3 = bar.getCenterX();
0855           double x4 = bar.getMaxX() - offset;
0856           double x5 = bar.getMaxX();
0857           double x6 = bar.getMaxX() + offset;
0858
0859           double y0 = bar.getMaxY() + offset;
0860           double y1 = bar.getMaxY();
0861           double y2 = bar.getMaxY() - offset;
0862           double y3 = bar.getCenterY();
0863           double y4 = bar.getMinY() + offset;
0864           double y5 = bar.getMinY();
0865           double y6 = bar.getMinY() - offset;
0866
0867           if (anchor == ItemLabelAnchor.CENTER) {
0868               result = new Point2D.Double(x3, y3);
0869           }
0870           else if (anchor == ItemLabelAnchor.INSIDE1) {
0871               result = new Point2D.Double(x4, y4);
0872           }
0873           else if (anchor == ItemLabelAnchor.INSIDE2) {
0874               result = new Point2D.Double(x4, y4);
0875           }
0876           else if (anchor == ItemLabelAnchor.INSIDE3) {
0877               result = new Point2D.Double(x4, y3);
0878           }
0879           else if (anchor == ItemLabelAnchor.INSIDE4) {
0880               result = new Point2D.Double(x4, y2);
0881           }
0882           else if (anchor == ItemLabelAnchor.INSIDE5) {
0883               result = new Point2D.Double(x4, y2);
0884           }
0885           else if (anchor == ItemLabelAnchor.INSIDE6) {
0886               result = new Point2D.Double(x3, y2);
0887           }
0888           else if (anchor == ItemLabelAnchor.INSIDE7) {
0889               result = new Point2D.Double(x2, y2);
0890           }
0891           else if (anchor == ItemLabelAnchor.INSIDE8) {
0892               result = new Point2D.Double(x2, y2);
0893           }
0894           else if (anchor == ItemLabelAnchor.INSIDE9) {
0895               result = new Point2D.Double(x2, y3);
0896           }
0897           else if (anchor == ItemLabelAnchor.INSIDE10) {
0898               result = new Point2D.Double(x2, y4);
0899           }
0900           else if (anchor == ItemLabelAnchor.INSIDE11) {
0901               result = new Point2D.Double(x2, y4);
0902           }
0903           else if (anchor == ItemLabelAnchor.INSIDE12) {
0904               result = new Point2D.Double(x3, y4);
0905           }
```

C:/jfreechart-1.0.2/jfreechart-1.0.2-modified/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 885

```
0885        *
0886        * @param anchor  the anchor.
0887        * @param bar  the bar.
0888        * @param orientation  the plot orientation.
0889        *
0890        * @return The anchor point.
0891        */
0892       private Point2D calculateLabelAnchorPoint(ItemLabelAnchor anchor,
0893                                                 Rectangle2D bar,
0894                                                 PlotOrientation orientation) {
0895
0896           Point2D result = null;
0897           double offset = getItemLabelAnchorOffset();
0898           double x0 = bar.getX() - offset;
0899           double x1 = bar.getX();
0900           double x2 = bar.getX() + offset;
0901           double x3 = bar.getCenterX();
0902           double x4 = bar.getMaxX() - offset;
0903           double x5 = bar.getMaxX();
0904           double x6 = bar.getMaxX() + offset;
0905
0906           double y0 = bar.getMaxY() + offset;
0907           double y1 = bar.getMaxY();
0908           double y2 = bar.getMaxY() - offset;
0909           double y3 = bar.getCenterY();
0910           double y4 = bar.getMinY() + offset;
0911           double y5 = bar.getMinY();
0912           double y6 = bar.getMinY() - offset;
0913
0914           if (anchor == ItemLabelAnchor.CENTER) {
0915               result = new Point2D.Double(x3, y3);
0916           }
0917           else if (anchor == ItemLabelAnchor.INSIDE1) {
0918               result = new Point2D.Double(x4, y4);
0919           }
0920           else if (anchor == ItemLabelAnchor.INSIDE2) {
0921               result = new Point2D.Double(x4, y4);
0922           }
0923           else if (anchor == ItemLabelAnchor.INSIDE3) {
0924               result = new Point2D.Double(x4, y3);
0925           }
0926           else if (anchor == ItemLabelAnchor.INSIDE4) {
0927               result = new Point2D.Double(x4, y2);
0928           }
0929           else if (anchor == ItemLabelAnchor.INSIDE5) {
0930               result = new Point2D.Double(x4, y2);
0931           }
0932           else if (anchor == ItemLabelAnchor.INSIDE6) {
0933               result = new Point2D.Double(x3, y2);
0934           }
0935           else if (anchor == ItemLabelAnchor.INSIDE7) {
0936               result = new Point2D.Double(x2, y2);
0937           }
0938           else if (anchor == ItemLabelAnchor.INSIDE8) {
0939               result = new Point2D.Double(x2, y2);
0940           }
0941           else if (anchor == ItemLabelAnchor.INSIDE9) {
0942               result = new Point2D.Double(x2, y3);
0943           }
0944           else if (anchor == ItemLabelAnchor.INSIDE10) {
0945               result = new Point2D.Double(x2, y4);
0946           }
0947           else if (anchor == ItemLabelAnchor.INSIDE11) {
0948               result = new Point2D.Double(x2, y4);
0949           }
0950           else if (anchor == ItemLabelAnchor.INSIDE12) {
0951               result = new Point2D.Double(x3, y4);
0952           }
```

C:/jfreechart-1.0.2/jfreechart-1.0.2/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 906

```
0906            else if (anchor == ItemLabelAnchor.OUTSIDE1) {
0907                result = new Point2D.Double(x5, y6);
0908            }
0909            else if (anchor == ItemLabelAnchor.OUTSIDE2) {
0910                result = new Point2D.Double(x6, y5);
0911            }
0912            else if (anchor == ItemLabelAnchor.OUTSIDE3) {
0913                result = new Point2D.Double(x6, y3);
0914            }
0915            else if (anchor == ItemLabelAnchor.OUTSIDE4) {
0916                result = new Point2D.Double(x6, y1);
0917            }
0918            else if (anchor == ItemLabelAnchor.OUTSIDE5) {
0919                result = new Point2D.Double(x5, y0);
0920            }
0921            else if (anchor == ItemLabelAnchor.OUTSIDE6) {
0922                result = new Point2D.Double(x3, y0);
0923            }
0924            else if (anchor == ItemLabelAnchor.OUTSIDE7) {
0925                result = new Point2D.Double(x1, y0);
0926            }
0927            else if (anchor == ItemLabelAnchor.OUTSIDE8) {
0928                result = new Point2D.Double(x0, y1);
0929            }
0930            else if (anchor == ItemLabelAnchor.OUTSIDE9) {
0931                result = new Point2D.Double(x0, y3);
0932            }
0933            else if (anchor == ItemLabelAnchor.OUTSIDE10) {
0934                result = new Point2D.Double(x0, y5);
0935            }
0936            else if (anchor == ItemLabelAnchor.OUTSIDE11) {
0937                result = new Point2D.Double(x1, y6);
0938            }
0939            else if (anchor == ItemLabelAnchor.OUTSIDE12) {
0940                result = new Point2D.Double(x3, y6);
0941            }
0942
0943            return result;
0944
0945        }
0946
0947        /**
0948         * Returns <code>true</code> if the specified anchor point is inside a bar.
0949         *
0950         * @param anchor  the anchor point.
0951         *
0952         * @return A boolean.
0953         */
0954        private boolean isInternalAnchor(ItemLabelAnchor anchor) {
0955            return anchor == ItemLabelAnchor.CENTER
0956                    || anchor == ItemLabelAnchor.INSIDE1
0957                    || anchor == ItemLabelAnchor.INSIDE2
0958                    || anchor == ItemLabelAnchor.INSIDE3
0959                    || anchor == ItemLabelAnchor.INSIDE4
0960                    || anchor == ItemLabelAnchor.INSIDE5
0961                    || anchor == ItemLabelAnchor.INSIDE6
0962                    || anchor == ItemLabelAnchor.INSIDE7
0963                    || anchor == ItemLabelAnchor.INSIDE8
0964                    || anchor == ItemLabelAnchor.INSIDE9
0965                    || anchor == ItemLabelAnchor.INSIDE10
0966                    || anchor == ItemLabelAnchor.INSIDE11
0967                    || anchor == ItemLabelAnchor.INSIDE12;
0968        }
0969
0970        /**
0971         * Tests this instance for equality with an arbitrary object.
0972         *
0973         * @param obj  the object (<code>null</code> permitted).
```

C:/jfreechart-1.0.2/jfreechart-1.0.2-modified/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 953

```
0953            else if (anchor == ItemLabelAnchor.OUTSIDE1) {
0954                result = new Point2D.Double(x5, y6);
0955            }
0956            else if (anchor == ItemLabelAnchor.OUTSIDE2) {
0957                result = new Point2D.Double(x6, y5);
0958            }
0959            else if (anchor == ItemLabelAnchor.OUTSIDE3) {
0960                result = new Point2D.Double(x6, y3);
0961            }
0962            else if (anchor == ItemLabelAnchor.OUTSIDE4) {
0963                result = new Point2D.Double(x6, y1);
0964            }
0965            else if (anchor == ItemLabelAnchor.OUTSIDE5) {
0966                result = new Point2D.Double(x5, y0);
0967            }
0968            else if (anchor == ItemLabelAnchor.OUTSIDE6) {
0969                result = new Point2D.Double(x3, y0);
0970            }
0971            else if (anchor == ItemLabelAnchor.OUTSIDE7) {
0972                result = new Point2D.Double(x1, y0);
0973            }
0974            else if (anchor == ItemLabelAnchor.OUTSIDE8) {
0975                result = new Point2D.Double(x0, y1);
0976            }
0977            else if (anchor == ItemLabelAnchor.OUTSIDE9) {
0978                result = new Point2D.Double(x0, y3);
0979            }
0980            else if (anchor == ItemLabelAnchor.OUTSIDE10) {
0981                result = new Point2D.Double(x0, y5);
0982            }
0983            else if (anchor == ItemLabelAnchor.OUTSIDE11) {
0984                result = new Point2D.Double(x1, y6);
0985            }
0986            else if (anchor == ItemLabelAnchor.OUTSIDE12) {
0987                result = new Point2D.Double(x3, y6);
0988            }
0989
0990            return result;
0991
0992        }
0993
0994        /**
0995         * Returns <code>true</code> if the specified anchor point is inside a bar.
0996         *
0997         * @param anchor  the anchor point.
0998         *
0999         * @return A boolean.
1000         */
1001        private boolean isInternalAnchor(ItemLabelAnchor anchor) {
1002            return anchor == ItemLabelAnchor.CENTER
1003                    || anchor == ItemLabelAnchor.INSIDE1
1004                    || anchor == ItemLabelAnchor.INSIDE2
1005                    || anchor == ItemLabelAnchor.INSIDE3
1006                    || anchor == ItemLabelAnchor.INSIDE4
1007                    || anchor == ItemLabelAnchor.INSIDE5
1008                    || anchor == ItemLabelAnchor.INSIDE6
1009                    || anchor == ItemLabelAnchor.INSIDE7
1010                    || anchor == ItemLabelAnchor.INSIDE8
1011                    || anchor == ItemLabelAnchor.INSIDE9
1012                    || anchor == ItemLabelAnchor.INSIDE10
1013                    || anchor == ItemLabelAnchor.INSIDE11
1014                    || anchor == ItemLabelAnchor.INSIDE12;
1015        }
1016
1017        /**
1018         * Tests this instance for equality with an arbitrary object.
1019         *
1020         * @param obj  the object (<code>null</code> permitted).
```

C:/jfreechart-1.0.2/jfreechart-1.0.2/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 974

```
0974        *
0975        * @return A boolean.
0976        */
0977       public boolean equals(Object obj) {
0978
0979           if (obj == this) {
0980               return true;
0981           }
0982           if (!(obj instanceof BarRenderer)) {
0983               return false;
0984           }
0985           if (!super.equals(obj)) {
0986               return false;
0987           }
0988           BarRenderer that = (BarRenderer) obj;
0989           if (this.base != that.base) {
0990               return false;
0991           }
0992           if (this.itemMargin != that.itemMargin) {
0993               return false;
0994           }
0995           if (this.drawBarOutline != that.drawBarOutline) {
0996               return false;
0997           }
0998           if (this.maximumBarWidth != that.maximumBarWidth) {
0999               return false;
1000           }
1001           if (this.minimumBarLength != that.minimumBarLength) {
1002               return false;
1003           }
1004           if (!ObjectUtilities.equal(this.gradientPaintTransformer,
1005                   that.gradientPaintTransformer)) {
1006               return false;
1007           }
1008           if (!ObjectUtilities.equal(this.positiveItemLabelPositionFallback,
1009               that.positiveItemLabelPositionFallback)) {
1010               return false;
1011           }
1012           if (!ObjectUtilities.equal(this.negativeItemLabelPositionFallback,
1013               that.negativeItemLabelPositionFallback)) {
1014               return false;
1015           }
1016           return true;
1017
1018       }
1019
1020   }
1021
```

C:/jfreechart-1.0.2/jfreechart-1.0.2-modified/source/org/jfree/chart/renderer/category/BarRenderer.java, Top line: 1021

```
1021        *
1022        * @return A boolean.
1023        */
1024       public boolean equals(Object obj) {
1025
1026           if (obj == this) {
1027               return true;
1028           }
1029           if (!(obj instanceof BarRenderer)) {
1030               return false;
1031           }
1032           if (!super.equals(obj)) {
1033               return false;
1034           }
1035           BarRenderer that = (BarRenderer) obj;
1036           if (this.base != that.base) {
1037               return false;
1038           }
1039           if (this.itemMargin != that.itemMargin) {
1040               return false;
1041           }
1042           if (this.drawBarOutline != that.drawBarOutline) {
1043               return false;
1044           }
1045           if (this.maximumBarWidth != that.maximumBarWidth) {
1046               return false;
1047           }
1048           if (this.minimumBarLength != that.minimumBarLength) {
1049               return false;
1050           }
1051           if (!ObjectUtilities.equal(this.gradientPaintTransformer,
1052                   that.gradientPaintTransformer)) {
1053               return false;
1054           }
1055           if (!ObjectUtilities.equal(this.positiveItemLabelPositionFallback,
1056               that.positiveItemLabelPositionFallback)) {
1057               return false;
1058           }
1059           if (!ObjectUtilities.equal(this.negativeItemLabelPositionFallback,
1060               that.negativeItemLabelPositionFallback)) {
1061               return false;
1062           }
1063           return true;
1064
1065       }
1066
1067   }
1068
```