





Gauge Programmer’s Guide

© 2009 Quest Software, Inc. ALL RIGHTS RESERVED.

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

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

Quest Software World Headquarters LEGAL Dept

5 Polaris Way

Aliso Viejo, CA 92656 Email: [legal@quest.com](mailto:legal@quest.com)

Refer to our Web site (www.quest.com) for regional and international office information.

Trademarks

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

Disclaimer

The information in this document is provided in connection with Quest products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest products. EXCEPT AS SET FORTH IN QUEST'S TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST HAS

BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest does not make any commitment to update the information contained in this document.

Third Party Contributions

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

Gauge Programmer’s Guide January 2009

**Version 6.4.2**

**Table of Contents**

[Preface 11](#_bookmark0)

[Introducing JClass Gauge 11](#_bookmark1)

[Assumptions 12](#_bookmark3)

[Typographical Conventions 12](#_bookmark4)

[How to Use This Guide 13](#_bookmark5)

[Example & Demo Gallery 14](#_bookmark7)

[API Documentation (Javadoc) 15](#_bookmark9)

[Licensing 15](#_bookmark11)

[Related Documents 15](#_bookmark13)

[About Quest Software, Inc. 16](#_bookmark15)

[JClass Community 17](#_bookmark17)

1. [Learning JClass Gauge Basics 19](#_bookmark19)
   1. [Types of Gauges 20](#_bookmark23)

[Indicator Gauge 20](#_bookmark24)

[Circular Gauge 21](#_bookmark26)

[Linear Gauge 21](#_bookmark28)

* 1. [Graph Component for Gauges 22](#_bookmark30)
  2. [Containment Hierarchies 22](#_bookmark31)

[Indicator Gauge Containment Hierarchy 23](#_bookmark32)

[Circular and Linear Gauges Containment Hierarchy 24](#_bookmark33)

* 1. [Key Concepts 25](#_bookmark34)

[Gauge Size 25](#_bookmark35)

[Images and the PortableImage Object 25](#_bookmark36)

[Layering Gauges 26](#_bookmark38)

[Layering a Graph on Top of a Gauge 27](#_bookmark40)

* 1. [Key Concepts for Circular and Linear Gauges 27](#_bookmark42)

[How Components are Rendered on a Gauge 28](#_bookmark43)

[Sizing Components Using Extent Parameters 29](#_bookmark45)

* 1. [Internationalization 31](#_bookmark47)

[Part](#_bookmark51) [I: Indicator Gauges and Graphs](#_bookmark49)

1. [Creating Indicator Gauges and Panels 35](#_bookmark52)
   1. [Overview of the Indicator Gauge Objects 35](#_bookmark55)
   2. [Summary of Properties Used 37](#_bookmark57)
   3. [A Sample Standalone Indicator Gauge 38](#_bookmark58)

[Sample Gauge 38](#_bookmark59)

[Sample Code 39](#_bookmark61)

* 1. [Creating a Standalone Indicator Gauge 41](#_bookmark62)

[Constructing a JCIndicatorGauge Object 41](#_bookmark64)

[Assigning a Value to Your Indicator Gauge 42](#_bookmark65)

[Next Steps 42](#_bookmark67)

* 1. [Customizing the Behavior of Your Indicator Gauge 42](#_bookmark68)

[Defining Ranges for a Standalone Gauge 42](#_bookmark70)

[Setting the Base Value for a Standalone Gauge 45](#_bookmark74)

* 1. [A Sample Indicator Gauge Panel 46](#_bookmark76)

[Sample Panel 46](#_bookmark77)

[Sample Code 47](#_bookmark79)

* 1. [Creating an Indicator Gauge Panel 48](#_bookmark80)

[Constructing a JCBaseIndicatorGauge Object 48](#_bookmark82)

[Constructing a JCIndicatorGaugePanel Object 49](#_bookmark84)

[Assigning a Value to Your Indicator Gauge Panel 49](#_bookmark86)

[Next Steps 49](#_bookmark88)

* 1. [Customizing the Behavior of Your Panel 50](#_bookmark89)

[Setting the Value Type and Defining Ranges for a Panel 50](#_bookmark91)

[Setting the Panel Direction 55](#_bookmark96)

[Setting the Panel Orientation 56](#_bookmark99)

[Setting the Base Value for a Panel 56](#_bookmark100)

[Changing the Gauge on the Panel 57](#_bookmark104)

[Changing the Number of Gauges Displayed 57](#_bookmark106)

* 1. [Customizing the Appearance of Gauges and Panels 58](#_bookmark108)

[Setting the Icon Shape 58](#_bookmark110)

[Setting the Gauge Size 59](#_bookmark112)

[Setting the Padding 59](#_bookmark114)

[Setting a Background Color 60](#_bookmark116)

[Specifying and Scaling an Image 62](#_bookmark118)

[Setting the Font and Color for Text 63](#_bookmark120)

[Making Gauge Icons Blink 63](#_bookmark122)

1. [Creating Graphs 65](#_bookmark124)
   1. [Overview of the Graph Object 65](#_bookmark127)
   2. [Summary of Properties Used 66](#_bookmark129)
   3. [A Sample Graph 67](#_bookmark130)

[Sample Graph 67](#_bookmark131)

[Sample Code 67](#_bookmark132)

* 1. [Creating a Graph 69](#_bookmark133)

[Constructing a JCGraph Object 69](#_bookmark134)

[Supplying Values to Your Graph 70](#_bookmark136)

[Next Steps 71](#_bookmark139)

* 1. [Customizing the Graph 71](#_bookmark140)

[Setting the Direction 71](#_bookmark142)

[Setting the Minimum and Maximum Values 72](#_bookmark144)

[Setting the Distance Between Value Points 72](#_bookmark146)

[Setting Line Attributes 73](#_bookmark148)

[Setting a Background Color 73](#_bookmark150)

[Specifying and Scaling an Image 74](#_bookmark152)

[Overlaying a Graph 74](#_bookmark154)

[Part](#_bookmark157) [II: Circular and Linear Gauges](#_bookmark155)

1. [Creating Circular and Linear Gauges 77](#_bookmark158)
   1. [Overview of the Gauge Objects 77](#_bookmark161)
   2. [Creating a Circular Gauge 78](#_bookmark163)

[Constructing a JCCircularGauge Object 78](#_bookmark164)

[Next Steps 81](#_bookmark167)

* 1. [Creating a Linear Gauge 81](#_bookmark168)

[Constructing a JCLinearGauge Object 82](#_bookmark169)

[Next Steps 83](#_bookmark172)

* 1. [Assigning Values to a Gauge 83](#_bookmark173)
  2. [Customizing the Gauge Container 84](#_bookmark175)

[Setting a Background Color 84](#_bookmark176)

[Specifying and Scaling an Image 84](#_bookmark178)

1. [Defining the Scale 87](#_bookmark180)
   1. [Overview of the Scale Objects 87](#_bookmark183)
   2. [Summary of Properties Used 88](#_bookmark185)
   3. [Defining a Circular Scale 89](#_bookmark186)

[Accessing the Preconstructed Circular Scale Object 89](#_bookmark189)

[Constructing a JCCircularScale Object 90](#_bookmark191)

* 1. [Customizing a Circular Scale 90](#_bookmark193)

[Setting Min and Max Values on a Circular Scale 90](#_bookmark195)

[Setting Start and Stop Angles 91](#_bookmark197)

[Setting the Direction on a Circular Scale 92](#_bookmark201)

[Setting the Zoom Factor on a Circular Scale 93](#_bookmark203)

[Setting the Initial Size of a Circular Scale 93](#_bookmark206)

[Setting the Background Color 94](#_bookmark208)

[Specifying and Scaling an Image 94](#_bookmark210)

* 1. [Defining a Linear Scale 95](#_bookmark212)

[Accessing the Preconstructed Linear Scale Object 96](#_bookmark215)

[Constructing a JCLinearScale Object 96](#_bookmark217)

* 1. [Customizing a Linear Scale 97](#_bookmark219)

[Setting Min and Max Values on a Linear Scale 97](#_bookmark221)

[Setting the Orientation 97](#_bookmark223)

[Setting the Direction on a Linear Scale 98](#_bookmark225)

[Setting a Border on a Linear Scale 98](#_bookmark227)

[Setting the Zoom Factor on a Linear Scale 98](#_bookmark229)

[Setting the Initial Size of a Linear Scale 99](#_bookmark232)

[Setting Background Properties 100](#_bookmark234)

* 1. [Enabling User Interaction for Scale Values 100](#_bookmark236)

1. [Defining Ticks and Tick Labels 101](#_bookmark238)
   1. [Overview of Tick Objects 101](#_bookmark241)
   2. [Summary of Properties Used 102](#_bookmark243)
   3. [Defining Ticks 102](#_bookmark244)

[Accessing the Preconstructed Tick Object 103](#_bookmark246)

[Constructing JCCircularTick and JCLinearTick Objects 103](#_bookmark248)

* 1. [Customizing Tick Marks 104](#_bookmark250)

[Setting the Number of Ticks and the Bounds 105](#_bookmark254)

[Setting the Placement and Length of Tick Marks 105](#_bookmark256)

[Setting the Tick Style and Width 107](#_bookmark260)

[Setting the Color 108](#_bookmark262)

[Defining a Custom Tick Style 108](#_bookmark264)

[Connecting Tick Marks 109](#_bookmark266)

[Displaying Tick Marks 110](#_bookmark267)

* 1. [Customizing Tick Labels 110](#_bookmark268)

[Displaying Tick Labels 110](#_bookmark270)

[Setting the Tick Label Extent 110](#_bookmark271)

[Setting the Precision Used for Values Displayed in Tick Labels 111](#_bookmark272) [Adding Text 112](#_bookmark274)

[Placing Labels Outside of the Scale Bounds 112](#_bookmark276)

1. [Defining Indicators and Needles 113](#_bookmark278)
   1. [Overview of the Indicator and Needle Objects 113](#_bookmark281)
   2. [Summary of Properties Used 114](#_bookmark283)
   3. [Defining Indicators 115](#_bookmark284)

[Constructing JCCircularIndicator or JCLinearIndicator Objects 115](#_bookmark286) [7.4 Defining Needles 116](#_bookmark288)

[Accessing the Preconstructed Needle Object 116](#_bookmark290)

[Constructing JCCircularNeedle or JCLinearNeedle Objects . 116](#_bookmark292)

* 1. [Customizing Indicators and Needles 117](#_bookmark294)

[Updating the Value 117](#_bookmark296)

[Setting the Style 117](#_bookmark298)

[Setting the Width 118](#_bookmark300)

[Setting the Size of an Indicator 118](#_bookmark301)

[Setting the Size of a Needle 118](#_bookmark302)

[Reversing an Indicator or Needle 118](#_bookmark304)

[Setting the Color 118](#_bookmark305)

[Hiding an Indicator or Needle 119](#_bookmark307)

[Defining a Custom Indicator Style 119](#_bookmark308)

* 1. [Adding User Interaction to Needles 120](#_bookmark310)

1. [Defining Text Components 121](#_bookmark311)
   1. [Overview of Text-based Objects 121](#_bookmark314)
   2. [Summary of Properties Used 123](#_bookmark315)
   3. [Defining Headers and Footers 124](#_bookmark316)
   4. [Defining Legends 124](#_bookmark318)

[Types of Legends 124](#_bookmark320)

[Customizing Legends 125](#_bookmark326)

[Creating Custom Legends 128](#_bookmark335)

[JCLegend Interfaces 128](#_bookmark337)

[Enabling User Interaction in Legends 128](#_bookmark339)

* 1. [Defining Labels 129](#_bookmark340)

[Aligning Text 129](#_bookmark342)

[Sample Code 130](#_bookmark345)

* 1. [Understanding the Constraint Mechanism 131](#_bookmark346)

[RadialConstraint and RadialLayout 131](#_bookmark348)

[LinearConstraint and LinearLayout 131](#_bookmark350)

* 1. [Positioning Elements on the Gauge Object 132](#_bookmark352)

1. [Defining Ranges 133](#_bookmark354)
   1. [Overview of the Range Objects 133](#_bookmark357)
   2. [Summary of Properties Used 134](#_bookmark358)
   3. [Defining a Range 134](#_bookmark359)

[Constructing a JCCircularRange or JCLinearRange Object . 135](#_bookmark362) [Setting Range Values 135](#_bookmark364)

* 1. [Customizing a Range 136](#_bookmark366)

[Setting the Breadth and Location of a Range 136](#_bookmark368)

[Setting the Range Color 137](#_bookmark371)

[Specifying and Scaling an Image 137](#_bookmark373)

* 1. [Specifying a Foreground Image in a Linear Range 138](#_bookmark375)
  2. [Creating an Offset Range in a Circular Gauge 139](#_bookmark377)

1. [Defining the Center Object in a Circular Gauge 141](#_bookmark380)
   1. [Overview of the Center Object 141](#_bookmark383)
   2. [Summary of Properties Used 142](#_bookmark385)
   3. [Defining the Center 142](#_bookmark386)

[Accessing the Preconstructed Center Object 142](#_bookmark387)

[Constructing a JCCenter Object 142](#_bookmark389)

* 1. [Customizing the Center 143](#_bookmark391)

[Setting the Center Radius 143](#_bookmark393)

[Setting the Background Color 144](#_bookmark395)

[Specifying and Scaling an Image 144](#_bookmark397)

[Part](#_bookmark401) [III: Using XML](#_bookmark399)

1. [Creating Gauges and Graphs with XML 147](#_bookmark402)
   1. [Overview of XML for JClass Gauge 147](#_bookmark405)

[DTDs 147](#_bookmark407)

[Factory Classes 148](#_bookmark409)

[LoadProperties Class 148](#_bookmark411)

[Images and the OutputProperties Class 148](#_bookmark413)

[XML Examples 149](#_bookmark415)

* 1. [Creating a Gauge or Graph From XML 149](#_bookmark416)

[Using the Factories to Make a Gauge or Graph 149](#_bookmark418)

[Setting JClass Gauge Properties Using XML 151](#_bookmark421)

[Constructing a LoadProperties Object 153](#_bookmark425)

* 1. [Updating a Gauge or Graph From XML 154](#_bookmark427)

[Updating a Circular or Linear Gauge from XML 154](#_bookmark429)

[Updating an Indicator Gauge or Panel from XML 155](#_bookmark431)

[Updating Graphs from XML 155](#_bookmark432)

* 1. [Saving a Gauge or Graph to XML 156](#_bookmark433)

[Saving a Circular or Linear Gauge to XML 156](#_bookmark435)

[Saving an Indicator Gauge or Panel to XML 156](#_bookmark436)

[Saving Graphs to XML 156](#_bookmark437)

* 1. [Preparing to Save Information About Images to XML 157](#_bookmark438)

[Constructing an OutputProperties Object 157](#_bookmark440)

[Setting Output Properties on a Background Image 158](#_bookmark442)

[Setting Output Properties on a Foreground Image 158](#_bookmark444)

[How Image Information is Stored and Accessed 159](#_bookmark446)

* 1. [Internationalizing Your XML-based Gauge 160](#_bookmark448)

[Using Variables 160](#_bookmark450)

[Creating a Resource Bundle 161](#_bookmark451)

[Using Resource Bundles 162](#_bookmark452)

1. [XML DTDs 165](#_bookmark453)
   1. [IndicatorGauge DTD 165](#_bookmark456)

[indicator-gauge 166](#_bookmark458)

[indicator-gauge-panel 167](#_bookmark460)

[base-gauge 169](#_bookmark462)

[icon 169](#_bookmark464)

[indicator-range 170](#_bookmark466)

* 1. [JCGauge DTD 171](#_bookmark468)

[gauge 171](#_bookmark469)

[bevel-border 171](#_bookmark471)

[center 172](#_bookmark473)

[circular-gauge 173](#_bookmark475)

[circular-range 174](#_bookmark477)

[circular-scale 175](#_bookmark479)

[compound-border 176](#_bookmark480)

[dash-array 177](#_bookmark482)

[empty-border 177](#_bookmark484)

[etched-border 177](#_bookmark486)

[external-java-code 178](#_bookmark488)

[font 179](#_bookmark490)

[footer 179](#_bookmark492)

[gauge-area 180](#_bookmark494)

[graph 181](#_bookmark496)

[header 182](#_bookmark498)

[image-file 183](#_bookmark500)

[indicator 184](#_bookmark502)

[insets 185](#_bookmark504)

[label 186](#_bookmark506)

[layout-hints 187](#_bookmark508)

[legend 187](#_bookmark510)

[legend-column 189](#_bookmark512)

[line-border 190](#_bookmark514)

[line-style 190](#_bookmark516)

[linear-constraint 191](#_bookmark518)

[linear-gauge 192](#_bookmark520)

[linear-range 193](#_bookmark522)

[linear-scale 194](#_bookmark524)

[matte-border 196](#_bookmark526)

[multi-col 196](#_bookmark528)

[needle 197](#_bookmark530)

[radial-constraint 198](#_bookmark532)

[tick 199](#_bookmark534)

[titled-border 201](#_bookmark536)

[value 202](#_bookmark538)

[values 202](#_bookmark540)

[Index 203](#_bookmark542)

## Preface

[*Introducing JClass Gauge*](#_bookmark1)■ [*Assumptions*](#_bookmark3)■ [*Typographical Conventions*](#_bookmark4)[*How to Use This Guide*](#_bookmark5)■ [*API Documentation (Javadoc)*](#_bookmark9)■ [*Licensing*](#_bookmark11)[*Related Documents*](#_bookmark13)■ [*About Quest Software, Inc.*](#_bookmark15)

#### Introducing JClass Gauge

JClass Gauge is a set of GUI components that you can use to add gauges to your user interface. They are 100% Java, highly configurable, and scalable. The indicator gauges can be used to highlight a fluctuating value in your real-time application in a visually meaningful way. Linear and circular gauges give you the flexibility to present data in terms of a graduated scale. For example, you can create a linear gauge that looks like a thermometer or a circular gauge that simulates an airplane cockpit gauge.

Feature Overview

JClass Gauge offers the following scalable gauge components:

* indicator gauge that can be used standalone or replicated within a panel
* circular gauge that can be shown as a full circle, half circle, or as a quadrant
* linear gauge that can be displayed horizontally or vertically
* light-weight graph component

Circular and linear gauges offer the following highly-customizable features:

* scale
* tick marks and tick labels
* needles and indicators
* header and footer titles
* labels
* legend
* ranges
* center disk (for circular gauges)
* support for images

JClass Gauge can be used in conjunction with the other products in JClass DesktopViews, as well as with ordinary Swing components. You can freely distribute the images generated with JClass Gauge according to the terms of the *License Agreement* that appears during the installation.

11

**Who Uses JClass Gauge**

JClass Gauge is designed to offer ease-of-use for the following Java developers:

* *Java component users* can create gauges and set properties programmatically.
* *JavaBean developers* can add gauges and set properties using the JClass Gauge Beans within their Integrated Development Environment (IDE).
* *XML developers* can create gauges using the JClass Gauge factory and set properties using the XML elements defined in the JClass Gauge DTDs.
* *Object-oriented developers* can extend JClass Gauge objects using the interfaces provided.

#### Assumptions

This manual assumes that you have some experience with the Java programming language. You should have a basic understanding of object-oriented programming and Java programming concepts such as classes, methods, and packages before proceeding with this manual.

If you want to use the JClass Gauge Beans, you should already be familiar with the JavaBeans technology and how to use beans in your IDE before attempting to add a gauge. For a selection of reference documents, see [“Related Documents” on page 15](#_bookmark13).

#### Typographical Conventions

The following typographical conventions are used throughout this guide:

Typewriter Font

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

*Italic Text*

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

**Bold**

* Keyboard key names and menu references.

#### How to Use This Guide

Everyone should review the first section of [Chapter 1, “Learning JClass Gauge Basics,”](#_bookmark20) to understand the types of gauges available. After you select the kind of gauge that you want to add your interface, you can focus on the relevant chapters. If you are using XML, there is some supplementary material to review as well.

Indicator Gauges

If you want to create an indicator gauge, you should review [Chapter 2, “Creating](#_bookmark53) [Indicator Gauges and Panels.”](#_bookmark53) It describes how to create and customize a standalone indicator gauge and a panel of gauges.

Circular and Linear Gauges

If you want to create a circular or linear gauge, Part II of the guide describes how to create and customize a gauge. In particular, you should review the following chapters:

* [Chapter 4, “Creating Circular and Linear Gauges,”](#_bookmark159) describes how to create a gauge and assign a value to it.
* [Chapter 5, “Defining the Scale,”](#_bookmark181) describes how to set the scale properties.
* [Chapter 6, “Defining Ticks and Tick Labels,”](#_bookmark239) describes how to display tick marks and/or tick labels on your gauge and set their properties
* [Chapter 7, “Defining Indicators and Needles,”](#_bookmark279) describes how to display needles and/or indicators on your gauge and set their properties. Needles and indicators are used to point to values on your scale.
* [Chapter 8, “Defining Text Components,”](#_bookmark312) describes how to display a header, footer, legend, or other labels on the gauge.
* [Chapter 9, “Defining Ranges,”](#_bookmark355) describes how you can use ranges to highlight values.
* If you selected a circular gauge, [Chapter 10, “Defining the Center Object in a Circular Gauge,”](#_bookmark381) describes how to customize the center disk in a circular scale.

Graph Component

If you want to create a JCGraph component beside or on top of your gauge, you should review [Chapter 3, “Creating Graphs.”](#_bookmark125)

XML Developers

In addition to the chapters outlined above, review the following material.

* [Chapter 11, “Creating Gauges and Graphs with XML,”](#_bookmark403) describes how to create and update gauges and graphs from XML and how to save gauges and graphs to XML.
* [Chapter 12, “XML DTDs,”](#_bookmark454) is a reference chapter that describes the DTDs.

JavaBean Developers

Refer to your IDE documentation for information on how to add third-party beans to your environment.

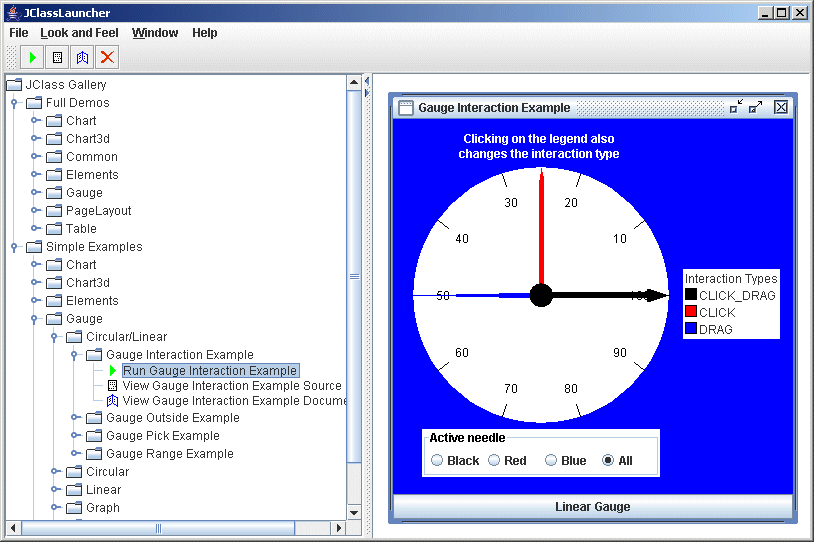
#### Example & Demo Gallery

The Example & Demo Gallery provides an interface for running examples and demos that implement JClass DesktopViews components. The examples show you how to create and customize a single JClass DesktopViews component. Demos demonstrate how to use one or more JClass DesktopViews components within the context of a larger application. Many of the JClass Gauge examples and demos are referenced in this guide.

You can start the Gallery by launching *jcexamples* from the *JCLASS\_HOME/bin/* directory, or in Microsoft Windows by double-clicking the Example & Demo Gallery item in the JClass DesktopViews program group off the Start menu. The Gallery runs within a custom JComponent.

In the Example & Demo Gallery, you can:

* locate examples for JClass Gauge under the Simple Examples > Gauge branch
* locate demos for JClass Gauge under Full Demos > Gauge branch
* run examples and demos without compiling
* view a description of an example or demo
* open the source code for an example or demo



*Figure 1 JClass DesktopViews Example & Demo Gallery*

Alternatively, you can find JClass Gauge examples in the *JCLASS\_HOME/examples/gauge*

directory and demos in the *JCLASS\_HOME/demos/gauge* directory.

#### API Documentation (Javadoc)

The Javadocs for the JClass Gauge API are part of the *JClass DesktopViews API Documentation*. The *API documentation* is installed automatically when you install JClass DesktopViews. It is located in the *JCLASS\_HOME/docs/api/* directory.

The following packages are particularly relevant for JClass Gauge:

* + com.klg.jclass.gauge
  + com.klg.jclass.gauge.beans
  + com.klg.jclass.gauge.graph
  + com.klg.jclass.gauge.indicator
  + com.klg.jclass.util.legend

On Microsoft Windows installations, you can find a link to the *API documentation* from the Start menu under the JClass DesktopViews program group.

#### Licensing

In order to use JClass Gauge, you need a valid license. Complete details about licensing are outlined in the *JClass DesktopViews Installation Guide*. The *Installation Guide* is installed automatically when you install JClass DesktopViews. You can find a PDF version and an HTML version in the *JCLASS\_HOME/docs/getstarted/* directory.

#### Related Documents

The following resources may be useful:

* + “*Java Platform Documentation*” at [*http://java.sun.com/docs/index.html*](http://java.sun.com/docs/index.html)
  + “*Java Tutorial*” at [*http://java.sun.com/docs/books/tutorial/index.html*](http://java.sun.com/docs/books/tutorial/index.html)
  + For an introduction to creating enhanced user interfaces, see “*Creating a GUI with JFC/Swing*” at [*http://java.sun.com/docs/books/tutorial/uiswing/index.html*](http://java.sun.com/docs/books/tutorial/uiswing/index.html)
  + *Java in a Nutshell, 2nd Edition* from O’Reilly & Associates Inc. See the O’Reilly Java Resource Center at [*http://java.oreilly.com*](http://java.oreilly.com/).
  + For a tutorial on XML, visit [*http://www.w3schools.com/xml/*](http://www.w3schools.com/xml/).
  + Resources for using JavaBeans are at [*http://java.sun.com/beans/resources.html*](http://java.sun.com/beans/resources.html)

These resources are not required to develop applications using JClass Gauge, but they do provide useful background information on various aspects of the Java programming language and XML.

#### About Quest Software, Inc.

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

Contacting Quest Software

|  |  |
| --- | --- |
| **Email** | [*info@quest.com*](mailto:info@quest.com) |
| **Mail** | Quest Software, Inc. World Headquarters 5 Polaris Way  Aliso Viejo, CA 92656 USA |
| **Web site** | [*www.quest.com*](http://www.quest.com/) |

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

Contacting Quest Support

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

From SupportLink, you can do the following:

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

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

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

##### JClass Community

For the latest product information, helpful resources, and discussions with theJClass Quest team and other community members, join the JClass community at [http://jclass.inside.quest.com/.](http://jclass.inside.quest.com/)

# 1

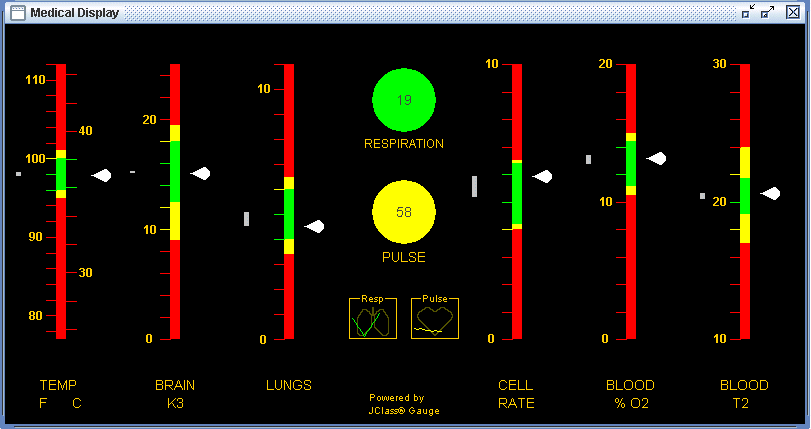
## Learning JClass Gauge Basics

[*Types of Gauges*](#_bookmark23)■ [*Graph Component for Gauges*](#_bookmark30)■ [*Containment Hierarchies*](#_bookmark31)[*Key Concepts*](#_bookmark34)■ [*Key Concepts for Circular and Linear Gauges*](#_bookmark42)

[*Internationalization*](#_bookmark47)

JClass Gauge provides highly customizable gauge components that you can use to implement some very effective *user interface metaphors*. A user interface metaphor maps your data on to a generally known environment. When done well, metaphors can make your application easier to learn and use.

For example, the following image is from the *Medical Display* demo (see [Example & Demo](#_bookmark8) [Gallery on page 14](#_bookmark8)). This application uses a combination of linear gauges and indicator gauges as well as the JCGraph component to create a display that monitors the health of a patient. You could reuse this user interface metaphor to monitor the “health” of, say, investments or stocks.



Linear Gauges Indicator Gauges JCGraph

*Figure 2 Medical Display implements linear gauges, indicator gauges, and the JCGraph component*

This chapter introduces each of the gauge types and the graph component for gauges. The gauge containment hierarchies are included here as well as key concepts that you should know before building gauges.

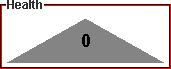
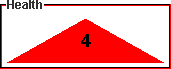
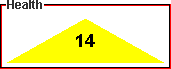
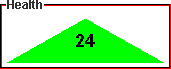
#### Types of Gauges

There are three types of gauges: indicator gauges, circular gauges, and linear gauges. The circular and linear gauges share a common code base, and it is easy to convert from one to the other.

##### Indicator Gauge

Indicator gauges are used to show a value within the context of a range of values. When the value assigned to the gauge enters a new range, the icon changes to reflect the color, image, and/or text associated with that range. For example, you may have an indicator gauge that you want to use to highlight a poor, fair, and good result. You can tie poor results to an image showing a frowning face, fair results to a smiling face, and good to a laughing face. Or use red, yellow, and green to implement a traffic light metaphor.

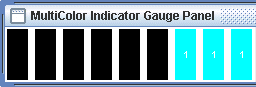
For example, the *Dungeon Game* demo (see [Example & Demo Gallery on page 14](#_bookmark8)) uses a triangular indicator gauge to show the health status of the character in the game. Green is good, yellow is fair, red is poor, and gray means the character is dead.



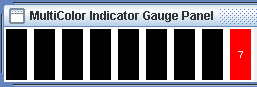
*Figure 3 The same indicator gauge using color to highlight health status: good, fair, poor, dead*

You can also create a panel of indicator gauges. A panel lets you represent the value as a binary number, an integer, or as magnitude in a scale. For example, the following figure shows the value 7 displayed in four different ways in a horizontal indicator gauge panel.

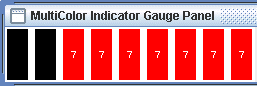
Binary



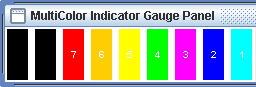
Decimal



Unichrome (magnitude)



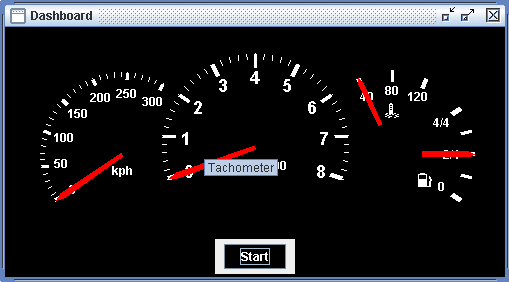
Multichrome (magnitude)



*Figure 4 Indicator gauge panels showing the value 7 in binary, decimal, and magnitude-type formats*

##### Circular Gauge

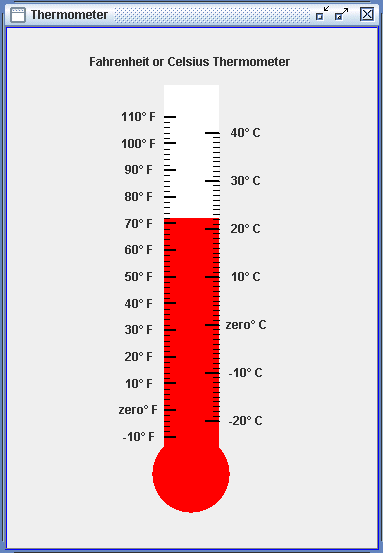
Circular gauges present a value in the context of a circular graduated scale. The gauge can be an entire circle or a partial circle. You use indicators or needles to point out values on the scale. You can use circular gauges to give your application the flavor of an automobile instrument cluster, airline cockpit, control room, old time radio, or any other metaphor suitable for circular gauges.



*Figure 5 Dashboard created using four circular gauges*

##### Linear Gauge

Linear gauges present a value in the context of a vertical or horizontal graduated scale. You use an indicator or needle to point to values on the scale. Linear gauges can be made to look like thermometers (for hot and not-so-hot stock opportunities as well as temperature measurement), jazzed-up progress meters, and level and volume indicators.



*Figure 6 Sample linear gauge*

#### Graph Component for Gauges

JClass Gauge also comes with a light-weight graph component, called JCGraph, that is useful in real-time applications. The graph displays the relative fluctuations in a changing value. The graph is a JComponent. It can be placed in a JPanel or layered on top of a gauge. For more information, see [Chapter 3, “Creating Graphs.”](#_bookmark125)

#### Containment Hierarchies

All the gauges and the graph are derived, directly or indirectly, from JComponent. This section outlines the containment hierarchies for each type of gauge. The circular and linear gauges are treated together because they derive from a common object and share other objects.

##### Indicator Gauge Containment Hierarchy

The indicator gauge is founded upon JCBaseIndicatorGauge, which extends JComponent. JCBaseIndicatorGauge contains a JCIndicatorIcon object that is controlled via the IconController interface.

JCBaseIndicatorGauge is used by both the standalone indicator gauge and the indicator gauge panel. The standalone indicator gauge, called JCIndicatorGauge, extends JCBaseIndicatorGauge. The indicator gauge panel, called JCIndicatorGaugePanel (extended from JPanel), contains one or more instances of a JCBaseIndicatorGauge object.

JCIndicatorIcon

instances of a

JCBaseIndicatorGauge

**Indicator Gauge Panel Containment Listing**

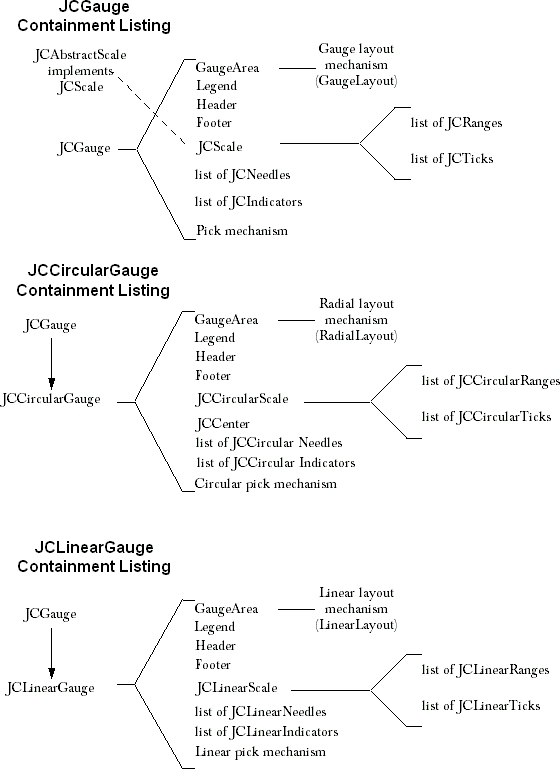
JCIndicatorGaugePanel

**Indicator Gauge Containment Listing**

JCIndicatorGauge JCBaseIndicatorGauge JCIndicatorIcon

*Figure 7 Indicator gauge containment hierarchy*

##### Circular and Linear Gauges Containment Hierarchy

In keeping with the goal of making the circular and linear gauges as configurable as possible, the classes inherit from an abstract class JCGauge, which is itself a JComponent. The following diagram shows where you can find the components and sub-objects that make up a circular or linear gauge. Objects are designated by their class names.

*Figure 8 JCGauge containment hierarchy*

JCGauge is the abstract superclass for JCCircularGauge and JCLinearGauge. JCGauge creates a header, footer, and legend whenever a circular or linear gauge is instantiated, and it has methods for adding or removing indicators, needles, ranges, and ticks.

Use the gauge’s add() methods rather than the add() methods in JComponent. The addLabel() methods in JCCircularGauge and JCLinearGauge are there because these objects depend on LinearConstraint and RadialConstraint classes to position labels based on a linear extent and a pixel value, or a specified angle and radial distance. For more information, see [Section 8.6, “Understanding the Constraint Mechanism,” on page](#_bookmark347) [131](#_bookmark347).

At any given time, each gauge contains one scale. A scale holds information about the minimum and maximum values for the scale, and the direction, forward or backward, in which scale values increase. The circular scale has start and stop angles that determine the portion of a full circle occupied by the scale. A needle object provides the visual indication of a particular value on the scale. Its length is specified by setting its *inner extent* and its *outer extent*. If the scale changes its size, the needle adjusts itself accordingly, maintaining a proper position and proportional length relative to the size of the scale. For more information, see [Section 1.5.2, “Sizing Components Using Extent Parameters,” on](#_bookmark45) [page 29](#_bookmark45).

#### Key Concepts

The following concepts apply to all the gauge types.

##### Gauge Size

A gauge’s size is determined by the size of its container. You can set the initial size of the container using its size or preferredSize property. Gauges may be resized as long as the container layout manager permits it.

##### Images and the PortableImage Object

JClass Gauge supports the use of images in the background of containers. This means that you can place an image in the background of a gauge, and also in the background of some of the gauge components. For example, in a circular gauge, you can set an image in the background of the gauge, the scale, and a range. In addition, JClass Gauge also supports a foreground image for linear ranges only.

To add an image, you specify it as a java.awt.Image or java.net.URL object using the container’s image property. JClass Gauge stores the image in a PortableImage object – a wrapper object in JClass DesktopViews that stores an image and properties related to that image. A PortableImage object has as its properties the name of the image, whether or not the image is scaled, and how information about the image is saved to XML.

The preferred way to add images is to set the container’s image property. This is the method used in the procedures within this guide. You can, however, choose to provide images via PortableImage objects using the portableImage property. For example, the following code creates a new PortableImage and then sets it as the background image of a gauge.

// Create and use a simple PortableImage object

// The image is scaled by default PortableImage image1 = new PortableImage(

new URL(“http://www.my\_site.com/smallcircles.jpg”)); myGauge.setPortableImage(image1);

For more information, look up com.klg.jclass.io.PortableImage in the

*JClass DesktopViews API documentation*.

##### Layering Gauges

You can layer other gauges on top of circular and linear gauges. A secondary gauge can be added using any of the gauge types, including the same gauge type as the host gauge. From a usability perspective, you want to make sure that the data you are presenting is easy to understand, so bear in mind that each additional gauge adds complexity and possibly reduces the overall usability of the gauge. For example, if your user can interact with layered gauges, you need to figure out which needle object to get and manipulate.

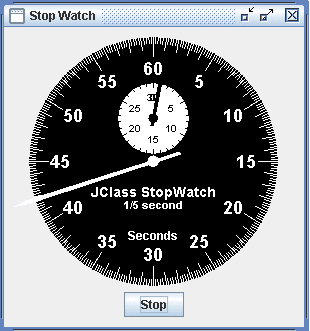
Note: Indicator gauges cannot contain other gauges, but may be added to circular and linear gauges.

To add a gauge as a label to a host gauge, you use the addLabel() method from the host gauge and specify the secondary gauge object and its location. The location is defined using constraints. For more information, see [Section 8.6, “Understanding the Constraint](#_bookmark347) [Mechanism,” on page 13](#_bookmark347)1.

For example, consider a case where you need to provide a smaller circular gauge within a larger one, such as a stopwatch whose larger scale counts off seconds and whose smaller scale indicates the number of minutes that have elapsed. In the code snippet that follows, bigGauge is the one containing the second hand and smallGauge is the one containing the minute hand. It does not show the details of setting up the properties of the two scales, but once they are configured it is easy to place the smaller gauge within the larger.

JCCircularGauge bigGauge; JCCircularGauge smallGauge; bigGauge.addLabel(smallGauge,

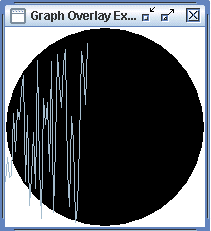
new RadialConstraint(smallGauge, 0.35, 90), -1);



*Figure 9 One circular gauge within another*

##### Layering a Graph on Top of a Gauge

You can layer a JCGraph component on top of another component in the same way that you do with a standard JComponent. For example, the *GraphOverlayExample* uses Swing’s OverlayLayout to layer a graph on a gauge.



*Figure 10 A graph on top of an indicator gauge*

#### Key Concepts for Circular and Linear Gauges

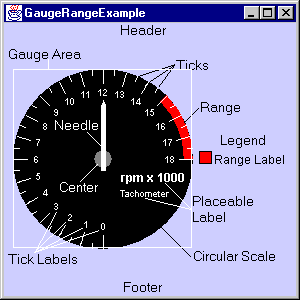
The circular and linear gauges offer the following components:

* Header. A header provides a title for the gauge. Headers are JComponents, and by default they are JLabels.
* Footer. A footer provides another option for titling a gauge. It too is a JComponent, and by default a JLabel.
* Placeable Labels. Any component may be placed on a gauge at a specified position by employing an add() method that takes a LinearConstraint or RadialConstraint as its second parameter. For more information, see [Section 8.6, “Understanding the Constraint Mechanism,” on page 13](#_bookmark347)1.
* Scale. A scale on which values may be defined. The scale can have associated

JCTick, JCNeedle, JCIndicator, and JCRange objects.

* Tick Marks and Tick Labels. A tick object is used to show the scale values. It is a collection of uniformly spaced marks and/or labels.
* Indicators and Needles. An indicator is a static marker placed at a particular scale value. A needle is a special type of indicator. Needle positions are dynamic and you can enable user-interaction for needles.
* Ranges. A range specifies a range of values on the scale and associates color or other attributes with that range.
* Gauge Constraint and Gauge Layout. These classes assist in placing some of the above components on the gauge.

The following figure shows the components in a gauge. Note that the gauge area (JCGaugeArea) contains the scale and its objects, but not the header, footer, or legend.



*Figure 11 Components of a JClass Gauge*

You have considerable flexibility in designing the appearance of your gauge. Interfaces for indicators, needles, ranges, scales, and ticks let you replace the built-in objects with those you design yourself. As well, it is easy to add additional items like numerical counters and images to the gauge.

##### How Components are Rendered on a Gauge

A render list, which in Java is often called the *z*-order of the components, is effectively created by the order in which components are added at execution time. This list determines in what order child objects are to be drawn. Components added last are drawn first. For example, in a circular scale, adding a needle and then a center allows the center object to be drawn first, then a needle, making the needle fully visible from center to tip instead of being partially covered by the center object. There are ways of manipulating the list so that a different drawing order can be specified. By drawing a needle first, it can appear to be attached to the edge of the center object rather than beginning at the center of the circle.

Caution: When setting a scale on a gauge, a center on a circular gauge, or adding indicators, needles, ranges, and ticks to a gauge, you must use the set() and add() methods provided in JCGauge; the standard java.awt.Container methods do not fulfill JCGauge’s requirements. When using the gauge’s add() method, you are still able to include an optional index that specifies the rendering order.

A gauge may have multiple instances of indicators, needles, ranges, and tick marks. In this case, the gauge maintains a collection for each of the object groups. The gauge has indicator and needle lists for marking one or more scale values, a range list for keeping track of the ranges used to mark regions of the scale, and a tick list to keep track of the different tick objects.

Labels, which may be any JComponent, not just a JLabel, are created and manipulated individually rather than being stored in a list. There are special-purpose methods called addLabel(label, radialConstraint) and addLabel(label, linearConstraint) for placing labels at a specified location on the gauge. For more information, see [Section 8.6,](#_bookmark347) [“Understanding the Constraint Mechanism,” on page 13](#_bookmark347)1.

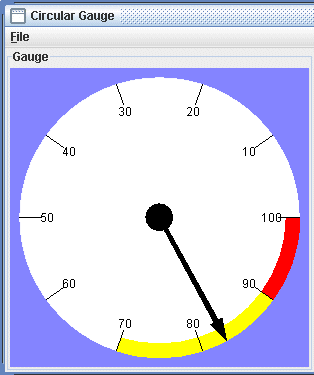
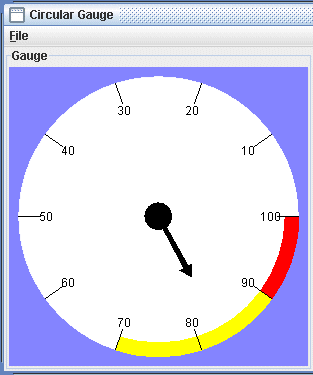
##### Sizing Components Using Extent Parameters

Some components can be sized by specifying two object parameters, called the *inner extent* and the *outer extent*. Extents are defined as ratios based on the underlying scale used for the gauge. This means that if the gauge is resized, the components that use extent parameters will adjust proportionately to suit the size of the gauge. The way that extents are measured differs depending on the type of gauge that you are creating.

Circular Gauge Extents

In the circular case, inner and outer extents refer to locations in a *radial* direction, with the center defined as 0.0. Thus, an object with an inner extent of 0.0 means that it is drawn from the center outwards to the position defined by the outer extent. If this outer extent is 0.8, the object extends out from the center a distance equal to 80% of the radius of the circular scale.

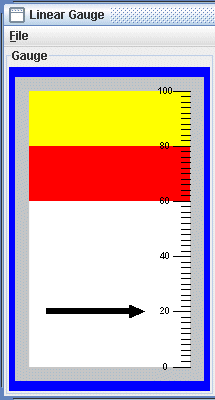
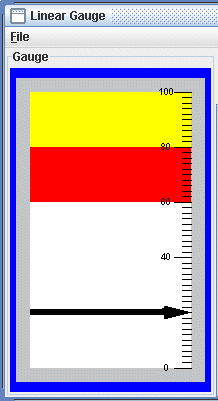
For example, the following images show the same circular gauge. The outer extent of the needle in the left-hand image is set to 1.0, which means that the needle stretches from the origin to the circumference of the scale. The right-hand image shows a needle with an outer extent of 0.5; the needle extends only halfway to the circumference.

*Figure 12 Comparing needle outer extents in a circular gauge: Left: outer extent of 1.0 Right: outer extent of 0.5*

Linear Gauge Extents

In the linear case, extents are measured in the direction transverse to the direction in which scale values increase. In a horizontal scale, extents are therefore measured from the top edge of the scale, while in a vertical scale, they are measured from the left edge. For example, an indicator on a vertical linear scale whose inner extent is 0.15 and whose outer extent is 0.75 is drawn beginning at 15% of the gauge’s width from the left edge to 75% of the gauge’s width.



*Figure 13 Comparing needle extents in a linear gauge: Left: inner extent of 0.0 and outer extent of 1.0*

*Right: inner extent of 0.15 and outer extent of 0.75*

#### Internationalization

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

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

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

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

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

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

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

For more information, including standard language and country identifiers, see

[*http://java.sun.com/j2se/1.4.2/docs/guide/intl/index.html*](http://java.sun.com/j2se/1.4.2/docs/guide/intl/index.html).

If you are creating XML-based gauges, you can internationalize the text on the gauge using variables and a resource bundle. For more information, see [Section 11.6,](#_bookmark449) [“Internationalizing Your XML-based Gauge,” on page 16](#_bookmark449)0.

*Part* *I*

*Indicator Gauges and Graphs*

***2***

## Creating Indicator Gauges and Panels

[*Overview of the Indicator Gauge Objects*](#_bookmark55)■ [*Summary of Properties Used*](#_bookmark57)[*A Sample Standalone Indicator Gauge*](#_bookmark58)■ [*Creating a Standalone Indicator Gauge*](#_bookmark62)[*Customizing the Behavior of Your Indicator Gauge*](#_bookmark68)■ [*A Sample Indicator Gauge Panel*](#_bookmark76)[*Creating an Indicator Gauge Panel*](#_bookmark80)■ [*Customizing the Behavior of Your Panel*](#_bookmark89)

[*Customizing the Appearance of Gauges and Panels*](#_bookmark108)

An indicator gauge is an effective way to draw your end-user’s attention to an important value. The standalone indicator gauge displays a single gauge, which can be a rectangle, oval, or a triangle. You can enhance the gauge by defining ranges with attributes such as color, images, and text. When the value represented by the gauge enters a new range, the gauge changes to reflect the attributes associated with that range. You can also customize the appearance of the gauge to suit your application.

In addition to the standalone indicator gauge, you can create a panel of indicator gauges. You create and replicate a gauge on a specialized indicator gauge panel. The value is tied to the panel (rather than a gauge) and ranges are defined for the panel depending on the *value type*. The value type means that you can use the gauges to display a value as a binary number, a decimal, or as magnitude in a scale.

This chapter describes how to add standalone indicator gauges and indicator gauge panels to your interface and describes how to customize them.

#### Overview of the Indicator Gauge Objects

The following diagram shows the inheritance for indicator gauge objects.

JCIndicatorIcon

java.lang.Object

JCIndicatorGaugePanel

javax.swing.JPanel

JCIndicatorGauge

IconController

ActionListener

JCBaseIndicatorGauge

ComponentListener

Cloneable

javax.swing.JComponent

Accessible

*Figure 14 Indicator gauge object inheritance*

JCBaseIndicatorGauge extends JComponent and implements Accessible, ComponentListener, and Cloneable. The JCBaseIndicatorGauge constructor creates an indicator gauge that can be replicated within an instance of JCIndicatorGaugePanel. The class provides properties and methods to define some elements of the gauge, including the icon to display (an instance of JCIndicatorIcon), and background elements. Other gauge elements are controlled by the panel, JCIndicatorGaugePanel.

JCIndicatorGaugePanel extends JPanel and implements IconController and ActionListener. It contains properties and methods to define the size and orientation of the panel, its background, the number of copies of the gauge (created with JCBaseIndicatorGauge), the value type (for example, binary format), the direction in which the gauge values increase, value ranges with text and color attributes, and the value that the panel of gauges represents.

JCIndicatorGauge extends JCBaseIndicatorGauge and implements IconController and ActionListener. The JCIndicatorGauge constructor creates an indicator gauge as a standalone component (that is, without using the panel). The classes provides properties and methods to define the gauge, including the icon to display (an instance of JCIndicatorIcon), value ranges with text and color attributes, and the value that the gauge represents.

IconController is a JClass interface that specifies the methods required to control the icon’s appearance.

JCServerGauge is a JClass interface that specifies the methods required to define a gauge or graph component.

JCIndicatorIcon implements Cloneable. It encapsulates attributes of the icon to be displayed in the gauge such as shape, color, image, text, and value. Generally speaking, the public methods in JCIndicatorIcon are not used directly. JCIndicatorGauge and JCIndicatorGaugePanel use these methods (via the methods provided by the IconController interface) to draw an icon that reflects the current value.

#### Summary of Properties Used

The following table summarizes the properties (grouped where appropriate) used in this chapter and provides links to the associated procedures. For a list of all the properties and methods available for indicator gauge objects, see the *API documentation*.

|  |  |
| --- | --- |
| **Properties Used** | **Procedures** |
| baseValue | [Section 2.5.2, “Setting the Base Value for a Standalone Gauge,”](#_bookmark74) [on page 45](#_bookmark74)  [Section 2.8.4, “Setting the Base Value for a Panel,” on page 56](#_bookmark100) |
| blinkInterval startBlinking stopBlinking | [Section 2.9.7, “Making Gauge Icons Blink,” on page 63](#_bookmark122) |
| background opaque | [Section 2.9.4, “Setting a Background Color,” on page 60](#_bookmark116) |
| iconFont iconForeground | [Section 2.9.6, “Setting the Font and Color for Text,” on page 63](#_bookmark120) |
| iconShape | [Section 2.9.1, “Setting the Icon Shape,” on page 58](#_bookmark110) |
| image imageScaled | [Section 2.9.5, “Specifying and Scaling an Image,” on page 62](#_bookmark118) |
| padding | [Section 2.9.3, “Setting the Padding,” on page 59](#_bookmark114) |
| preferredSize | [Section 2.9.2, “Setting the Gauge Size,” on page 59](#_bookmark112) |

|  |  |
| --- | --- |
| **Properties Used** | **Procedures** |
| rangeValues rangeDisplays rangeText rangeToolTips textValueDisplayed | [Section 2.5.1, “Defining Ranges for a Standalone Gauge,” on](#_bookmark70) [page 42](#_bookmark70)  [Section 2.8.1, “Setting the Value Type and Defining Ranges for](#_bookmark91) [a Panel,” on page 50](#_bookmark91) |
| ***Panel Only Properties*** |  |
| direction | [Section 2.8.2, “Setting the Panel Direction,” on page 55](#_bookmark96) |
| gauge | [Section 2.8.5, “Changing the Gauge on the Panel,” on page 57](#_bookmark104) |
| numGauges | [Section 2.8.6, “Changing the Number of Gauges Displayed,”](#_bookmark106) [on page 57](#_bookmark106) |
| orientation | [Section 2.8.3, “Setting the Panel Orientation,” on page 56](#_bookmark99) |
| valueType | [Section 2.8.1, “Setting the Value Type and Defining Ranges for](#_bookmark91) [a Panel,” on page 50](#_bookmark91) |

#### A Sample Standalone Indicator Gauge

This section contains a sample indicator gauge and the code used to create it. When you create a standalone indicator gauge, the gauge uses default values for all its properties. You can assign a value to it after it is constructed. After you set up this very basic gauge, you can choose to customize the gauge so that its appearance changes when the current value changes.

For general instructions on creating your own gauge, see these sections:

* + - [Section 2.4, “Creating a Standalone Indicator Gauge,” on page 41](#_bookmark62)
    - [Section 2.5, “Customizing the Behavior of Your Indicator Gauge,” on page 42](#_bookmark68)
    - [Section 2.9, “Customizing the Appearance of Gauges and Panels,” on page 58](#_bookmark108)

##### Sample Gauge

The following images show a sample indicator gauge that implements a traffic light metaphor. As the value goes down, the status message and background color changes.

###### 

*Figure 15 Four images of the sample indicator gauge representing each of its range attributes*

##### Sample Code

The following program creates the sample indicator gauge.

This is a modified version of *SimpleIndicatorGauge.java* located in *JCLASS\_HOME/examples/gauge/indicator/*. Please see the example to review the code that has been omitted here.

package examples.gauge.indicator;

import com.klg.jclass.gauge.indicator.JCIndicatorGauge; import com.klg.jclass.gauge.indicator.JCIndicatorIcon; import com.klg.jclass.gauge.JCGaugeEnumMappings;

import com.klg.jclass.util.swing.JCExitFrame; import com.klg.jclass.util.JCTypeConverter;

import javax.swing.\*; import java.awt.\*;

import java.awt.event.ActionListener; import java.awt.event.ActionEvent;

/\*\*

* Creates a Simple JCIndicatorGauge. The value changes based
* on button clicks which either add to or subtract from the current value.

\*/

public class SimpleIndicatorGauge extends JPanel

{

private JCIndicatorGauge myGauge;

/\*\*

* Creates the JCIndicatorGauge Example

\*/

public SimpleIndicatorGauge()

{

setLayout(new BorderLayout());

// Create a JCIndicatorGauge with an oval shaped icon.

myGauge = new JCIndicatorGauge(JCIndicatorIcon.SHAPE\_OVAL);

// Set the base value (lowest value that can be displayed) myGauge.setBaseValue(new Integer(-1));

// Set the ranges and their attributes myGauge.setRangeValues(new Integer[]{new Integer(-1),

new Integer(3), new Integer(6), new Integer(10)});

myGauge.setRangeDisplays(new Color[]{Color.gray,

Color.red, Color.yellow, Color.green});

// Set tooltip text myGauge.setRangeToolTips(new String[]{

"No status", "Urgent action required", "Monitor performance", "Good"});

// Set the font and color to use for icon text myGauge.setIconFont(new Font("Dialog", Font.BOLD, 18)); myGauge.setIconForeground(Color.black);

// Specify the text myGauge.setTextValueDisplayed(false); myGauge.setRangeText(new String[]{

"No Status", "Urgent", "Monitor", "Good"});

// Set a starting value myGauge.setValue(new Integer(7));

myGauge.setPadding(5); myGauge.setBackground(Color.white);

add(myGauge, BorderLayout.CENTER);

// Create a frame and place the gauge within it

...

// Create buttons to enable a user to increment/decrement the value

// and set the value of the gauge accordingly

// (See the SimpleIndicatorGauge.java example that ships

// with JClass Gauge)

...

}

// Run the example

public static void main(String[] args)

{

final JCExitFrame frame = new JCExitFrame("Simple JCIndicatorGauge"); frame.getContentPane().add(new SimpleIndicatorGauge());

frame.pack(); frame.setVisible(true);

}

}

#### Creating a Standalone Indicator Gauge

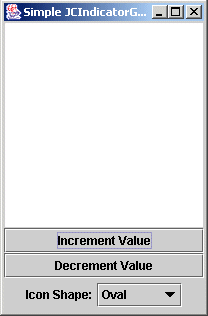
This section describes how to create a standalone indicator gauge and assign a value to it.

##### Constructing a JCIndicatorGauge Object

The JCIndicatorGauge constructor creates a gauge that can be displayed as a standalone component. You can use the constructor with no arguments to create an indicator gauge with the default rectangular shape:

// Create a standalone gauge with default icon shape JCIndicatorGauge myGauge = new JCIndicatorGauge();

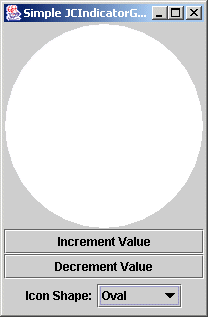
The constructor in turn creates the icon that appears on the gauge as an instance of JCIndicatorIcon. The icon is, by default, a rectangle that extends to within one pixel of the borders of its container.



*Figure 16 Default standalone indicator gauge shown inside a JCExitFrame container*

Alternatively, you can specify the shape of the icon in the constructor. For more information, see [Section 2.9.1, “Setting the Icon Shape,” on page 58](#_bookmark110).

// Specify a shape for the icon when creating a gauge JCIndicatorGauge myGauge =

new JCIndicatorGauge(JCIndicatorIcon.SHAPE\_OVAL);

*Figure 17 Default standalone indicator gauge shown inside a JCExitFrame container*

##### Assigning a Value to Your Indicator Gauge

To set the current value of the gauge, set the value property and specify an Integer

object.

For example:

// Create a rectangular gauge and set the value explicitly JCIndicatorGauge myGauge = new JCIndicatorGauge(); myGauge.setValue(new Integer(7));

By default, the gauge contains no text. You can, however, choose to display either the current value or some other text on the gauge. For more information, see [Section 2.5.1,](#_bookmark70) [“Defining Ranges for a Standalone Gauge,” on page 42](#_bookmark70).

##### Next Steps

After you complete the necessary tasks of constructing the gauge and assigning a value to it, you can decide how you want the gauge to behave and look.

To customize how the gauge behaves, you can choose from the following tasks:

* + - * [Section 2.5.1, “Defining Ranges for a Standalone Gauge,” on page 42](#_bookmark70)
      * [Section 2.5.2, “Setting the Base Value for a Standalone Gauge,” on page 45](#_bookmark74)

To customize the appearance of the gauge, you can choose from the following tasks:

* + - * [Section 2.9.1, “Setting the Icon Shape,” on page 58](#_bookmark110)
      * [Section 2.9.2, “Setting the Gauge Size,” on page 59](#_bookmark112)
      * [Section 2.9.3, “Setting the Padding,” on page 59](#_bookmark114)
      * [Section 2.9.4, “Setting a Background Color,” on page 60](#_bookmark116)
      * [Section 2.9.5, “Specifying and Scaling an Image,” on page 62](#_bookmark118)
      * [Section 2.9.6, “Setting the Font and Color for Text,” on page 63](#_bookmark120)
      * [Section 2.9.7, “Making Gauge Icons Blink,” on page 63](#_bookmark122)

#### Customizing the Behavior of Your Indicator Gauge

This section describes how to define ranges and change the base value of the gauge. To learn how to change the appearance of the gauge, see [Section 2.9, “Customizing the](#_bookmark108) [Appearance of Gauges and Panels,” on page 58](#_bookmark108).

##### Defining Ranges for a Standalone Gauge

You can define ranges of values for your gauge and then associate different icon attributes with each range. The icon attributes are display object, text, and tooltips. The display object can be a color or an image. The text can be the current value or some other text

that you specify. The tooltips display the RGB value of the fill color by default, but you can specify your own text instead. When you run your application and the current value falls into a new range, the gauge redraws the icon with the attributes specified for that range.

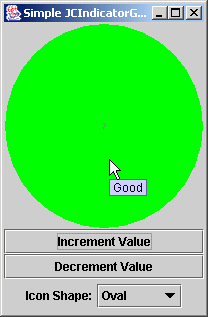
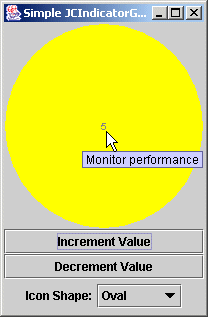
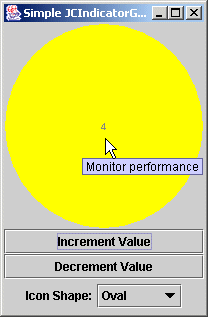
To set range values, set the rangeValues property to an array of values of the same type as the value property used by the icon. To set the icon display object, set the rangeDisplays property to an array of Color, URL, or PortableImage objects (or a mix thereof), where the URL or PortableImage objects reference images. To specify tooltips, set the rangeToolTips property to a String[] object with an array of text strings.

You have two choices for displaying text on the icon. To display the current value, set the textValueDisplayed property to true. To specify custom text, set the rangeText property to a String[] object with an array of text strings. If you want to change the font used for icon text, see [Section 2.9.6, “Setting the Font and Color for Text,” on page 63](#_bookmark120).

For example, the following table outlines the ranges and range attributes used in the sample indicator gauge:

|  |  |  |  |
| --- | --- | --- | --- |
| **Range** | **Display Object** | **Text** | **Tooltip** |
| -1 | Color.gray | *<current value>* | No status |
| 0–3 | Color.red | *<current value>* | Urgent action |
| 4–6 | Color.yellow | *<current value>* | Monitor performance |
| 7–10 | Color.green | *<current value>* | Good |

When the current value is 4, the icon is yellow, displays the numeral 4, and the tooltip says “Monitor performance.” If the value becomes 5, the numeral changes but the other icon attributes remain the same. When the current value enters a new range, all the icon attributes change accordingly. So if the value becomes 7, the gauge redraws the icon in green, with the numeral 7, and the tooltip “Good.”



*Figure 18 Gauge attributes change when the value enters a new range*

In the preceding figure, the default font used to display the value on the gauge is hard to read. To change the font style and size, see [Section 2.5.1.1, “Specifying Custom Text on](#_bookmark72) [the Icon,” on page 44](#_bookmark72).

The following code implements the ranges used in the preceding example.

// Define ranges myGauge.setRangeValues(new Integer[]{

new Integer(-1), new Integer(3), new Integer(6), new Integer(10)});

// Set Colors for each range myGauge.setRangeDisplays(new Color[]{

Color.gray, Color.red, Color.yellow, Color.green});

// Set tooltip text myGauge.setRangeToolTips(new String[]{

"No status", "Urgent action required", "Monitor performance", "Good"});

// Display the current value on the icon myGauge.setTextValueDisplayed(true);

* + - 1. Specifying Custom Text on the Icon

If you want to display text other than the current value, change the textValueDisplayed property to false and set the rangeText property to a String[] object with an array of text strings.

// Set the font and color to use for icon text myGauge.setIconFont(new Font("Dialog", Font.BOLD, 18)); myGauge.setIconForeground(Color.black);

// Specify the text myGauge.setTextValueDisplayed(false); myGauge.setRangeText(new String[]{

"No Status", "Urgent", "Monitor", "Good"});

See [Figure 15](#_bookmark60) to view the results of specifying text, font, and font color.

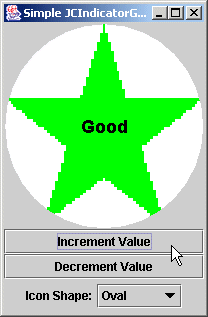
* + - 1. Specifying Images on the Icon

If you want to use an image instead of a color on the icon, set the rangeDisplays property to an array of URL or PortableImage objects. For more information, see [Section 2.9.5,](#_bookmark118) [“Specifying and Scaling an Image,” on page 62](#_bookmark118).

For example

// Use Images instead of Colors for each range myGauge.setRangeDisplays(new URL[]{

this.getClass().getResource("star\_silver.gif"), this.getClass().getResource("star\_red.gif"), this.getClass().getResource("star\_yellow.gif"), this.getClass().getResource("star\_green.gif")});



*Figure 19 Using an image for the icon*

Alternatively, you can specify an array of Objects that contains a mix of Color, URL, and

PortableImage objects.

By default, the icon image scales with its container. To turn off scaling, set the

imageScaled property to false.

// Do not scale the image on the icon myGauge.getIcon().setImageScaled(false);

Note that if you provide an array of PortableImage objects, the icon’s setImageScaled() method is ignored. The imageScaled property for each PortableImage object is used instead. For more information, see [Section 1.4.2, “Images and the PortableImage Object,”](#_bookmark37) [on page 25](#_bookmark37).

##### 2.5.2 Setting the Base Value for a Standalone Gauge

The base value is the default value of the gauge. It is also the lowest value that the gauge can display. By default, the baseValue property is set to Integer(0).

To change the base value, set the baseValue property to an Object of the same type used for the value property, usually an Integer object. You need to set the baseValue property before setting the rangeValues property. For more information, see [Section 2.4.2,](#_bookmark65) [“Assigning a Value to Your Indicator Gauge,” on page 42](#_bookmark65).

For example:

// Set a minimum value of -1 myGauge.setBaseValue(new Integer(-1));

If the current value ever becomes less than the base value, the current value is ignored and the gauge continues to display the last valid value. For example, the base value used for the sample indicator gauge is -1. Assume a current value of 7. If the current value becomes -2, that is setValue(-2), the value is ignored and the gauge does not change. Calling getValue() at this point would return 7.

#### A Sample Indicator Gauge Panel

This section contains a sample indicator gauge panel and the code used to create it. When you create an indicator gauge panel, you specify the number of gauges to display in the JCIndicatorGaugePanel constructor. All other properties are assigned default values.

This means that, by default, you see a horizontal panel containing the specified number of gauges. The gauges are displayed as black rectangles and represent zero in binary format. When you assign a value to the panel, the gauges change to reflect to the new value in binary format.

After you set up the basic panel, you can choose to customize how the value is displayed, so that instead of having the value represented as a binary number, you could represent it in decimal format (whole, positive numbers only) or as magnitude on a scale (using a single color or multiple colors).

For general instructions on creating your own indicator gauge panel, see these sections:

* [Section 2.7, “Creating an Indicator Gauge Panel,” on page 48](#_bookmark80)
* [Section 2.8, “Customizing the Behavior of Your Panel,” on page 50](#_bookmark89)
* [Section 2.9, “Customizing the Appearance of Gauges and Panels,” on page 58](#_bookmark108)

Note: You can also create a panel of indicator gauges using a standard JPanel. In this case, you set the value of each indicator gauge separately. You can find an example in *JCLASS\_HOME/examples/gauge/indicator/Toggle.java*.

##### Sample Panel

The following figure shows how an indicator gauge panel can be used to implement a five-star rating system. The sample indicator gauge panel is implemented with five gauges in a horizontal orientation. The direction of the panel is backward, which means that the values associated with the gauges increase from left to right.

The value displayed by the panel is 4, hence the four stars. The top image shows a *unichrome* implementation, which means the stars are the same color. The bottom image shows the same indicator gauge panel using a *multichrome* implementation.



*Figure 20 Five star rating system implemented in unichrome (top) and multichrome (bottom)*

##### Sample Code

The following code creates the unichrome version of the sample indicator gauge panel.

package examples.gauge.indicator;

import com.klg.jclass.util.swing.JCExitFrame;

import com.klg.jclass.gauge.indicator.JCIndicatorGaugePanel; import com.klg.jclass.gauge.indicator.JCBaseIndicatorGauge; import com.klg.jclass.gauge.indicator.JCIndicatorIcon;

import javax.swing.\*; import java.awt.\*; import java.net.URL;

public class SimpleIndicatorGaugePanel extends JPanel

{

private JCIndicatorGaugePanel myPanel = null; private JCBaseIndicatorGauge myGauge = null;

public SimpleIndicatorGaugePanel()

{

setLayout(new BorderLayout()); setPreferredSize(new Dimension(375, 75));

// Configure a gauge of type JCBaseIndicatorGauge JCBaseIndicatorGauge myGauge =

new JCBaseIndicatorGauge(JCIndicatorIcon.SHAPE\_RECTANGLE);

// Create the indicator gauge panel and display five gauges myPanel = new JCIndicatorGaugePanel(5, myGauge);

// Set the value type: UNICHROME or MULTICHROME myPanel.setValueType(

JCIndicatorGaugePanel.VALUE\_MAGNITUDE\_UNICHROME);

// Configure the appearance myPanel.setOrientation(

JCIndicatorGaugePanel.ORIENTATION\_HORIZONTAL); myPanel.setDirection(JCIndicatorGaugePanel.DIRECTION\_BACKWARD); myPanel.setPadding(4);

myPanel.setBackground(Color.white);

// Configure the range attributes for the five-star rating system myPanel.setRangeValues(new Integer[]

{new Integer(0), new Integer(1), new Integer(2), new Integer(3), new Integer(4), new Integer(5)});

myPanel.setRangeDisplays(new URL[]{

this.getClass().getResource("star\_none.gif"), this.getClass().getResource("star\_cyan.gif"), this.getClass().getResource("star\_blue.gif"), this.getClass().getResource("star\_bronze.gif"), this.getClass().getResource("star\_silver.gif"), this.getClass().getResource("star\_gold.gif")});

myPanel.setRangeToolTips(new String[]{

"", "Poor", "Okay", "Good", "Great", "Exceptional"});

// Scale the image on the icon when the container scales myGauge.getIcon().setImageScaled(true);

myPanel.setIconForeground(Color.white); myPanel.setValue(new Integer(4));

add(myPanel, BorderLayout.CENTER);

}

// Run the example

public static void main(String[] args)

{

final JCExitFrame frame =

new JCExitFrame("Five-star Rating"); frame.getContentPane().add(new SimpleIndicatorGaugePanel()); frame.pack();

frame.setVisible(true);

}

}

#### Creating an Indicator Gauge Panel

The JCBaseIndicatorGauge constructor creates a gauge that can be replicated in a specialized panel. You create the panel using the JCIndicatorGaugePanel constructor and specify how many copies of the gauge to place in the panel.

Note: You can also create a panel of indicator gauges using a standard JPanel. In this case, you set the value of each indicator gauge separately. You can find an example in *JCLASS\_HOME/examples/gauge/indicator/Toggle.java*.

##### Constructing a JCBaseIndicatorGauge Object

The following code snippet creates a gauge that can be replicated in the indicator gauge panel. As with the standalone gauge component, the constructor creates the icon that appears on the gauge as an instance of JCIndicatorIcon. By default, the shape of the icon is a rectangle.

// Create a gauge with default rectangular icon shape JCBaseIndicatorGauge myGauge = new JCBaseIndicatorGauge();

Alternatively, you can specify the icon shape in the constructor:

// Specify a shape for the icon when creating a gauge JCBaseIndicatorGauge myGauge =

new JCBaseIndicatorGauge(JCIndicatorIcon.SHAPE\_TRIANGLE);

For more information, see [Section 2.9.1, “Setting the Icon Shape,” on page 58](#_bookmark110).

##### Constructing a JCIndicatorGaugePanel Object

When you create the panel, you specify the number of gauges and the JCBaseIndicatorGauge object that you created. The panel replicates the base gauge the specified number of times. The number parameter is required. If you omit the JCBaseIndicatorGauge object, a default one is created and used.

// Create a panel with eight instances of the gauge myGauge JCIndicatorGaugePanel myPanel = new JCIndicatorGaugePanel(8, myGauge);

##### Assigning a Value to Your Indicator Gauge Panel

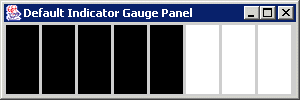
To set the current value of the panel, set the value property to an Integer object. The value must be greater than or equal to the base value, which is zero by default.

Note: For binary and decimal representations, the base value is always zero even if the baseValue property has been set to a different value. Note that this means that negative integers are not supported for these value types. For more information, see [Section 2.8.4,](#_bookmark100) [“Setting the Base Value for a Panel,” on page 56](#_bookmark100).

For example:

// Create the gauge and panel and assign a value to the panel JCBaseIndicatorGauge myGauge = new JCBaseIndicatorGauge(); JCIndicatorGaugePanel myPanel = new JCIndicatorGaugePanel(8, myGauge); myPanel.setValue(new Integer(7));

The preceding example creates a default horizontal panel displaying the value 7 as a binary number. The size of the gauge used in the panel is calculated so that the instances of the gauge fill the panel container, leaving a one pixel padding all around. The color of the gauges is controlled by two default ranges. The first range represents zero and is assigned the color black, while the second range represents the value one (or greater) and uses white.



*Figure 21 Default indicator gauge panel showing the value 7 in binary format*

##### Next Steps

After you complete the necessary tasks of constructing the panel and assigning a value to it, you can decide how you want the panel of gauges to behave and look.

To customize how the panel behaves, you can choose from the following tasks:

* [Section 2.8.1, “Setting the Value Type and Defining Ranges for a Panel,” on page 50](#_bookmark91)
* [Section 2.8.2, “Setting the Panel Direction,” on page 55](#_bookmark96)
* [Section 2.8.3, “Setting the Panel Orientation,” on page 56](#_bookmark99)
* [Section 2.8.4, “Setting the Base Value for a Panel,” on page 56](#_bookmark100)
* [Section 2.8.5, “Changing the Gauge on the Panel,” on page 57](#_bookmark104)
* [Section 2.8.6, “Changing the Number of Gauges Displayed,” on page 57](#_bookmark106)

To customize the appearance of the panel and its gauges, you can choose from the following tasks:

* [Section 2.9.1, “Setting the Icon Shape,” on page 58](#_bookmark110)
* [Section 2.9.2, “Setting the Gauge Size,” on page 59](#_bookmark112)
* [Section 2.9.3, “Setting the Padding,” on page 59](#_bookmark114)
* [Section 2.9.4, “Setting a Background Color,” on page 60](#_bookmark116)
* [Section 2.9.5, “Specifying and Scaling an Image,” on page 62](#_bookmark118)
* [Section 2.9.6, “Setting the Font and Color for Text,” on page 63](#_bookmark120)
* [Section 2.9.7, “Making Gauge Icons Blink,” on page 63](#_bookmark122)

#### Customizing the Behavior of Your Panel

This section describes how to change the value type of the panel, define ranges, set the orientation of the panel, reverse the direction in which values increase across the panel, set the base value, change the gauge used, and specify the number of gauges on the panel. To learn how to change the appearance of the panel and gauge, see [Section 2.9,](#_bookmark108) [“Customizing the Appearance of Gauges and Panels,” on page 58](#_bookmark108).

##### Setting the Value Type and Defining Ranges for a Panel

Your panel of gauges can display the current value in binary form, as a decimal value, or as magnitude in a scale. To set the value type, set the valueType property to one of the predefined constants. The valid constants are:

* JCIndicatorGaugePanel.VALUE\_MAGNITUDE\_UNICHROME
* JCIndicatorGaugePanel.VALUE\_MAGNITUDE\_MULTICHROME
* JCIndicatorGaugePanel.VALUE\_DECIMAL
* JCIndicatorGaugePanel.VALUE\_BINARY *(default)*

Ranges are applied differently, depending on the value type selected. The value types and their use of ranges are described in the subsections that follow.

For background information on ranges, see [Section 2.5.1, “Defining Ranges for a](#_bookmark70) [Standalone Gauge,” on page 42](#_bookmark70) and the *API* documentation.

* + - 1. Magnitude

The VALUE\_MAGNITUDE\_UNICHROME and VALUE\_MAGNITUDE\_MULTICHROME value types show the current value within the context of a scale. The scale is created by assigning the first gauge the value of baseValue+1 and incrementing the value by one for each consecutive gauge. For example, if a panel has ten gauges and the base value is zero, the gauges are assigned values consecutively from one through ten. The current value is represented in the scale by turning on the gauges with values equal to or less than the current value. A gauge is turned on when its display is different than the one used for null and zero values.

Note: You must define ranges and colors to show meaningful information in this type of indicator gauge panel. For background information on ranges, see [Section 2.5.1,](#_bookmark70) [“Defining Ranges for a Standalone Gauge,” on page 42](#_bookmark70).

For example, the following table outlines the ranges and attributes for the sample indicator gauge panel ([Figure 20](#_bookmark78)):

|  |  |  |
| --- | --- | --- |
| **Range** | **Display Object** | **Tooltip** |
| 0 | *star\_none.gif* | *none* |
| 1 | *star\_cyan.gif* | Poor |
| 2 | *star\_blue.gif* | Satisfactory |
| 3 | *star\_bronze.gif* | Good |
| 4 | *star\_silver.gif* | Great |
| 5 | *star\_gold.gif* | Exceptional |

The following code snippet implements the unichrome version of magnitude and the ranges and attributes used in the preceding example. Note that no text is specified.

// Set the value type myGauge.setValueType(

JCIndicatorGaugePanel.VALUE\_MAGNITUDE\_UNICHROME);

// Configure the range attributes for the five-star rating system myPanel.setRangeValues(new Integer[]{

new Integer(0), new Integer(1), new Integer(2), new Integer(3), new Integer(4), new Integer(5)});

myPanel.setRangeDisplays(new URL[]{

this.getClass().getResource("star\_none.gif"), this.getClass().getResource("star\_cyan.gif"), this.getClass().getResource("star\_blue.gif"), this.getClass().getResource("star\_bronze.gif"), this.getClass().getResource("star\_silver.gif"), this.getClass().getResource("star\_gold.gif")});

myPanel.setRangeToolTips(new String[]{

"",

"Poor",

"Okay",

"Good",

"Great", "Exceptional"});

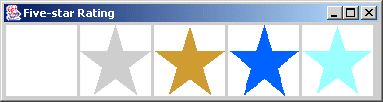
*Figure 22 Unichrome indicator panel*

Difference Between Unichrome and Multichrome

Ranges are defined in the same way for both magnitude value types. However, the range attributes are applied to the gauges differently, depending on the value type. For VALUE\_MAGNITUDE\_UNICHROME, the gauges that are turned on display the range attributes associated with the *current value*. For VALUE\_MAGNITUDE\_MULTICHROME, the gauges display the range attributes associated with the individual *gauge value*s.

The following images show the sample indicator gauge panel in unichrome and in multichrome when the current value is 4.





*Figure 23 Value of 4 in UNICHROME (top) and MULTICHROME (bottom)*

In both images, gauges one through four are turned on. The VALUE\_MAGNITUDE\_UNICHROME version displays all silver stars, which is the star associated with the range value 4. In the VALUE\_MAGNITUDE\_MULTICHROME version, the cyan, blue, bronze, and silver stars correspond to the range values 1, 2, 3, and 4 respectively. The inactive gauges are white, because an empty image with a white background is assigned to the zero range.

* + - 1. Decimal

When the value type is VALUE\_DECIMAL, you can display the current value as a whole number in decimal format. Negative values cannot be displayed using this value type. If a negative value is set, it is ignored and the last valid value continues to be displayed.

Each gauge in the panel represents a place value holder, that is, ones, tens, hundreds, and so on, from right to left. Each number in the current value is assigned to the gauge holding its place value.

By default, the gauges are black with no numerals. To show meaningful information, you must define ranges, one for each number from zero to nine. You can then associate the numerals with the ranges and, if you like, colors as well.

For example, the following code creates an indicator gauge panel with the value type VALUE\_DECIMAL. Range values are defined using a convenience array called DECIMAL\_VALUES that sets up the values zero through nine. Colors are specified for each range. The current value is set to 204157.

// Create a panel with 10 default gauges of type JCBaseIndicatorGauge myPanel = new JCIndicatorGaugePanel(10);

// Set the value type myPanel.setValueType(JCIndicatorGaugePanel.VALUE\_DECIMAL);

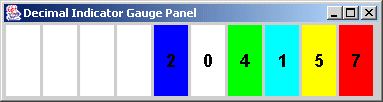
// Set the current value myPanel.setValue(new Integer(204157));

// Assign range colors to range values 0-9 myPanel.setRangeDisplays(new Color[]{

Color.white, Color.cyan, Color.blue, Color.magenta, Color.green, Color.yellow, Color.orange, Color.red, Color.pink, Color.gray});

myPanel.setRangeValues(JCIndicatorGaugePanel.DECIMAL\_VALUES); myPanel.setTextValueDisplayed(true);

// Set the font and color to use for icon text myPanel.setIconFont(new Font("Dialog", Font.BOLD, 18)); myPanel.setIconForeground(Color.black);



*Figure 24 Value displayed as a decimal*

Recall that the default direction is forward, which means right to left, so the first gauge is on the far right. The first gauge is the ones place value holder and displays the 7. The number seven has the color red associated with it, so the gauge is red. The next gauge is the tens place value and holds the value 5; the gauge is yellow because that is the color associated with the number five. The third gauge is hundreds and holds the value 1; the number one is associated with cyan. The remaining numerals are treated similarly. The extra gauges display the color associated with zero (white).

You can specify text or tooltips by setting the rangeText and rangeToolTips properties. Both take a String[] object with an array of text strings. When setting the rangeText property, you need to set the textValueDisplayed property to false.

// Define custom tooltips myPanel.setTextValueDisplayed(false); myPanel.setRangeText(new String[]{

"zero", "one", "two", "three", "four"

"five", "six", "seven", "eight", "nine"}); myPanel.setRangeToolTips(new String[]{

"zero", "one", "two", "three", "four"

"five", "six", "seven", "eight", "nine"});

For background information on ranges, see [Section 2.5.1, “Defining Ranges for a](#_bookmark70) [Standalone Gauge,” on page 42](#_bookmark70).

Changing the Number of Gauges

If you like, you can eliminate the leading, empty gauges and display the example number 204157 in six gauges. You need to set up the panel to change the number of gauges dynamically, allowing you to show only the number of gauges necessary to display the value. For more information, see [Section 2.8.6, “Changing the Number of Gauges](#_bookmark106) [Displayed,” on page 57](#_bookmark106).

* + - 1. Binary

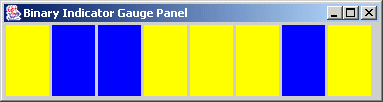
When the value type is set to VALUE\_BINARY (the default value type), the current value is displayed as a binary number. Negative values cannot be displayed using this value type. If a negative value is set, it is ignored and the last valid value continues to be displayed.

Each gauge in the panel can represent a one (1) or a zero (0). The zeros are represented as black gauges and the ones are white gauges. If you like, you can change these colors and add text.

For example, the following code displays the current value in binary format and changes the default colors. Conveniently, gauges are created with two default ranges that have the values 0 and 1. In this example, the zero gauges are set to blue, while the ones gauges are set to yellow.

// Display the current value as a binary number myPanel.setValueType(JCIndicatorGaugePanel.VALUE\_BINARY);

// Change the color of the gauges

myPanel.setRangeDisplays(new Color[]{Color.blue, Color.yellow}); myPanel.setValue(new Integer(157));

*Figure 25 Value displayed as binary*

You can specify different text or tooltips by setting the rangeText and rangeToolTips

properties. Both properties take a String[] object with an array of text strings.

Note: setTextValueDisplayed() takes precedence over setRangeText(). When using

rangeText, set the textValueDisplayed property to false.

// Specify text and tooltips for binary format myPanel.setTextValueDisplayed(false); myPanel.setRangeText(new String[]{"zero", "one"}); myPanel.setRangeToolTips(new String[]{"off", "on"});

For background information on ranges, see [Section 2.5.1, “Defining Ranges for a](#_bookmark70) [Standalone Gauge,” on page 42](#_bookmark70). If you want to change the font used for text, see [Section](#_bookmark120) [2.9.6, “Setting the Font and Color for Text,” on page 63](#_bookmark120).

##### Setting the Panel Direction

By default, the right-most gauge in a horizontal panel represents the least significant range value (excluding zero). For example, in decimal format, the right-most gauge represents the ones place. The range values of the gauges increase from right to left, with the next gauges representing tens, hundreds, and so on. If the panel is vertical, the values increase from bottom to top. You can change the direction in which the range values increase.

To set the direction, set the direction property to one of the following constants: JCIndicatorGaugePanel.DIRECTION\_FORWARD (*default*) or JCIndicatorGaugePanel.DIRECTION\_BACKWARD. In a horizontal panel, DIRECTION\_FORWARD causes values to increase from right to left; for a vertical panel, values increase from bottom to top. DIRECTION\_BACKWARD reverses the direction for each of the orientations.





*Figure 26 FORWARD (top) and BACKWARD (bottom) directions*

For example:

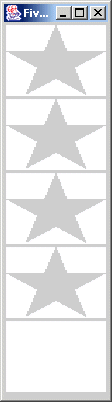
// Increase values from left to right in a horizontal panel

// (or from top to bottom in a vertical panel) myPanel.setDirection(JCIndicatorGaugePanel.DIRECTION\_BACKWARD);

Caution: If you reverse the direction of a panel that is meant to show a binary or decimal value type (where order is important), the result is likely to be misleading. For example, the current value 000157 displayed as a decimal on a reversed panel would show 751000!

##### Setting the Panel Orientation

You can specify whether the gauges are displayed horizontally or vertically within the panel. To set the orientation, set the orientation property to one of the following constants: JCIndicatorGaugePanel.ORIENTATION\_HORIZONTAL (default) or JCIndicatorGaugePanel.ORIENTATION\_VERTICAL.



*Figure 27 HORIZONTAL (left) and VERTICAL (right) orientations*

For example:

// Make the panel vertical myPanel.setOrientation(JCIndicatorGaugePanel.ORIENTATION\_VERTICAL);

If the orientation is changed, you may need to resize the panel’s container to see the entire array of gauges.

##### Setting the Base Value for a Panel

The base value is the lowest value that the panel can display. When the current value is equal to the base value, it is equivalent to turning off all the gauges. By default, the baseValue property is set to null, which is interpreted as zero.

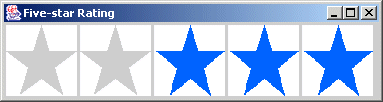
When the value type of the gauge is VALUE\_MAGNITUDE\_UNICHROME or VALUE\_MAGNITUDE\_MULTICHROME, you can specify a different base value. The first gauge on the panel takes the value of baseValue+1. So, if you set a base value of 2, the first gauge has a value of 3. If the current value falls below the base value, it cannot be displayed; the base value is used and all the gauges are turned off.

To change the base value, set the baseValue property to an Object of the same type as the value property, usually an Integer object. For more information, see [Section 2.7.3,](#_bookmark86) [“Assigning a Value to Your Indicator Gauge Panel,” on page 49](#_bookmark86).

For example:

// Set a base value of 2 for the VALUE\_MAGNITUDE value types myPanel.setBaseValue(new Integer(2));

To see the results of changing the base value, consider the sample indicator gauge using a value of 4 and a VALUE\_MAGNITUDE\_UNICHROME value type. The direction of the indicator gauge panel is backwards, which means that the gauges increase from left to right. With a base value of 2, the left-most gauge represents 3 and the next gauge is 4. These two gauges are silver, which is the color associated with the value 4. The remaining gauges display the color associated with the value 2 (the base value), that is, blue.



*Figure 28 Sample indicator gauge panel with a base value of 2 and a current value of 4*

As you can see, the end result of changing the base value for this application is accurate but non-intuitive. Take care when adjusting the base value in your own applications that the result is meaningful to your end-user.

When the value type of the panel is set to VALUE\_BINARY or VALUE\_DECIMAL, the

baseValue property is ignored. It is always assumed to be zero.

##### Changing the Gauge on the Panel

When you created the indicator gauge panel, you had the opportunity to specify the gauge that you wanted replicated on the panel. If you chose not to create and specify a gauge at that time, a default gauge was created and used instead.

You can choose to use a different gauge after the panel is constructed. To change the indicator gauge displayed, set the gauge property to a gauge of type JCBaseIndicatorGauge.

For example:

// Change the gauge used myPanel.setGauge(myOtherGauge);

##### Changing the Number of Gauges Displayed

When you created the indicator gauge panel, you specified how many copies of the gauge to display on the panel. You can choose to change the number of gauges used on the panel after the panel is constructed. For example, you may want to increase the number of gauges for a large value and decrease them to display a small value. To change the number of gauges, set the numGauges property to a positive int.

For example:

// Change the number of gauges on the panel from 3 to 10 myPanel.setNumGauges(10);

To test whether or not you need to update the number of gauges dynamically, you can query for the current number of gauges. For example, the following code (taken from the JClass Dungeon Game demo) increases the number of gauges to match the length of the value that represents character points:

// Get the number of character points

Integer pts = new Integer(character.getPoints());

if (pts.toString().length() > ptsGauge.getNumGauges()){ ptsGauge.setNumGauges(pts.toString().length());

}

ptsGauge.setValue(pts);

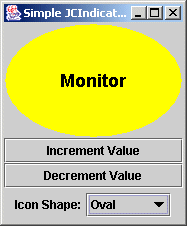
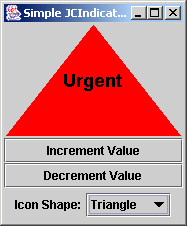
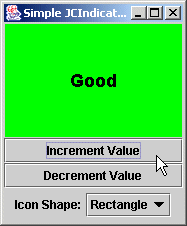
You can use the same approach to reduce the number of gauges.

#### Customizing the Appearance of Gauges and Panels

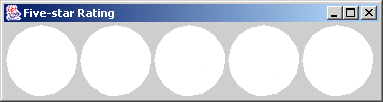
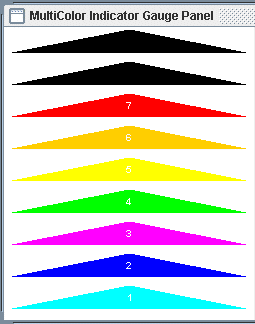
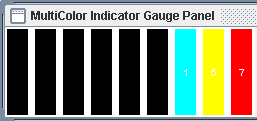
This section contains procedures for changing the shape of the icon and customizing some of the container characteristics, such as gauge size, padding, and background attributes. You can also make the gauge icon blink.

##### Setting the Icon Shape

The iconShape property specifies the shape of the icon. To set the shape, you can either specify it in the gauge constructor or set the iconShape property from JCBaseIndicator. The setIconShape() method enables you to change the shape of the icon when the value of the icon changes. The valid constants are: JCIndicatorIcon.SHAPE\_RECTANGLE (default), JCIndicatorIcon.SHAPE\_TRIANGLE, and JCIndicatorIcon.SHAPE\_OVAL.



*Figure 29 Sample icons shapes in indicator gauges*

*Figure 30 Sample icon shapes in indicator gauge panels*

For example:

// Change the icon shape after the gauge is constructed myGauge.setIconShape(JCIndicatorIcon.SHAPE\_OVAL);

To set the shape in the gauge constructor, see [Section 2.4.1, “Constructing a](#_bookmark64) [JCIndicatorGauge Object,” on page 41](#_bookmark64) and [Section 2.7.1, “Constructing a](#_bookmark82) [JCBaseIndicatorGauge Object,” on page 48](#_bookmark82).

##### Setting the Gauge Size

You can control the initial size of the gauge. If the end-user changes the size of the container, the gauge is resized to fill it. To set the initial size of the gauge, set the preferredSize property from JComponent.

For example:

// Set the initial size (component can be resized by end-user) myGauge.setPreferredSize(new Dimension(100, 100));

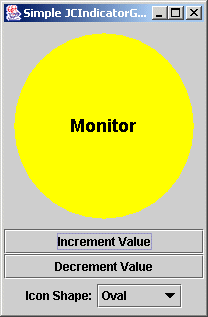
##### Setting the Padding

You can set the *padding* for a gauge and, if you are creating a panel of gauges, for the panel. In this context, padding is the gap around the edges and between the contents of the container. By default, the padding is one pixel wide. If you increase the padding property to be greater than one, more of the container background is exposed around the contents, achieving a border-like effect. For more information, see [Section 2.9.4, “Setting](#_bookmark116) [a Background Color,” on page 60](#_bookmark116).

To set the padding, set the padding property to a positive int that represents the size of the padding in pixels.

Gauge example:

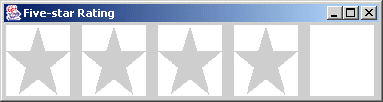
// Set the padding for the gauge myGauge.setPadding(10);



*Figure 31 Gauge padding*

Panel example:

// Set the padding for the panel myPanel.setPadding(10);



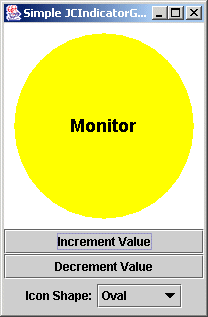
*Figure 32 Panel padding*

##### Setting a Background Color

To set the background color of a gauge or panel, set the background and opaque properties inherited from JComponent. The amount of the background that is exposed is controlled by the padding property. For more information, see [Section 2.9.3, “Setting the](#_bookmark114) [Padding,” on page 59](#_bookmark114).

Gauge example:

// Set a white background for the gauge myGauge.setBackground(Color.white); myGauge.setOpaque(true); myGauge.setPadding(10);



*Figure 33 Gauge background set to white*

Panel example:

// Set the panel background to white myPanel.setBackground(Color.white); myPanel.setOpaque(true); myPanel.setPadding(5);



*Figure 34 Panel background set to white*

The above example eliminates the gray background of the panel, giving a clean appearance for the five-star rating application. However, you can create different results by setting both the gauge and panel backgrounds to different colors. For example, the following image sets the panel background to yellow with a padding of 15 pixels, and the gauge background to red with a padding of 10 pixels. The white area is part of the image used on the icon.



*Figure 35 Gauge background in red (10 pixel padding); panel background in yellow (15 pixel padding)*

If you want to use an image in the background of a gauge or panel, see [Section 2.9.5,](#_bookmark118) [“Specifying and Scaling an Image,” on page 62](#_bookmark118).

##### Specifying and Scaling an Image

You can set an image in the background of a gauge or a panel. By default, the image scales to fill the background area. If an image contains transparent pixels and a background color is specified for the container, the background color shows through the transparent pixels. For more information, see [Section 2.9.4, “Setting a Background Color,” on page](#_bookmark116) [60](#_bookmark116).

Note: If you want to place an image on the icon, you need to associate images with ranges by setting the rangeDisplays property. Do not use the method described in this section. For more information, see [Section 2.5.1, “Defining Ranges for a Standalone](#_bookmark70) [Gauge,” on page 42](#_bookmark70).

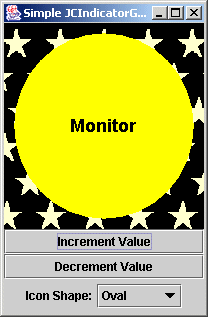
To specify an image in the background of a gauge or panel, set the image property to an

Image or a URL.

By default, the image scales with its container. If you prefer not to have the image scale, set the imageScaled property to false. In this case, the image is centered within the container. If the image is smaller than its container, the background (if drawn) shows around the image. If the image is larger than its container, the image is clipped.

Gauge example:

myGauge.setPadding(10); myGauge.setImage(this.getClass().getResource("stars.gif")); myGauge.setImageScaled(true);



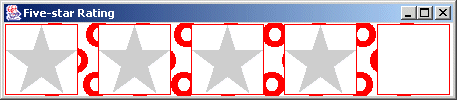
*Figure 36 Indicator gauge with an image in background*

Panel example:

// Create a thin border around the gauge to

// help distinguish the gauge from the panel background myGauge.setPadding(1); myGauge.setBackground(Color.red);

// Specify an image for the panel background and turn off scaling myPanel.setPadding(20); myPanel.setImage(this.getClass().getResource("circles.gif")); myPanel.setImageScaled(false);



*Figure 37 Indicator gauge panel with an image in the panel background*

Images are stored as a PortableImage objects. For more information, see [Section 1.4.2,](#_bookmark37) [“Images and the PortableImage Object,” on page 25](#_bookmark37).

XML Users: If you want to save image information, you also need to set the imageOutputProperties property. For more information, see [Section 11.5, “Preparing to](#_bookmark439) [Save Information About Images to XML,” on page 15](#_bookmark439)7.

##### Setting the Font and Color for Text

If you add text to your gauge, you can customize the font used. To customize the font, set the iconFont and iconForeground properties.

For example:

// Specify font and font color

myGauge.setIconFont(new Font(“Helvetica”, Font.BOLD, 12)); myGauge.setIconForeground(new Color(255, 128, 0));

To specify the actual text, you need to associate Strings with each of the ranges using the rangeText property. For more information, see [Section 2.5.1, “Defining Ranges for a](#_bookmark70) [Standalone Gauge,” on page 42](#_bookmark70) and [Section 2.8.1, “Setting the Value Type and Defining](#_bookmark91) [Ranges for a Panel,” on page 50](#_bookmark91).

##### Making Gauge Icons Blink

By default, a gauge icon does not blink. If you want the icon to blink, you need to specify the interval between blinks in milliseconds and then start the blink timer. For example, you could make the icon blink every ten seconds by setting the blinkInterval to 10000 milliseconds.

Note: Avoid setting a very small blink interval; the gauge icon will blink too fast to be useful and may violate accessibility requirements.

To set a blink interval, set the blinkInterval property to a positive int that represents the interval in milliseconds (a value of zero means that the gauge does not blink). The setBlinkInterval() method creates a timer, which is used to start and stop the blinking. To start the blink timer, use the startBlinking() method. To stop the timer, use the

stopBlinking() method. In a panel of gauges, these methods cause the icons in all the gauges to start and stop blinking as a group.

Gauge example:

// Set the gauge icon to blink every ten seconds myGauge.setBlinkInterval(10000);

// Start the gauge blinking myGauge.startBlinking();

...

// Stop the blinking myGauge.stopBlinking();

Panel example:

// Set the gauge icons to blink simultaneously every ten seconds myPanel.setBlinkInterval(10000);

// Start all the icons blinking myPanel.startBlinking();

...

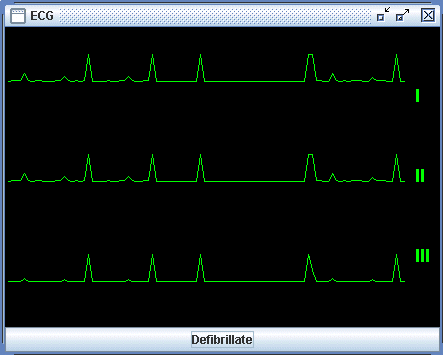
// Stop the blinking myPanel.stopBlinking();

# 3

## Creating Graphs

[*Overview of the Graph Object*](#_bookmark127)■ [*Summary of Properties Used*](#_bookmark129)[*A Sample Graph*](#_bookmark130)■ [*Creating a Graph*](#_bookmark133)■ [*Customizing the Graph*](#_bookmark140)

JCGraph is a light-weight component that you can use to represent fluctuations in a changing value. This graph is not intended as a scientific tool; it does not display scale values nor does it store a history of the values. Rather, the graph is useful as a guidepost to how much or how little a value fluctuates in real time. For example, you could place a graph beside or on top of a standalone indicator gauge. As the gauge value changes during the session, the graph line marks the relative change. This chapter describes how to create and customize a JCGraph component.



*Figure 38 ECG demo from the Example & Demo Gallery*

#### Overview of the Graph Object

The following diagram shows the inheritance for the graph object.

|  |  |  |  |
| --- | --- | --- | --- |
| javax.swing.JComponent | | |  |
|  |  | | |
|  | JCGraph | |
|  | |

*Figure 39 Graph object inheritance*

ComponentListener Scaled

JCGraph extends JComponent and implements the ComponentListener and Scaled

interfaces. The class provides properties and methods to create and define a graph.

#### Summary of Properties Used

The following table summarizes the properties (grouped where appropriate) used in this chapter and provides links to the associated procedures. For a list of all the properties and methods available for graph objects, see the *API documentation*.

|  |  |
| --- | --- |
| **Properties Used** | **Procedures** |
| direction | [Section 3.5.1, “Setting the Direction,” on page 71](#_bookmark142) |
| image imageScaled | [Section 3.5.6, “Specifying and Scaling an Image,” on page 74](#_bookmark152) |
| lineColor lineStroke | [Section 3.5.4, “Setting Line Attributes,” on page 73](#_bookmark148) |
| minValue maxValue | [Section 3.5.2, “Setting the Minimum and Maximum Values,”](#_bookmark144) [on page 72](#_bookmark144) |
| pixelsPerValue | [Section 3.5.3, “Setting the Distance Between Value Points,” on](#_bookmark146) [page 72](#_bookmark146) |
| values addValue addValues | [Section 3.4.2, “Supplying Values to Your Graph,” on page 70](#_bookmark136) |

#### A Sample Graph

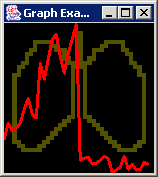
This section contains a sample graph and the code used to create it. Values are added to the graph by setting an initial array and then by adding values during run time. The appearance of the graph is customized by setting properties that control the line and background attributes of the graph.

For general instructions on creating your own graph, see these sections:

* + - [Section 3.4.1, “Constructing a JCGraph Object,” on page 69](#_bookmark134)
    - [Section 3.4.2, “Supplying Values to Your Graph,” on page 70](#_bookmark136)
    - [Section 3.5, “Customizing the Graph,” on page 71](#_bookmark140)

##### Sample Graph

The following figure shows an implementation of a sample graph used in a medical context to track breathing rate. The graph line is red and three pixels wide. An image of lungs is displayed in the background of the graph.



*Figure 40 Sample graph tracking respiration*

##### Sample Code

The following program creates the sample graph. The code is modified from *GraphExample.java* and uses the image and initial values from *MedicalDisplay.java*. To locate examples and demos, see [Example & Demo Gallery on page 14](#_bookmark8).

package examples.gauge.graph;

import com.klg.jclass.gauge.graph.JCGraph; import com.klg.jclass.util.swing.JCExitFrame;

import java.awt.\*;

import java.awt.event.ActionEvent; import java.awt.event.ActionListener; import java.util.Random;

import javax.swing.BorderFactory; import javax.swing.JPanel;

import javax.swing.Timer;

public class GraphExample extends JPanel

{

JCGraph myGraph = null;

/\*\*

\* Creates the GraphExample

\*/

public GraphExample()

{

// Create the graph with some initial values to display

myGraph = new JCGraph(new double[]{15.0, 20.0, 30.0, 25.0, 27.0, 30.0,

38.0, 45.0, 35.0, 33.0, 66.0, 56.0,

68.0, 80.0, 83.0, 70.0, 60.0, 72.0,

80.0, 89.0});

// By default it is very small. Make it bigger. myGraph.setPreferredSize(new Dimension(150, 150));

// Configure some appearance properties myGraph.setBackground(Color.black); myGraph.setOpaque(true);

myGraph.setImage(this.getClass().getResource("hollowlungs.gif")); myGraph.setImageScaled(true);

// Set the line attributes myGraph.setLineColor(Color.red); myGraph.setPixelsPerValue(4);

myGraph.setLineStroke(new BasicStroke(3, BasicStroke.CAP\_ROUND,

BasicStroke.JOIN\_ROUND));

// Configure the panel setLayout(new BorderLayout()); add(myGraph, BorderLayout.CENTER);

// Create a timer so we can add values to the graph, and

// watch it scroll

Timer pulseTimer = new Timer(500, new ActionListener(){ Random random = new Random();

public void actionPerformed(ActionEvent e){

double value = random.nextInt(10) + random.nextDouble(); myGraph.addValue(value);

}

});

pulseTimer.start();

}

//Runs the example

public static void main(String[] args)

{

GraphExample ge = new GraphExample();

JCExitFrame jf = new JCExitFrame(“Graph Example”); jf.setContentPane(ge);

jf.pack(); jf.setVisible(true);

}

}

#### Creating a Graph

This section describes how to create a graph and assign values to it.

##### Constructing a JCGraph Object

The JCGraph constructor creates a graph component that can be displayed within a container. You can use the constructor with no arguments to create a graph with default properties.

// Create a graph with default values JCGraph myGraph = new JCGraph();

There are other JCGraph constructors that can be used to set the direction of the graph, provide an initial set of values to the graph, and set initial minimum and maximum boundaries for the graph. These constructors are described in the following subsections.

* + - 1. Specifying the Direction

By default, the graph travels from left to right. You can change this behavior by specifying the direction parameter in the constructor.

// Specify the direction of the graph JCGraph myGraph = new JCGraph(JCGraph.DOWN);

For a list of valid constants for the direction parameter, see [Section 3.5.1, “Setting the](#_bookmark142) [Direction,” on page 71](#_bookmark142).

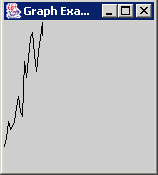
* + - 1. Specifying Initial Values

You can provide an initial array of values in the constructor. The values are specified as an array of doubles. The graph is plotted using these values.

// Specify an array of values myGraph = new JCGraph(new double[]{

15.0, 20.0, 30.0, 25.0, 27.0, 30.0,

38.0, 45.0, 35.0, 33.0, 66.0, 56.0, 68.0,

80.0, 83.0, 70.0, 60.0, 72.0, 80.0, 89.0});

*Figure 41 Graph with default properties and an initial array of values*

If you have a real-time application, you are likely to provide values while the application is running. For more information, see [Section 3.4.2, “Supplying Values to Your Graph,”](#_bookmark136) [on page 70](#_bookmark136).

* + - 1. Specifying Direction, Values, and Boundaries

You can set the direction of the graph, its initial set of values, and its initial minimum and maximum boundaries using this constructor. The direction and values parameters are set as described in the preceding sections. The minimum and maximum boundary values are specified as doubles, where the minimum value must be less than the maximum value.

// Specify a complete constructor

double[] myValues = [-3.0, 5.0, 2.0, 9.0, -10.0];

JCGraph myGraph = new JCGraph(JCGraph.LEFT, myValues, -10.0, 10.0);

During run time, if a value is added to the graph that exceeds a boundary, that value becomes the new boundary value. For more information, see [Section 3.5.2, “Setting the](#_bookmark144) [Minimum and Maximum Values,” on page 72](#_bookmark144).

##### Supplying Values to Your Graph

After you create a graph, you can add values as an array of values or one at a time.

* + - 1. Setting an Array of Values

To set an array of values, set the values property to a double[] object. These values replace any existing graph values.

For example:

// Create a graph with default values JCGraph myGraph = new JCGraph();

// Specify an array of values

myGraph.setValues(new double[] {15.0, 20.0, 30.0, 100.0, 30.0});

* + - 1. Appending a Value

To append one value at a time, pass a double to the addValue() method. If the value is valid, the new value is appended to the graph’s current list of values. For more information on valid values, see [Section 3.4.2.4, “Defining Acceptable Values,” on page](#_bookmark138) [71](#_bookmark138).

For example:

// Add a value myGraph.addValue(75.0);

If the graph area is full, that is, the graph values span the entire display area, the oldest value is discarded so that the new value can be displayed.

* + - 1. Appending an Array of Values

To append an array of values, pass an array of type double[] to the addValues() method. If the values are valid, the new values are appended to the graph’s current list of values. For more information on valid values, see [Section 3.4.2.4, “Defining Acceptable Values,”](#_bookmark138) [on page 71](#_bookmark138).

For example:

// Add an array of value

myGraph.addValues(new double[] {3.0, 7.0, -10.0, -3.0});

If the graph area is full, that is, the graph values span the entire display area, the oldest values are discarded so that the new values can be displayed.

* + - 1. Defining Acceptable Values

If you want, you can define what constitutes a valid value for your graph. When values are added to the graph, the value is checked against your criteria before it is added.

To accomplish this check, JCGraph uses two events to track the state of the value: valueChangingEvent and valueChangedEvent. The valueChanging event is fired whenever the addValue() or addValues() method is called. If the value passes your criteria, the value is added and the valueChangedEvent is fired. If the value does not pass your check, the value is not added to the graph and the valueChangedEvent is not fired.

##### Next Steps

After you complete the necessary tasks of constructing the graph and supplying values to it, you can decide how you want the graph to behave and look.

To customize the graph, you can choose from the following tasks:

* [Section 3.5.1, “Setting the Direction,” on page 71](#_bookmark142)
* [Section 3.5.2, “Setting the Minimum and Maximum Values,” on page 72](#_bookmark144)
* [Section 3.5.4, “Setting Line Attributes,” on page 73](#_bookmark148)
* [Section 3.5.3, “Setting the Distance Between Value Points,” on page 72](#_bookmark146)
* [Section 3.5.5, “Setting a Background Color,” on page 73](#_bookmark150)
* [Section 3.5.6, “Specifying and Scaling an Image,” on page 74](#_bookmark152)

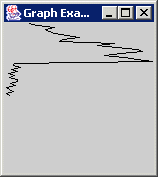
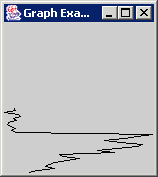
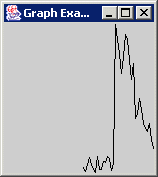
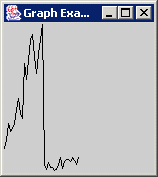
#### Customizing the Graph

You can change the direction in which the graph line travels and the graph’s upper and lower boundary values. The line color and other properties are customizable, as well as the distance between value points in the graph. You can also specify a color and an image in the background.

##### Setting the Direction

By default, the graph travels from left to right. To set a different direction, specify it in the constructor or set the direction property. Valid values are: JCGraph.RIGHT (*default*), JCGraph.LEFT, JCGraph.UP, and JCGraph.DOWN.

For example:

// Make the line travel upwards, from bottom to top myGraph.setDirection(JCGraph.UP);

*Figure 42 Directions (from left to right): RIGHT, LEFT, UP, and DOWN*

##### Setting the Minimum and Maximum Values

The minimum and maximum boundary values for the graph are set dynamically based on the values passed to the graph. If no initial boundaries are set in the constructor, the graph takes the first valid value it receives and uses it as a basis to set both boundary values.

You can choose to set initial boundary values on the graph. To set initial boundary values, specify them in the constructor or set the minValue and maxValue properties. Both properties are of type double. The minimum value must be less than the maximum value.

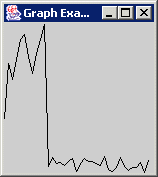
// Set the minimum and maximum values myGraph.setMinValue(1.0); myGraph.setMaxValue(25.0);

During run time, if a value falls outside one of the initial boundary values, the graph resets that boundary so that the value falls within the new bounds.

##### Setting the Distance Between Value Points

By default, values are drawn on the graph every two pixels. For example, a value is drawn at pixel 0, 2, 4, 6, 8, and so on. You can draw the value points closer together or farther apart. To specify the distance from one value point to the next, set the pixelsPerValue property to a positive int.

For example:

// Draw a value every 4 pixels myGraph.setPixelsPerValue(4);

*Figure 43 Drawing value points farther apart*

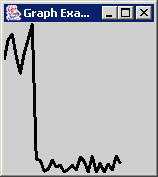
##### Setting Line Attributes

By default, the graph line is 1 pixel wide and is displayed in the color defined for the foreground of the graph. You can change the line properties and the line color.

To change the line properties, such as width, cap style, and join style, you define a new BasicStroke object and then pass it to the setLineStroke() method. You can also create a dashed line by supplying a dash array. For more information, look up java.awt.BasicStroke in the *Java API documentation*.

For example:

// Make the line thicker and specify cap and join myGraph.setLineStroke(new BasicStroke(3, BasicStroke.CAP\_ROUND,

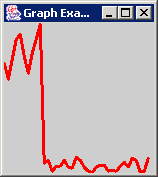
BasicStroke.JOIN\_ROUND));

*Figure 44 Setting line properties*

To change the line color, set the lineColor property to a Color object. For example:

// Make the line red

myGraph.setLineColor(Color.red);



*Figure 45 Changing the line color*

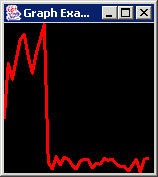
##### Setting a Background Color

To set the background color, set the background and opaque properties inherited from

JComponent. For example:

// Specify a background color and make it opaque

myGraph.setBackground(Color.blue); myGraph.setOpaque(true);



*Figure 46 Changing the background color*

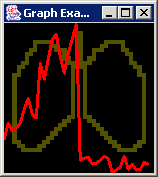
##### Specifying and Scaling an Image

You can set an image in the background of the graph. By default, the image scales to fill its container. If the image has transparent pixels and a background color is specified for the container, the background color will show through the transparent pixels.

To specify an image in the background, set the image property to an Image or a URL.

By default, the image scales with its container. If you prefer not to have the image scale, set the imageScaled property to false. In this case, the image is centered within the container. If the image is smaller than its container, the background (if drawn) shows around the image. If the image is larger than its container, the image is clipped.

For example:

// Specify an image and turn off scaling myGraph.setImage(this.getClass().getResource("hollowlungs.gif")); myGraph.setImageScaled(false);

*Figure 47 Graph with an image in background*

The image, including its imageScaled setting, is stored as a PortableImage object. For more information, see [Section 1.4.2, “Images and the PortableImage Object,” on page 25](#_bookmark37).

XML Users: If you want to save image information, you also need to set the imageOutputProperties property. For more information, see [Section 11.5, “Preparing to](#_bookmark439) [Save Information About Images to XML,” on page 15](#_bookmark439)7.

##### Overlaying a Graph

You can overlay a graph on another component, such as an indicator gauge. For more information, see [Section 1.4.4, “Layering a Graph on Top of a Gauge,” on page 27](#_bookmark41).

*Part II*

*Circular and Linear Gauges*

***4***

## Creating Circular and Linear Gauges

[*Overview of the Gauge Objects*](#_bookmark161)■ [*Creating a Circular Gauge*](#_bookmark163)■ [*Creating a Linear Gauge*](#_bookmark168)[*Assigning Values to a Gauge*](#_bookmark173)■ [*Customizing the Gauge Container*](#_bookmark175)

Circular and linear gauges are derived from the same code base and share many of the same classes and interfaces. Once you learn how to create one type of gauge, it is a simple matter to create the other type. This chapter describes how to create circular and linear gauges. The remaining chapters in Part II of this guide show how to customize a gauge.

#### Overview of the Gauge Objects

The following diagram shows the inheritance for JCGauge objects.

MouseListener

javax.swing.JComponent

MouseMotionListener LegendComponentLayoutUser ToolTipUser

Serializable

Accessible

JCGauge

JCCircularGauge

JCLinearGauge

*Figure 48 Circular and linear gauge object inheritance*

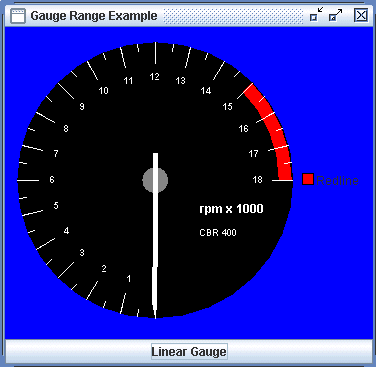
JCGauge is an abstract class that encapsulates the properties and methods required to set up a generic gauge, including adding a header, footer, legend, as well as indicators, needles, ranges, and tick objects. It implements the following interfaces: MouseListener, MouseMotionListener, LegendComponentLayoutUser, ToolTipUser, Serializable, and Accessible.

JCCircularGauge extends JCGauge. The class defines the properties and methods required to create a circular gauge.

JCLinearGauge extends JCGauge. This class defines the properties and methods required to create a linear gauge.

#### Creating a Circular Gauge

When you create a gauge, the gauge uses default values for all its properties. To use the gauge to show values from your application, you set the value on a needle object or an indicator object. This value can be static, dynamically calculated by your application, or chosen by the user. A gauge can have multiple needles and indicators displaying different values. After you set up a basic gauge, you can choose to customize its behavior and appearance. This section describes how to create a circular gauge object.



*Figure 49 Circular gauge from GaugeRangeExample*

##### Constructing a JCCircularGauge Object

The JCCircularGauge constructor can be used without arguments to create a gauge that contains a scale object, as well as some of the objects that can populate the gauge, that is the center, a needle, and a set of tick marks.

// Create a circular gauge

JCCircularGauge myGauge = new JCCircularGauge();

To access the default objects created, use the following syntax:

// Access the default circular scale

JCCircularScale cscale = (JCCircularScale) myGauge.getScale();

// Access the default center

JCCenter center = myGauge.getCenter();

// Access the default tick marks JCCircularTick tick = (JCCircularTick)

myGauge.getScale().getTicks().elementAt(0);

// Access the default needle

JCCircularNeedle needle = ((JCCircularNeedle)

myGauge.getNeedles().firstElement());

Note: To learn how you can customize these objects, see the following sections: [Section 5.4, “Customizing a Circular Scale,” on page 90](#_bookmark194); [Section 10.4, “Customizing](#_bookmark392) [the Center,” on page 14](#_bookmark392)3; “[Customizing Tick Marks](#_bookmark251)” and [Section 6.5, “Customizing](#_bookmark269) [Tick Labels,” on page 11](#_bookmark269)0; and [Section 7.5, “Customizing Indicators and Needles,” on](#_bookmark295) [page 11](#_bookmark295)7.

The JCCircularGauge constructor can also be used with one of two possible parameters:

createScale or gaugeType. These parameters are described in the following sections.

* + - 1. Controlling Object Creation

The no-arguments constructor creates a set of default objects. If you would rather create a gauge with only the default scale object, you can specify the createScale parameter in the JCCircularGauge constructor. The createScale parameter is a boolean that determines whether or not a scale object is created. When in use, no other default objects are constructed.

For example:

// Create a circular gauge with a scale object only JCCircularGauge myGauge = new JCCircularGauge(true);

If the value is false, an empty gauge object is constructed.

* + - 1. Specifying the Type of Circular Gauge

By default, a circular gauge is a complete circle. If you want to use only a portion of a circle, you can specify the gaugeType parameter in the JCCircularGauge constructor. The values for the gaugeType parameters are listed in the table that follows the code examples. When this parameter is in use, no default objects are constructed with the gauge.

For example:

// Create a circular gauge that spans degrees 0 through 180 JCCircularGauge myGauge = new JCCircularGauge

(JCCircularGauge.GaugeType.TOP\_HALF\_CIRCLE);

Alternatively, you can first create a gauge with the default objects, and then set the

gaugeType property. For example:

// Create a circular gauge with default objects

JCCircularGauge myGauge = new JCCircularGauge();

// Set the gauge type myGauge.setGaugeType(JCCircularGauge.GaugeType.TOP\_HALF\_CIRCLE);

The gaugeType property takes an enumeration. The following table shows the start and stop degrees, the enumeration, and a picture of how the gauge might look.

|  |  |  |
| --- | --- | --- |
| **Sweep in Degrees** | **Circular Gauge Type Enumeration** | **Result** |
| 0o–360o | JCCircularGauge.GaugeType.FULL\_CIRCLE |  |
| 0o–180o | JCCircularGauge.GaugeType.TOP\_HALF\_CIRCLE |  |
| 180o–360o | JCCircularGauge.GaugeType.BOTTOM\_HALF\_CIRCLE |  |
| 90o–270o | JCCircularGauge.GaugeType.LEFT\_HALF\_CIRCLE |  |
| 270o–90o | JCCircularGauge.GaugeType.RIGHT\_HALF\_CIRCLE |  |
| 0o–90o | JCCircularGauge.GaugeType.UPPER\_RIGHT\_QUARTER\_CIRCLE |  |
| 90o–180o | JCCircularGauge.GaugeType.UPPER\_LEFT\_QUARTER\_CIRCLE |  |

|  |  |  |
| --- | --- | --- |
| **Sweep in Degrees** | **Circular Gauge Type Enumeration** | **Result** |
| 180o–270o | JCCircularGauge.GaugeType.LOWER\_LEFT\_QUARTER\_CIRCLE |  |
| 270o–360o | JCCircularGauge.GaugeType.LOWER\_RIGHT\_QUARTER\_CIRCLE |  |

You can also create a circular gauge with a different sweep than the predefined types. To create a custom sweep, specify a scale with a start and stop angle for the range you need. For more information, see [Section 5.4.2, “Setting Start and Stop Angles,” on page 91](#_bookmark198).

##### Next Steps

After you create a gauge, you can assign a value to its default needle (if created). You can also add additional needles and indicators, each with their own value. See [Section 4.4,](#_bookmark173) [“Assigning Values to a Gauge,” on page 83](#_bookmark173).

To customize the default components of a circular gauge or to add components, you can choose from the following tasks:

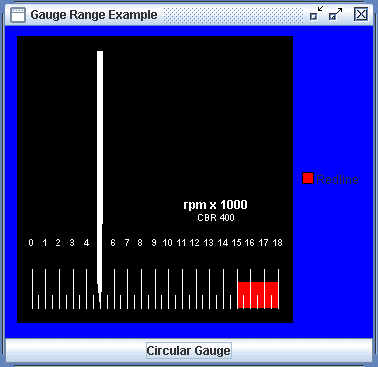
* [Chapter 5, “Defining the Scale”](#_bookmark181)
* [Chapter 6, “Defining Ticks and Tick Labels”](#_bookmark239)
* [Chapter 7, “Defining Indicators and Needles”](#_bookmark279)
* [Chapter 10, “Defining the Center Object in a Circular Gauge”](#_bookmark381)

To display optional components on the gauge, you can choose from the following tasks:

* [Chapter 8, “Defining Text Components”](#_bookmark312)
* [Chapter 9, “Defining Ranges”](#_bookmark355)

#### Creating a Linear Gauge

When you create a gauge, the gauge uses default values for all its properties. To use the gauge to show values from your application, you set the value on a needle object or an indicator object. This value can be static, dynamically calculated by your application, or chosen by the user. A gauge can have multiple needles and indicators displaying different values. After you set up a basic gauge, you can choose to customize its behavior and appearance. This section describes how to create a linear gauge object.



*Figure 50 Linear gauge from GaugeRangeExample*

##### 4.3.1 Constructing a JCLinearGauge Object

The JCLinearGauge constructor can be used without arguments to create a linear gauge that contains a scale object, as well as some of the objects that populate the gauge, that is a needle and a set of tick marks.

// Create a linear gauge

JCLinearGauge myGauge = new JCLinearGauge();

To access the default objects created, use the following syntax:

// Access the default linear scale

JCLinearScale scale = (JCLinearScale) myGauge.getScale();

// Access the default tick marks JCLinearTick tick = (JCLinearTick)

myGauge.getScale().getTicks().elementAt(0);

// Access the default needle JCLinearNeedle needle = ((JCLinearNeedle)

myGauge.getNeedles().firstElement());

Note: To learn how you can customize these objects, see the following sections: [Section 5.6, “Customizing a Linear Scale,” on page 97](#_bookmark220); “[Customizing Tick Marks](#_bookmark251)” and [Section 6.5, “Customizing Tick Labels,” on page 11](#_bookmark269)0; and [Section 7.5,](#_bookmark295) [“Customizing Indicators and Needles,” on page 117](#_bookmark295).

Alternatively, you can provide a createScale parameter to the JCLinearGauge

constructor. See the next section, “[Controlling Object Creation](#_bookmark171).”

4.3.1.1 Controlling Object Creation

The no-arguments constructor creates a set of default objects. If you would rather create a gauge with only the default scale object, you can specify the createScale parameter in the JCLinearGauge constructor. The createScale parameter is a boolean that determines

whether or not a scale object is created. When in use, no other default objects are constructed.

For example:

// Create a linear gauge with a scale object only JCLinearGauge myGauge = new JCLinearGauge(true);

If the value is false, an empty gauge object is constructed.

##### Next Steps

After you create a gauge, you can assign a value to its default needle (if created). You can also add additional needles and indicators, each with their own value. See [Section 4.4,](#_bookmark173) [“Assigning Values to a Gauge,” on page 83](#_bookmark173).

To customize the default components of a linear gauge or to add components, you can choose from the following tasks:

* + - * [Chapter 5, “Defining the Scale”](#_bookmark181)
      * [Chapter 6, “Defining Ticks and Tick Labels”](#_bookmark239)
      * [Chapter 7, “Defining Indicators and Needles”](#_bookmark279)

To display optional components on the gauge, you can choose from the following tasks:

* + - * [Chapter 8, “Defining Text Components”](#_bookmark312)
      * [Chapter 9, “Defining Ranges”](#_bookmark355)

#### Assigning Values to a Gauge

You assign values by setting the value property for each needle object or indicator object associated with your gauge.

Note: This section refers to JCAbstractNeedle, which is the superclass of JCCircularNeedle and JCLinearNeedle. In your own code, you should use JCCircularNeedle with circular gauges and JCLinearNeedle for linear gauges.

Recall that the no-arguments constructor for the gauge objects automatically creates a needle. By default, the initial value of the needle is the minimum value on the scale. To specify the value of the needle, set the value property from JCAbstractNeedle and specify the value as a double.

For example, the following code points the needle at the value 4 on some scale.

// Get the preconstructed needle and set its value

// Substitute JCCircularNeedle or JCLinearNeedle for JCAbstractNeedle. JCAbstractNeedle needle =

((JCAbstractNeedle)myGauge.getNeedles().firstElement()); needle.setValue(4);

If you used one of the other gauge constructors, you first need to create a needle or indicator object. You can set the value as above, or you can include the initial value of the needle or indicator as a parameter in its constructor. For more information, see [Chapter 7,](#_bookmark279) [“Defining Indicators and Needles,”](#_bookmark279) particularly [Section 7.5.1, “Updating the Value,” on](#_bookmark297) [page 11](#_bookmark297)7.

#### Customizing the Gauge Container

You can specify the background color of the container and use an image in the background.

##### Setting a Background Color

To set the background color, set the background and opaque properties inherited from

JComponent. For example:

// Specify a background color and make it opaque

myGauge.setBackground(Color.blue); myGauge.setOpaque(true);

##### Specifying and Scaling an Image

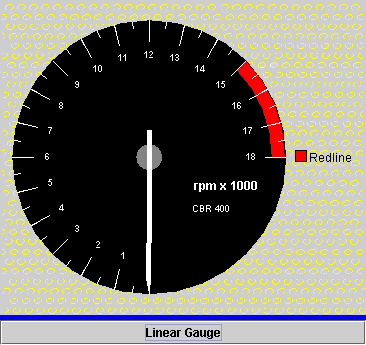
You can set an image in the background of the gauge. By default, the image scales to fill its container. If the image has transparent pixels and a background color is specified for the container, the background color will show through the transparent pixels.

To specify an image in the background, set the image property to an Image or a URL.

By default, the image scales with its container. If you prefer not to have the image scale, set the imageScaled property to false. In this case, the image is centered within the container. If the image is smaller than its container, the background (if drawn) shows around the image. If the image is larger than its container, the image is clipped.

For example:

// Specify an image and turn off scaling myGauge.setImage(this.getClass().getResource("pattern.gif")); myGauge.setImageScaled(false);



*Figure 51 Gauge with an image in background*

The image, including its imageScaled setting, is stored as a PortableImage object. For more information, see [Section 1.4.2, “Images and the PortableImage Object,” on page 25](#_bookmark37).

XML Users: If you want to save images to XML, you also need to set the imageOutputProperties property. For more information, see [Section 11.5, “Preparing to](#_bookmark439) [Save Information About Images to XML,” on page 15](#_bookmark439)7.

# 5

## Defining the Scale

[*Overview of the Scale Objects*](#_bookmark183)■ [*Summary of Properties Used*](#_bookmark185)[*Defining a Circular Scale*](#_bookmark186)■ [*Customizing a Circular Scale*](#_bookmark193)[*Defining a Linear Scale*](#_bookmark212)■ [*Customizing a Linear Scale*](#_bookmark219)[*Enabling User Interaction for Scale Values*](#_bookmark236)

A circular or linear gauge can have one scale object. While the job of the scale object is to hold the information required to construct a scale, displaying the scale is the responsibility of the tick, range, indicator, needle, and text-based objects such as labels. These object are described elsewhere in this guide. The only pieces of visual information in a scale object are the color and image (if any) used for the background of the scale. This chapter introduces the scale objects and describes how to create and customize a circular or linear scale.

#### Overview of the Scale Objects

The following diagram shows the inheritance for the scale objects. Each object is summarized below the diagram.

JCScale

JCAbstractScale

javax.swing.JComponent

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | |
| JCCircularScale |  |  | JCLinearScale |
|  | |

*Figure 52 Scale object inheritance*

JCScale is the interface that represents a graduated scale. If you create a custom gauge object, you can use this interface to define a custom scale.

JCAbstractScale extends JComponent and implements JCScale. It encapsulates the minimum and maximum values for the scale, the direction in which the values on the scale increment, lists of ticks and ranges, and the zoom factor. Its pick() method is used for processing a scale value corresponding to the point at which a mouse click occurred.

JCCircularScale inherits the encapsulated values from JCAbstractScale and provides the implementation details required to create a circular scale from these properties. In addition, it defines a start angle, stop angle, and radius.

JCLinearScale inherits the encapsulated values from JCAbstractScale and provides the implementation details required to create a linear scale from these properties. In addition, it defines the orientation of the scale, that is, whether it is vertical or horizontal.

#### Summary of Properties Used

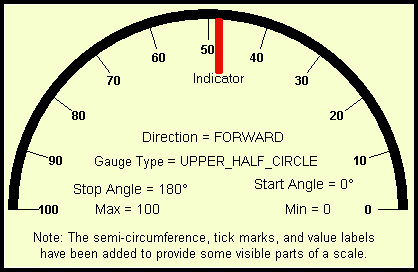
The following table summarizes the properties (grouped where appropriate) used in this chapter and provides links to the associated procedures. For a list of all the properties and methods available for scale objects, see the *API documentation*.

|  |  |  |
| --- | --- | --- |
| **Properties Used** | **Procedures for Circular Scales** | **Procedures for Linear Scales** |
| background | [Section 5.4.6, “Setting the Background](#_bookmark208) [Color,” on page 94](#_bookmark208) | [Section 5.6.7, “Setting Background](#_bookmark234) [Properties,” on page 100](#_bookmark234) |
| border | -- | [Section 5.6.4, “Setting a Border on a](#_bookmark227) [Linear Scale,” on page 98](#_bookmark227) |
| direction | [Section 5.4.3, “Setting the Direction on](#_bookmark201) [a Circular Scale,” on page 92](#_bookmark201) | [Section 5.6.3, “Setting the Direction on](#_bookmark225) [a Linear Scale,” on page 98](#_bookmark225) |
| min max | [Section 5.4.1, “Setting Min and Max](#_bookmark195) [Values on a Circular Scale,” on page 90](#_bookmark195) | [Section 5.6.1, “Setting Min and Max](#_bookmark221) [Values on a Linear Scale,” on page 97](#_bookmark221) |
| orientation | -- | [Section 5.6.2, “Setting the](#_bookmark223) [Orientation,” on page 97](#_bookmark223) |
| paintComplete Background | [Section 5.4.6, “Setting the Background](#_bookmark208) [Color,” on page 94](#_bookmark208) | -- |
| preferredSize | [Section 5.4.5, “Setting the Initial Size of](#_bookmark206) [a Circular Scale,” on page 93](#_bookmark206) | [Section 5.6.6, “Setting the Initial Size](#_bookmark232) [of a Linear Scale,” on page 99](#_bookmark232) |
| radius | [Section 5.4.5, “Setting the Initial Size of](#_bookmark206) [a Circular Scale,” on page 93](#_bookmark206) | -- |
| scaleSize | -- | [Section 5.6.6, “Setting the Initial Size](#_bookmark232) [of a Linear Scale,” on page 99](#_bookmark232) |

|  |  |  |
| --- | --- | --- |
| **Properties Used** | **Procedures for Circular Scales** | **Procedures for Linear Scales** |
| startAngle stopAngle | [Section 5.4.2, “Setting Start and Stop](#_bookmark197) [Angles,” on page 91](#_bookmark197) | -- |
| useZoomFactorForMin useZoomFactorForMax | -- | [Section 5.6.5, “Setting the Zoom](#_bookmark229) [Factor on a Linear Scale,” on page 98](#_bookmark229) |
| zoomFactor | [Section 5.4.4, “Setting the Zoom Factor](#_bookmark203) [on a Circular Scale,” on page 93](#_bookmark203) | [Section 5.6.5, “Setting the Zoom](#_bookmark229) [Factor on a Linear Scale,” on page 98](#_bookmark229) |

#### Defining a Circular Scale

This section describes how to get or create a circular scale.



*Figure 53 A circular scale showing representative values for its properties*

##### Accessing the Preconstructed Circular Scale Object

If you created your gauge using the no-arguments gauge constructor, you already have a scale object associated with the gauge.

The following examples show how to access the scale for a circular gauge.

// Access the default circular scale object

JCCircularScale scale = (JCCircularScale) myGauge.getScale();

The scale object has default values for all properties. You can change the value of any of the properties. For more information, see [Section 5.4, “Customizing a Circular Scale,” on](#_bookmark193) [page 90](#_bookmark193).

##### Constructing a JCCircularScale Object

If your gauge object does not have a scale object, you can create one using the JCCircularScale constructor. The constructors require the gauge parameter, that is, the scale needs to know its parent gauge object. All other properties are assumed to be default values.

// This scale belongs to the circular gauge called ‘myGauge’ JCAbstractScale scale = new JCCircularScale(myGauge);

The following example shows a complete constructor with default values for all the properties. To learn how to customize these properties, see [Section 5.4, “Customizing a](#_bookmark193) [Circular Scale,” on page 90](#_bookmark193).

// Create a scale object and set its properties JCAbstractScale scale = new JCCircularScale(

myGauge, // the parent circular gauge object JCAbstractScale.Direction.FORWARD, // direction (to increase values) 0, // min

100, // max

0, // startAngle (zero degree line)

360. // stopAngle (scale is a complete circle) Color.yellow); // background

Now you need to set the scale on the gauge.

// Set the scale on the gauge called ‘myGauge’ myGauge.setScale(scale);

#### Customizing a Circular Scale

The following procedures describe the scale properties in more detail and explain how to customize a circular scale.

##### Setting Min and Max Values on a Circular Scale

The min and max properties specify the beginning and ending values for the scale. To set the values, you can either specify them in the JCCircularScale constructor or set the min and max properties.

For example:

// Set the minimum and maximum values for the scale scale.setMin(5.0);

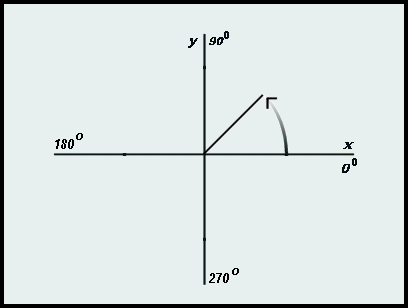
scale.setMax(25.0);

Multi-turn functionality (multiple turns required to move from min to max) is not supported.

##### Setting Start and Stop Angles

The start and stop angles specify the compass positions at which the min and max values are located. The convention for angular measurement in a circular scale defines east as the zero degree line. Angles increase in a counterclockwise direction, so that 90° is north, 180° is west, and 270° is south. [Figure 54](#_bookmark199) shows these four main compass points.

Caution: The JCCircularScale start angle and stop angle replace the traditional start angle and sweep angle used in Java. For example, a scale that occupies a lower half circle has a start angle of 180° and a stop angle of 360°. Avoid specifying these angles as 180° with a sweep of 180°!



*Figure 54 Diagram of a circular scale’s reference plane*

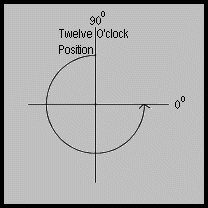
To set the angles, you can either specify them in the JCCircularScale constructor or set the startAngle and stopAngle properties. Both properties are of type double.

For example:

// Set the start and stop angles of the scale scale.setStartAngle(0); scale.setStopAngle(180);

[Figure 53](#_bookmark188) shows the start and stop angles used in the above example. The startAngle is zero degrees with a min value of zero, while the stopAngle is 180° with a max value of100. The values on the scale increment from zero to 100 in the same direction that the compass angle increases, that is, a counterclockwise direction. If you want, you can reverse the location of the minimum and maximum values and the direction in which the values increment on your scale. For more information, see [Section 5.4.3, “Setting the](#_bookmark201) [Direction on a Circular Scale,” on page 92](#_bookmark201).

Note that a start angle may be greater than a stop angle. For example, [Figure 55](#_bookmark200) shows a scale that begins at 90° (startAngle=90) and ends at zero degrees (stopAngle=0).



*Figure 55 A case where the start angle is greater than the stop angle*

##### Setting the Direction on a Circular Scale

By default, the values on the scale increment in the same direction as the angles, that is, counterclockwise. You can reverse the direction to display values incrementing in a clockwise direction.

To set the direction, you can either specify it in the JCCircularScale constructor or set the direction property. The valid constants are:

|  |  |
| --- | --- |
| JCAbstractScale.Direction.FORWARD | Values increase *counterclockwise*, the min value is displayed at the startAngle, and the max value is displayed at the stopAngle. (default) |
| JCAbstractScale.Direction.BACKWARD | Values increase *clockwise*, the min value is displayed at the stopAngle, and the max value is displayed at the startAngle. |

For example, to increment values in a clockwise direction, you could use the following code:

scale.setDirection(JCAbstractScale.Direction.BACKWARD);

Using [Figure 53](#_bookmark188) as a point of reference, setting the direction to BACKWARD means that the values on the scale would be reversed, with zero displayed at the stopAngle of 180° and 100 displayed at the startAngle of zero degrees, and the values would increment clockwise across the scale.

Note: The constants COUNTERCLOCKWISE and CLOCKWISE can be used in place of FORWARD and BACKWARD, respectively. However, using FORWARD and BACKWARD enables you to switch between circular and linear gauge types without potential confusion in terminology.

##### Setting the Zoom Factor on a Circular Scale

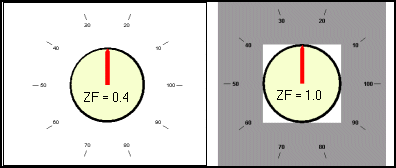
By default, the scale is drawn so that it fills the gauge area. If labels, ticks, or other components need to be placed outside the scale, they may be cropped. To prevent cropping, set the setZoomFactor property.

For example, the following code places tick marks and their labels seemingly well outside a circular scale’s boundary:

// Sets the scale factor scale.setZoomFactor(0.4);

// Places the tick marks and labels tick.setInnerExtent(1.85); tick.setOuterExtent(2.0); tick.setLabelExtent(1.75);

In [Figure 56](#_bookmark205), the left-side image shows a circular gauge with the zoom factor and tick settings used in the above example. Shrinking the scale to 40% of its original size has left room in the gauge container for the tick labels to be displayed. Otherwise, the labels would have been cropped out entirely.



*Figure 56 Placing ticks and their labels well outside the scale*

For comparison, the right-side image shows a similar result obtained without setting a zoom factor (that is, with the default zoom factor of 1.0) and implementing a border instead. The white rectangle is the gauge area and the border is shown in gray. Note that the tick labels would have been cropped if no border were specified.

While the above results appear similar, if you intend to allow end users to resize the gauge, you should use a zoom factor rather than a border. When a gauge is resized, a border remains at its fixed number of pixels. Therefore there is a chance that components may be clipped at some sizes. Setting the zoom factor avoids clipping problems because the components are all resized proportionately.

##### Setting the Initial Size of a Circular Scale

To set the initial size of the scale container, set the preferredSize property from JComponent. The size of the scale relative to its container is controlled by the zoomFactor property. For more information, see [Section 5.4.4, “Setting the Zoom Factor on a Circular](#_bookmark203) [Scale,” on page 93](#_bookmark203).

To get the current size of the scale, use the getRadius() method from JCCircularScale. If the scale has a zoom factor applied, the method returns the radius of the zoomed scale.

##### Setting the Background Color

To set the color of the scale, you can either define it in the constructor or set the

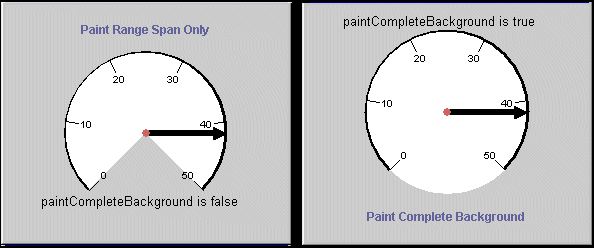
background and opaque properties inherited from JComponent.

// Specify a background color and make it opaque scale.setBackground(Color.blue); scale.setOpaque(true);

By default, only the portion of the scale between its start and stop angles is colored. Any remaining portion retains the color of the gauge. If you want to assign the color to the full scale, set the scale’s paintCompleteBackground property to true.

For example:

(JCCircularScale) myGauge.getScale().setPaintCompleteBackground(true);



*Figure 57 Using paintCompleteBackground to determine how much of the scale is colored*

##### Specifying and Scaling an Image

You can set an image in the background of the scale. By default, the image scales to fill its container. If the image has transparent pixels and a background color is specified for the container, the background color will show through the transparent pixels.

Note: Background images in a circular scale are drawn according to the paintCompleteBackground property. For more information, see the preceding section, “[Setting the Background Color](#_bookmark208).”

To specify an image in the background, set the image property to an Image or a URL. By default, the image scales with its container. If you prefer not to have the image scale, set the imageScaled property to false. In this case, the image is centered within the container. If the image is smaller than its container, the background (if drawn) shows around the image. If the image is larger than its container, the image is clipped.

For example:

// Specify an image and turn off scaling scale.setImage(this.getClass().getResource("stopwatch.gif")); scale.setImageScaled(false);

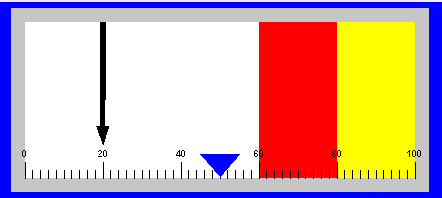
*Figure 58 Scale with background image of a stopwatch*

The image, including its imageScaled setting, is stored as a PortableImage object. For more information, see [Section 1.4.2, “Images and the PortableImage Object,” on page 25](#_bookmark37).

XML Users: If you want to save images to XML, you also need to set the imageOutputProperties property. For more information, see [Section 11.5, “Preparing to](#_bookmark439) [Save Information About Images to XML,” on page 15](#_bookmark439)7.

#### 5.5 Defining a Linear Scale

This section describes how to instantiate a linear scale.



*Figure 59 Linear scale*

[Figure 59](#_bookmark214) shows a simple linear scale in the context of a linear gauge. The scale has a minimum value of zero and a maximum value of 100, and it increments from left to right. In addition to the scale, the gauge in the figure has a triangularly-shaped indicator, a needle, two colored ranges, a set of labeled tick marks, and a collection of labels. Creating these additional components is described in the appropriate sections of this guide. To understand how to accommodate these components relative to the scale without losing

information, see the discussion under [Section 5.6.4, “Setting a Border on a Linear Scale,”](#_bookmark227) [on page 98](#_bookmark227).

##### Accessing the Preconstructed Linear Scale Object

If you created your gauge using the no-arguments gauge constructor, you already have a scale object associated with the gauge.

The following examples show how to access the scale for a linear gauge.

// Access the default linear scale object

JCLinearScale scale = (JCLinearScale) myGauge.getScale();

The scale object has default values for all properties. You can change the value of any of the properties. For more information, see [Section 5.6, “Customizing a Linear Scale,” on](#_bookmark219) [page 97](#_bookmark219).

##### Constructing a JCLinearScale Object

If your gauge object does not have a scale object, you can create one using either the JCLinearScale constructor. The constructor requires the gauge parameter, that is, it needs to know its parent linear gauge object. All other properties are assumed to be default values.

// This scale belongs to the linear gauge called ‘myGauge’ JCAbstractScale scale = new JCLinearScale(myGauge);

The following example shows a complete constructor with default values for all the properties. To learn how to customize these properties, see [Section 5.6, “Customizing a](#_bookmark219) [Linear Scale,” on page 97](#_bookmark219).

JCAbstractScale scale = new JCLinearScale(

myGauge, // the parent linear gauge object JCAbstractScale.Direction.FORWARD, // direction (to increase values) 0, // min

100, // max

JCLinearScale.Orientation.HORIZONTAL, // orientation Color.yellow); // background

#### Customizing a Linear Scale

The following procedures describe the scale properties in more detail and tell you how to customize a linear scale.

##### Setting Min and Max Values on a Linear Scale

The min and max properties specify the beginning and ending values for the scale. To set the values, you can either specify them in the JCLinearScale constructor or set the min and max properties. Both properties are of type double.

For example:

// Specify the beginning and ending values for the scale scale.setMin(5.0);

scale.setMax(25.0);

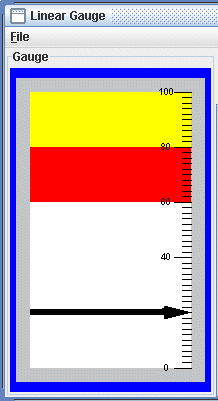
##### Setting the Orientation

A linear scale is oriented either horizontally (default) or vertically. To set the orientation, you can either specify it in the JCLinearScale constructor or set the orientation property. The valid constants are:

|  |  |
| --- | --- |
| JCLinearScale.Orientation.HORIZONTAL | Scale is drawn horizontally. (*Default*) |
| JCLinearScale.Orientation.VERTICAL | Scale is drawn vertically. |

For example:

// Specify that the scale is vertical scale.setOrientation(JCLinearScale.Orientation.VERTICAL);



*Figure 60 Linear scale in VERTICAL orientation*

##### Setting the Direction on a Linear Scale

By default, values increase from left to right on a horizontal scale, and top to bottom on a vertical scale. If you want, you can reverse the direction. To set the direction, you can either specify it in the JCLinearScale constructor or set the direction property. Valid constants are:

|  |  |
| --- | --- |
| JCAbstractScale.Direction.FORWARD | Values increase from left to right. (*Default*) |
| JCAbstractScale.Direction.BACKWARD | Values increase from right to left. |

For example:

// Increment values from left to right scale.setDirection(JCAbstractScale.Direction.FORWARD);

See [Figure 59](#_bookmark214) for a linear scale in the forward direction.

##### Setting a Border on a Linear Scale

While a border is not mandatory, elements within the scale may appear to be clipped by the edges of the container if a border is not applied. To set a border, use the BorderFactory class to define the border and the setBorder() method to apply the border to the scale.

For example:

// Create and use a border around the scale object Border border =

BorderFactory.createLineBorder(new Color(247, 255, 206), 20); scale.setBorder(border);

You can also use createEmptyBorder() to assign different widths to all four sides. For more information, look up the BorderFactory class in the Java API documentation.

[Figure 59](#_bookmark214) (the sample image in the section introduction) shows a linear gauge that uses the scale border defined in the above example. In the figure, the scale has a border 20 pixels wide in the same color as the scale background, which means the border is indistinguishable from the scale itself. Parts of the *Direction*, *Orientation*, *Min*, *Max*, and *Ranges* labels are in the scale’s border area. Note that the visible dark border in the image is part of the gauge, not the scale.

##### Setting the Zoom Factor on a Linear Scale

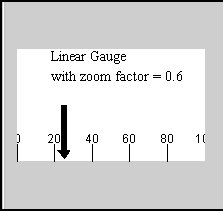
Your design may require that indicators, labels, and other elements be displayed outside of the scale area. You may be able to use a large border to accomplish this task, however, a more flexible approach is to use the zoomFactor property. A zoomFactor less than one applied to a horizontal scale compresses the height of the scale while leaving the width

unchanged. Similarly, a vertical scale is compressed through the width, but not the height. To set the zoom factor, set the zoomFactor property.

For example:

// Compress the scale to 60% of its size scale.setZoomFactor(0.6);

[Figure 61](#_bookmark231) shows a linear gauge with the zoom factor used in the example. The scale is compressed to 60% of its original height, leaving room totalling 40% of the scale height evenly above and below the scale. Needles or ticks with inner extents less than one or with outer extents greater than one will display nicely even when the component is resized.



*Figure 61 Linear gauge with zoom factor of 0.6*

Note: Setting a zoom factor does not remove the necessity of setting a border to avoid the clipping of tick labels at the extremities of the scale. For more information, see [Section](#_bookmark227) [5.6.4, “Setting a Border on a Linear Scale,” on page 98](#_bookmark227).

For even more control, two boolean properties, useZoomFactorForMin and useZoomFactorForMax are available. By default, both of these are true, but if one of these is set to false, the zoom factor will not be applied to the appropriate min and max extent portions of the scale. If both useZoomFactorForMin and useZoomFactorForMax are set to false, the zoom factor is ignored.

The zoom factor is not cumulative. If the zoom factor is later reset, it is applied to the original size of the scale, not the previously zoomed size.

##### Setting the Initial Size of a Linear Scale

To set the initial size of the scale container, set the preferredSize property inherited from JComponent. The size of the scale relative to its container is controlled by the zoomFactor property. For more information, see [Section 5.6.5, “Setting the Zoom Factor on a Linear](#_bookmark229) [Scale,” on page 98](#_bookmark229).

To get the current size of the scale, use the getScaleSize() method from JCLinearScale. If the scale has a zoom factor applied, the method returns the size of the zoomed scale.

##### Setting Background Properties

You set background properties for a linear scale in the same way as circular scales. To set the background color, see [Section 5.4.6, “Setting the Background Color,” on page 94](#_bookmark208). To set an image in the background, see [Section 5.4.7, “Specifying and Scaling an Image,” on](#_bookmark210) [page 94](#_bookmark210).

#### Enabling User Interaction for Scale Values

Given a screen position in pixels (corresponding to an end user’s mouse click), the pick()

method returns the closest scale value.

Events and Listeners in JCGauge

JCGaugePickEvent represents a pick event in JCGauge. A pick event occurs when a JCGaugePickListener is installed on a gauge and the mouse button is pressed over a JCGauge object.

|  |  |
| --- | --- |
| **Method** | **Description** |
| getComponent() | The component associated with this event. |
| getGauge() | Returns the gauge associated with this event. |
| getPoint() | Returns the (*x*, *y*) point of the click. |
| getValue() | The value associated with this event. This is the scale value corresponding to the place where the mouse click occurred. |
| toString() | Returns the point where the mouse was clicked and the associated scale value as a String. |

Interface JCGaugePickListener has one method, pick(). It is called on the object that has installed itself as a listener by invoking gauge.addPickListener(). See *GaugePickExample.java* for an example of the use of a pick listener.

# 6

## Defining Ticks and Tick Labels

[*Overview of Tick Objects*](#_bookmark241)■ [*Summary of Properties Used*](#_bookmark243)[*Defining Ticks*](#_bookmark244)■ [*Customizing Tick Marks*](#_bookmark250)■ [*Customizing Tick Labels*](#_bookmark268)

Ticks are the graduations found on measuring devices. In the context of JClass Gauge, a tick object describes how the ticks on a particular scale are generated, what they look like, how far apart they are, and how they are labelled. This chapter describes the tick objects and discusses how to customize tick marks and labels.

#### Overview of Tick Objects

The following diagram shows the inheritance for tick objects.

JCTick

java.io.Serializable

JCAbstractTick

javax.swing.JComponent

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | |
| JCCircularTick |  |  | JCLinearTick |
|  | |

*Figure 62 Tick object inheritance*

JCTick is the interface that defines a tick object. If you create a custom gauge object, you could implement this interface to define a custom tick object.

JCAbstractTick implements JCTick and extends JComponent. It encapsulates values for tick properties such as start, stop, and increment values, precision and draw properties, extents, and style information. It has methods that automatically calculate reasonable values for start, stop, and increment properties based on the associated scale, as well as the default precision value.

JCCircularTick inherits the encapsulated values from JCAbstractTick and provides the implementation details required to draw ticks on a circular scale.

JCLinearTick inherits the encapsulated values from JCAbstractTick and provides the implementation details required to draw ticks on a linear scale.

#### Summary of Properties Used

The following table summarizes the properties (grouped where appropriate) used in this chapter and provides links to the associated procedures. For a list of all the properties and methods available for tick objects, see the *API documentation*.

|  |  |
| --- | --- |
| **JCAbstractTick Properties** | **Procedures** |
| automatic, startValue, stopValue, incrementValue | [Section 6.4.1, “Setting the Number of Ticks and the](#_bookmark254) [Bounds,” on page 105](#_bookmark254) |
| drawLabels | [Section 6.5.1, “Displaying Tick Labels,” on page 110](#_bookmark270) |
| drawTicks (*or* visible) | [Section 6.4.7, “Displaying Tick Marks,” on page 110](#_bookmark267) |
| innerExtent, outerExtent | [Section 6.4.2, “Setting the Placement and Length of](#_bookmark256) [Tick Marks,” on page 105](#_bookmark256) |
| labelExtent | [Section 6.5.2, “Setting the Tick Label Extent,” on page](#_bookmark271) [110](#_bookmark271) |
| precisionUseDefault, precision | [Section 6.4.1, “Setting the Number of Ticks and the](#_bookmark254) [Bounds,” on page 105](#_bookmark254) |
| scale | [Section 6.3.2, “Constructing JCCircularTick and](#_bookmark248) [JCLinearTick Objects,” on page 103](#_bookmark248) |
| tickColor | [Section 6.4.4, “Setting the Color,” on page 108](#_bookmark262) |
| tickStyle, tickWidth | [Section 6.4.3, “Setting the Tick Style and Width,” on](#_bookmark260) [page 107](#_bookmark260) |

#### Defining Ticks

This section describes how to instantiate a tick for a circular or linear scale. The examples use the tick object customized for circular scales, JCCircularTick. If you are creating a tick for a linear scale, substitute JCLinearTick. The properties for each are the same, only the implementation details differ.

##### Accessing the Preconstructed Tick Object

If you created a gauge using a no-arguments gauge constructor, you already have a tick object associated with the scale. The following example shows how to access the tick.

// Access the default tick object

// For a linear gauge, substitute JCLinearTick for JCCircularTick JCCircularTick tick = (JCCircularTick)

myGauge.getScale().getTicks().elementAt(0);

The tick object has default values for all properties. You can change the value of any of the properties using the methods described under [Section 6.4, “Customizing Tick Marks,”](#_bookmark250) [on page 104](#_bookmark250) and [Section 6.5, “Customizing Tick Labels,” on page 11](#_bookmark268)0.

##### Constructing JCCircularTick and JCLinearTick Objects

You can use a tick constructor when there is no preconstructed tick object or when you want to add additional tick objects. For example, using two tick objects means that you can have major and minor graduations on your scale, as shown in the gauges displayed on the right in [Figure 64](#_bookmark253). To create tick objects, use the constructor appropriate for the type of gauge that you are creating and then add the tick to the scale.

The following example shows a complete tick constructor with the values specified for all properties. To learn how to customize these properties, see [Section 6.4, “Customizing](#_bookmark250) [Tick Marks,” on page 104](#_bookmark250) and [Section 6.5, “Customizing Tick Labels,” on page 11](#_bookmark268)0.

// Get the scale for the gauge "myGauge" JCAbstractScale scale = myGauge.getScale();

// Create a tick object and associate it with the scale

// For a linear scale, substitute JCLinearTick for JCCircularTick JCAbstractTick tick = new JCCircularTick(

scale, // the associated scale

true, // automatic (tick generation)

0, // startValue (taken from scale min value) 100, // stopValue (taken from scale max value)

10, // incrementValue (ignored unless automatic = false) true, // precisionUseDefault

0, // precision (ignored unless precisionUseDefault=false) 2, // tickWidth

true, // drawLabels

true, // drawTicks (also referred to as “visible”) 0.8, // labelExtent

0.85, // innerExtent

1.0, // outerExtent

Color.black, // tickColor (foreground)

JCTickStyle.LINE, // tickStyle new Font("Helvetica", Font.BOLD, 18), // font Color.black); // fontColor

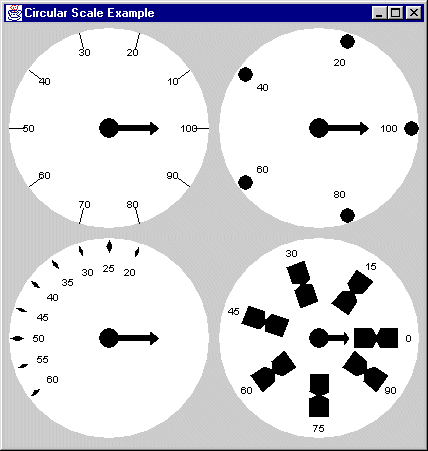
Now add the tick object to the scale specified in the constructor.

// Add the tick to the scale associated with ‘myGauge’ scale.add(tick);

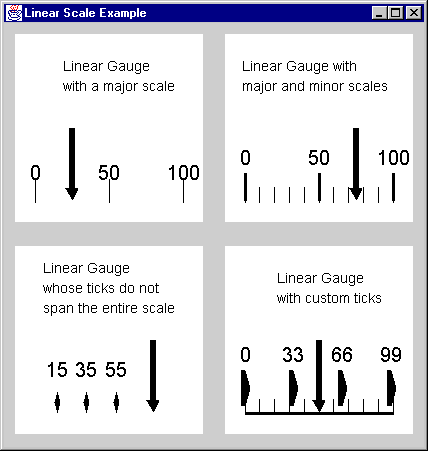
The tick is added to the list of tick objects maintained by the scale object. Recall that components are drawn on the gauge in the reverse order in which they are added. For more information, see [Section 1.5.1, “How Components are Rendered on a Gauge,” on](#_bookmark44) [page 28](#_bookmark44).

#### Customizing Tick Marks

This section describes how to customize the marks used on the scale, including how many and what they look like. The following figures illustrate a variety of tick styles (with labels) displayed on circular and linear gauges.



*Figure 63 Circular gauge tick objects and their associated labels*



*Figure 64 Linear gauge tick objects and their associated labels*

##### Setting the Number of Ticks and the Bounds

Tick marks are normally required at constant increments along a scale. The tick object uses the minimum and maximum values from the associated scale, plus the value of the precision property in the tick object, to determine how many marks to display on the scale and where the marks start and stop. You can choose to specify these values yourself. For example, in [Figure 63](#_bookmark252) and [Figure 64](#_bookmark253), the bottom left gauge shows tick marks limited to a portion of the entire scale.

To set the number of ticks, you need to set the automatic property to false and then specify values (as doubles) for the startValue, stopValue, and incrementValue properties. You can either specify these properties in the tick constructor or set the properties after the tick is created.

For example:

// Tell the tick object that you will define the number of ticks tick.setAutomatic(false);

// Specify the values on the scale where the marks start and stop tick.setStartValue(0.0);

tick.setStopValue(100.0);

// Specify the value between ticks tick.setIncrementValue(10.0);

If you are also displaying labels, you should verify that the default precision value is still suitable for the number of tick marks that you are displaying. The precision property affects the number of labels that can be displayed because it affects the width of the label. For example, a precision that allows for three decimal places to be displayed on the tick labels creates wider labels than when integers are displayed. Wider tick labels will be fewer in number compared to the same scale with narrower tick labels.

To recalculate a reasonable precision value based on your specified startValue, stopValue, and incrementValue, you can use the nicePrecision() method from the JCNumberUtil class, which resides in com.klg.jclass.util. Alternatively, you can set the precision property. For more information, see [Section 6.5.3, “Setting the Precision Used](#_bookmark272) [for Values Displayed in Tick Labels,” on page 11](#_bookmark272)1.

##### Setting the Placement and Length of Tick Marks

The tick object has methods to calculate automatically the placement and length of the tick marks. You can customize the placement or length using the extent properties of the tick object. The values for the extent properties are doubles that represent fractions of the size of the scale. This means that if the gauge is resized, the tick marks will adjust proportionately to suit the size of the scale. For background information on extents, see [Section 1.5.2, “Sizing Components Using Extent Parameters,” on page 29](#_bookmark46).

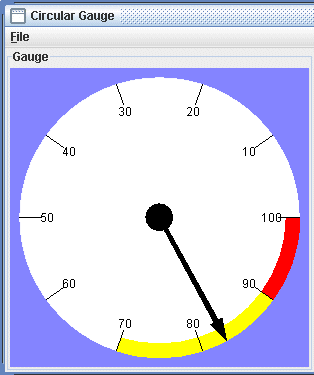
To specify the placement (that is, where to start drawing the marks), you set the innerExtent property. The length is specified indirectly using the outerExtent property, which sets where to stop drawing the marks. You can either specify these extent properties in the tick constructor or set the innerExtent and outerExtent properties.

For example:

// Start drawing the mark at 90% (see below for explanation) tick.setInnerExtent(0.9);

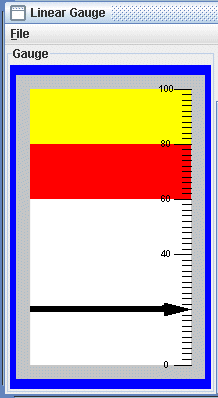
// Stop drawing the mark at 100% (the edge of the scale) tick.setOuterExtent(1.0);

For circular scales, the extent values represent fractions of the radius of the scale. Tick marks are drawn radially outward from the center of the scale to the circumference. Using the above example, a mark is placed at a distance away from the center of the scale that represents 90% of the scale’s radius (inner extent). The mark ends at 100% of the scale radius (outer extent), which is the circumference of the scale. The length of the mark is inferred to be 10% of the radius of the scale (100-90=10).



*Figure 65 Tick marks in a circular scale*

For horizontal linear scales, the extent values represent fractions of the scale’s height. Tick marks are drawn from top to bottom. Using the extents values in the above example, a mark is placed at a distance away from the top edge that represents 90% of the scale’s height (inner extent). The mark ends at 100% of the scale’s height (outer extent), which is the bottom edge of the scale. The length of the mark is inferred to be 10% of the height of the scale (100-90=10). The following image has two sets of tick marks with different inner extents.



*Figure 66 Two sets of tick marks in a linear scale with different inner extents*

On a vertical linear scale, the extent values are fractions of the scale’s width. Tick marks are drawn from left to right.

If the value of innerExtent greater than or equal to the outerExtent, no ticks are drawn. However, the preferred way of hiding tick marks is to set the drawTicks property to false. For more information, see [Section 6.4.7, “Displaying Tick Marks,” on page 11](#_bookmark267)0.

Tick Marks that Extend Outside the Scale

If your tick marks extend outside the scale, that is, at extents less than 0 or greater than 1.0, and you allow the scale to be resized, you may need to increase the dimensions of the scale’s borders to ensure that there is enough space to hold the tick marks, otherwise their outer extents may be clipped. For more information, see [Section 5.6.4, “Setting a Border](#_bookmark228) [on a Linear Scale,” on page 98](#_bookmark228).

A better alternative would be to keep your extents within the scale’s bounds and instead set the zoomFactor property of your scale object to be less than 1.0. For more information, see [Section 5.4.4, “Setting the Zoom Factor on a Circular Scale,” on page 93](#_bookmark204) or [Section](#_bookmark230) [5.6.5, “Setting the Zoom Factor on a Linear Scale,” on page 98](#_bookmark230). See also the *GaugeOutsideExample.java* example in the *JCLASS\_HOME/examples/gauge/* directory.

##### Setting the Tick Style and Width

You can choose from a selection of predefined tick styles or create your own. By default, the tick style is LINE and the width is 2 pixels. You can adjust the width to make the line thinner or thicker. If you choose a style other than LINE, you need to set the width to a value that is greater than the default so that the shape displays well.

To set a tick style, you can either specify it in the tick constructor or set the tickStyle

property. The valid constants are: JCTickStyle.CIRCLE, JCTickStyle.DIAMOND,

JCTickStyle.LINE (default), JCTickStyle.RECTANGLE, JCTickStyle.REVERSE\_TRIANGLE, and JCTickStyle.TRIANGLE.

To set the width, you can either specify it in the tick constructor or set the tickWidth

property, which takes a double. For example:

// Set the tick style and adjust the width of the mark

tick.setTickStyle(JCTickStyle.TRIANGLE); tick.setTickWidth(3);

To define your own tick style, see [Section 6.4.5, “Defining a Custom Tick Style,” on page](#_bookmark264) [108](#_bookmark264).

##### Setting the Color

By default, tick marks are black. To change the color, you can either set it in the tick constructor or set the tickColor property. Specify the value as a Color object.

For example:

// Set the color of the tick tick.setTickColor(Color.white);

##### Defining a Custom Tick Style

You can specify your own tick style in the same way that you create any Rectangle, that is, you use two arrays to define the shape, one for the *x* coordinates and one for the *y* coordinates. The layout algorithm assumes that the center of the tick mark’s bounding rectangle is at (0, 0). To set a custom tick style, you can either subclass the JCTickStyle object, which creates a named style, or use the JCTickStyle constructor to define your (*x*,*y*) coordinate pairs.

Subclassing JCTickStyle

The following code snippet creates a style called NOTCHED\_RECTANGLE by creating a subclass of JCTickStyle and defining the shape using arrays of int values. This custom tick mark is used in the gauge shown in the bottom right of [Figure 63](#_bookmark252).

// Create a custom tick style by extending JCTickStyle import com.klg.jclass.gauge.JCTickStyle;

public class MyTickStyle extends JCTickStyle {

public static final JCTickStyle NOTCHED\_RECTANGLE = new JCTickStyle( new int[] {-10, -2, 0, 2, 10, 10, 2, 0, -2, -10},

new int[] { 3, 3, 1, 3, 3, -3, -3, -1, -2, -2},

10);

}

To use the new tick style, pass the custom tick style to the setTickStyle() method.

// Use the custom tick style MyTickStyle.NOTCHED\_RECTANGLE tick.setTickStyle(MyTickStyle.NOTCHED\_RECTANGLE);

Using the JCTickStyle Constructor

You can use the JCTickStyle constructor to define your own (*x*, *y*) coordinate pairs. First create arrays of ints for the coordinates, and then pass these arrays to the JCTickStyle constructor together with an int specifying the number of points. The gauge in the bottom right of [Figure 64](#_bookmark253) uses this custom tick mark.

// Create a custom tick style using the JCTickStyle constructor int xpoints[] = {-100, 0, 100, 100, 0, -100};

int ypoints[] = { 0, 100, 100, -100, -100, 0};

int numpoints = 6;

tick.setTickStyle(new JCTickStyle(ypoints, xpoints, numpoints));

Note: The style is not named, so it is not as convenient to reuse as the custom style that was created by subclassing JCTickStyle.

##### Connecting Tick Marks

You can connect all the tick marks with a line. The line can connect the marks at their inner extent, outer extent, or run through some point in the marks. To draw a line through the tick marks, you need to create a range. The innerExtent and outerExtent properties of the range need to overlap one of the same extents provided for the tick object or fall between those extents. If you want all marks connected, the startValue and stopValue of the range need to be the same as those specified for the tick object.

For example, the following lines of code get the scale associated with a circular gauge, customize some of the properties of its default tick object, and then create a range that draws an arc that connects all the tick marks at their outer extents.

// Get the scale for the circular gauge ‘myGauge’ JCAbstractScale scale = myGauge.getScale();

// Get the default tick object for this scale

JCCircularTick tick = (JCCircularTick)scale.getTicks().elementAt(0);

// Tick marks extend from 80% of scale radius to its circumference tick.setInnerExtent(0.8);

tick.setOuterExtent(1.0);

// Set the scale values at which the tick marks begin and end tick.setStartValue(0.0);

tick.setStopValue(100.0); tick.setIncrementValue(10.0); tick.setAutomatic(false);

// Create a range to connect the tick marks at their outerExtent JCAbstractRange range = new JCCircularRange(

Color.black, // background color of range scale, // scale (same as tick)

0.98, // innerExtent (breadth is 2%)

1.0, // outerExtent (same as tick)

0, // startValue (same as tick)

100); // stopValue (same as tick)

// Add the range to the scale scale.add(range);

For more information, see [Section 9.4.1, “Setting the Breadth and Location of a Range,”](#_bookmark369) [on page 13](#_bookmark369)6.

##### Displaying Tick Marks

By default, tick marks are displayed on the scale. To set the visibility of the tick marks, you can either specify it in the tick constructor or set the drawTicks property. The value is a boolean.

For example:

// Hide the tick marks tick.setDrawTicks(false);

#### Customizing Tick Labels

By default, a tick label is displayed at each tick mark. These labels are the tick values formatted according to the tick’s precision. You can choose to hide these labels, adjust the precision, or create your own custom labels.

##### Displaying Tick Labels

By default, tick labels are displayed on the scale. You can choose to hide labels on some or all of your tick objects. For example, in a scale with multiple tick objects, you can reduce clutter and improve comprehension by hiding labels for minor graduations.

To set the visibility of the tick labels, you can either specify this property in the tick constructor or set the drawLabels property. The value is a boolean.

For example:

// Hide the tick labels for the ‘minorTick’ tick object minorTick.setDrawLabels(false);

##### Setting the Tick Label Extent

The labelExtent defines the position of the label. It works like a normal extent (see the discussion of innerExtent and outerExtent in [Section 6.4.2, “Setting the Placement and](#_bookmark256) [Length of Tick Marks,” on page 10](#_bookmark256)5). By default, the labelExtent is 0.8, which for circular ticks is 80% between the center and the outside edge of the scale. For linear ticks in a horizontal scale, it is 80% between the top of the scale and the bottom of the scale. For linear ticks in a vertical scale, it is 80% between the left and right boundaries of the scale.

For example, the following code sets the label outside of the scale, 120% away from the center:

// Change the tick label extent tick.setLabelExtent(1.2);

##### Setting the Precision Used for Values Displayed in Tick Labels

You can set the precision with which scale values are displayed on the tick labels. This is a formatting option only, the actual scale values are not affected.

Precision is controlled using the precisionUseDefault and precision properties of the tick object. The precisionUseDefault property is true by default, which means that the value of the precision property is automatically calculated. To change the precision, set the precision parameter in the tick constructor to the desired value and also set the precisionUseDefault parameter to false. To set the precision after the tick object is constructed, set the precision property. This method automatically changes the precisionUseDefault property to false.

The precision property takes an int. The following table describes your options:

|  |  |
| --- | --- |
| **Options** | **Precision Value** |
| To display integers  To display decimal places  To display rounded integers | Set the precision property to zero.  Set the precision property to a *positive* int.  For example, setting a precision of 3 means that scale values are displayed to three decimal places.  Set the precision property to a *negative* int.  For example, setting a precision of -1 means that scale values are rounded to a multiple of ten, that is, a value of -25 is rounded to -20. |

For example:

// Use only integers on the labels tick.setPrecision(0);

If the precisionUseDefault property is subsequently set to true, the precision is recalculated and the result replaces the previously set precision value.

You should be aware that changing the precision may introduce rounding issues that make the scale markings confusing. To avoid rounding issues, set the precision to a value appropriate for the range of values that the tick object represents.

The precision property also controls how many ticks are generated. For more information, see [Section 6.4.1, “Setting the Number of Ticks and the Bounds,” on page](#_bookmark254) [105](#_bookmark254).

##### Adding Text

Tick labels are drawn with the help of the JCLabelGenerator interface. It contains a single method, makeLabel(), which takes three parameters: a JCTick-based object, a scale value, and a GaugeConstraint1. To create a custom tick labeling mechanism, you need to

use the setLabelGenerator() method from JCAbstractTick and implement your own version of JCLabelGenerator.

For example, the following code creates a custom label generator that produces temperature values that include the unit of measurement, such as 20° C. The programmer uses an anonymous inner class to add an implementation of JCLabelGenerator to a tick object. The makeLabel() method is passed a reference to the tick object, the scale value for the tick mark, and a reference to the tick’s RadialConstraint; however, only the value parameter is illustrated in this example. The code adds the text “° C” to each generated value except 0°, where it supplies the word “zero” instead.

// create a label generator to mark

// the temperature values with their units tick.setLabelGenerator(new JCLabelGenerator() {

public JComponent makeLabel(JCTick tick, double value,

GaugeConstraint constraint) { String s = (value != 0) ? String.valueOf((int)value)

: "zero";

JLabel label = new JLabel(s + "\u00B0 C"); label.setToolTipText(s + "\u00B0 C"); return label;

}

});

For an example of a custom label in a user-defined class, see *GaugeSwitchExample.java.*

##### Placing Labels Outside of the Scale Bounds

If your tick marks are close to or exceed the bounds of the scale, the associated labels may be clipped unless you take steps to accommodate them. You can either add or increase the scale border or set the scale’s zoomFactor property. For details, see the discussion under [“Tick Marks that Extend Outside the Scale” on page 107](#_bookmark258) in [Section 6.4.2, “Setting](#_bookmark256) [the Placement and Length of Tick Marks,” on page 105](#_bookmark256).

1. The two subclasses of GaugeConstraint are LinearConstraint and RadialConstraint.

# 7

## Defining Indicators and Needles

[*Overview of the Indicator and Needle Objects*](#_bookmark281)■ [*Summary of Properties Used*](#_bookmark283)[*Defining Indicators*](#_bookmark284)■ [*Defining Needles*](#_bookmark288)■ [*Customizing Indicators and Needles*](#_bookmark294)

[*Adding User Interaction to Needles*](#_bookmark310)

An *indicator* is a static marker that points to a single value on a scale. A *needle* is a type of indicator that allows user interaction. You can have multiple indicators and/or needles on a scale. This chapter describes the indicator and needle objects and explains how to customize them.

Note: In this chapter, wherever the word indicator is used, the information also applies to needles. When the word needle is used, the information is restricted to needle objects.

#### Overview of the Indicator and Needle Objects

The following diagram shows the inheritance for indicator and needle objects.

JCIndicator

JCAbstractIndicator

JCNeedle

JCAbstractNeedle

JCLinearNeedle

JCCircularNeedle

javax.swing.JComponent

|  |  |
| --- | --- |
|  | |
| JCCircularIndicator |  |
|  |
|  | |
| JCLinearIndicator |  |
|  |

*Figure 67 Indicator and needle object inheritance*

Indicator Objects

JCIndicator is the interface that defines an indicator object. If you create a custom gauge object, you could implement this interface to define a custom indicator.

JCAbstractIndicator implements JCIndicator and extends JComponent. It encapsulates values for indicator properties such as the value it points to on the scale, style, width, and color of the indicator, as well as extents.

JCCircularIndicator inherits the values from JCAbstractIndicator and provides the implementation details required to draw indicators on a circular scale.

JCLinearIndicator inherits the values from JCAbstractIndicator and provides the implementation details required to draw indicators on a linear scale.

Needle Objects

JCNeedle extends the JCIndicator interface to define a needle object. It adds length, width, and style parameters for needles, and, more importantly, user interaction parameters. If you create a custom gauge object, you could implement this interface to define a custom needle.

JCAbstractNeedle extends JCAbstractIndicator and implements JCNeedle. It inherits the values from JCAbstractIndicator and encapsulates additional values for needle properties such needle-specific length, width, and style, as well as user interaction properties. It also provides methods for handling user interaction.

JCCircularNeedle inherits the values from JCAbstractNeedle and provides the implementation details required to draw needles on a circular scale.

JCLinearNeedle inherits the values from JCAbstractNeedle and provides the implementation details required to draw needles on a linear scale.

#### Summary of Properties Used

The following table summarizes the properties (grouped where appropriate) used in this chapter and provides links to the associated procedures. For a list of all the properties and methods available for indicator and needle objects, see the *API documentation*.

|  |  |
| --- | --- |
| **Properties** | **Procedures** |
| color  indicatorStyle indicatorWidth | [Section 7.5.7, “Setting the Color,” on page 118](#_bookmark305)  [Section 7.5.2, “Setting the Style,” on page 117](#_bookmark298) [Section 7.5.3, “Setting the Width,” on page 118](#_bookmark300) |

|  |  |
| --- | --- |
| **Properties** | **Procedures** |
| innerExtent outerExtent | [Section 7.5.4, “Setting the Size of an Indicator,” on page 118](#_bookmark301) [Section 7.5.5, “Setting the Size of a Needle,” on page 118](#_bookmark302) |
| reversed | [Section 7.5.6, “Reversing an Indicator or Needle,” on page 118](#_bookmark304) |
| value | [Section 7.5.1, “Updating the Value,” on page 117](#_bookmark296) |
| visible | [Section 7.5.8, “Hiding an Indicator or Needle,” on page 119](#_bookmark307) |

#### Defining Indicators

You can add one or more indicators to your gauge.

##### 7.3.1 Constructing JCCircularIndicator or JCLinearIndicator Objects

You can an indicator using the JCCircularIndicator or JCLinearIndicator constructor; use the one that matches the type of gauge you are creating. The constructor requires the scale parameter, that is, it needs to know the scale to which it belongs. All other properties are assumed to be default values.

// Get the scale for the gauge "myGauge" JCAbstractScale scale = myGauge.getScale();

// Create an indicator object and associate it with the scale. For a

// linear gauge, substitute JCLinearIndicator for JCCircularIndicator. JCAbstractIndicator indicator = new JCCircularIndicator(scale);

The following example shows a complete indicator constructor with the default values for all properties. As you may recall, the properties are encapsulated in JCAbstractIndicator, so values are the same for circular and linear indicator objects.

// Create an indicator object and associate it with the scale

// For a linear gauge, substitute JCLinearIndicator

// for JCCircularIndicator

JCAbstractIndicator indicator = new JCCircularIndicator( Color.black, // color (foreground)

15, // indicatorWidth

scale, // associatedScale

true, // visible

0.0 // innerExtent

1.0 // outerExtent

JCIndicatorStyle.RECTANGLE, // indicatorStyle

0); // value (default is min value of scale)

When you create a new indicator object, you need to add it to your gauge.

// Add the indicator to the gauge myGauge.addIndicator(indicator);

The gauge manages its list of indicators by keeping them in a Vector.

#### Defining Needles

You can add one or more needles to your gauge. Recall that a needle object is created by default when you use one of the no-arguments gauge constructors.

##### Accessing the Preconstructed Needle Object

If you created your gauge using a no-arguments gauge constructor, you already have a needle object associated with the gauge. The following example shows how to access it.

// Access the default needle object JCAbstractNeedle needle =

((JCAbstractNeedle)myGauge.getNeedles().firstElement());

The needle object has default values for all properties. For more information, see [Section](#_bookmark294)

[7.5, “Customizing Indicators and Needles,” on page 117](#_bookmark294).

##### Constructing JCCircularNeedle or JCLinearNeedle Objects

If your gauge does not have a needle object, or if you want additional needles, you can create needles using the JCCircularNeedle or JCLinearNeedle constructor; use the one that matches the type of gauge you are creating. The constructor requires the scale parameter, that is, it needs to know the scale to which it belongs. All other properties are assumed to be default values.

// Get the scale for the gauge "myGauge" JCAbstractScale scale = myGauge.getScale();

// Create a needle object and associate it with the scale

// For a linear gauge, substitute JCLinearNeedle for JCCircularNeedle JCAbstractNeedle needle = new JCCircularNeedle(scale);

The following example shows a complete needle constructor with the default values for all properties. As you may recall, the properties are encapsulated in JCAbstractNeedle, so the values are the same for circular and linear needle objects.

// Create a needle object and associate it with the scale

// For a linear gauge, substitute JCLinearNeedle for JCCircularNeedle JCAbstractNeedle needle = new JCCircularNeedle(

Color.black, // color (foreground)

10, // needleWidth

scale, // scale

JCAbstractNeedle.InteractionType.NONE, // interactionType true, // visible

0.0, // innerExtent

1.0, // outerExtent

JCIndicatorStyle.ARROW, // needleStyle

0); // value (default is min value of scale)

When you create a new needle object, you need to add it to your gauge.

// Add the needle to the gauge at position zero myGauge.addNeedle(needle, 0);

Specifying the position ensures that the needle is on top as the children are drawn in reverse order. The gauge manages its list of needles by keeping them in a Vector.

#### Customizing Indicators and Needles

You can update the value of an indicator or needle. You can also customize the style, color, length, width, and direction.

##### Updating the Value

The value property specifies the value on the scale to which an indicator or needle points. By default, the initial value is the minimum value on the scale. Recall that you can specify the initial value of an indicator or needle in its constructor. To update the value after the object is constructed, set the value property inherited from JCAbstractIndicator, which takes a double.

For example, the following code changes the value of an existing indicator to 10.

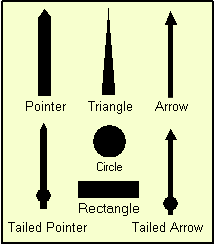
// Change the value that the indicator points to indicator.setValue(10);

Needle Values and Change Listeners

When a needle value changes, either programmatically or by user interaction, the needle notifies any change listeners that have been registered with the needle. Change listeners need to implement the javax.swing.event.ChangeListener interface and are registered with the needle using the addChangeListener() method. You can disable this mechanism by setting the needle’s sendEvents property to false. See also [Section 7.6, “Adding User](#_bookmark310) [Interaction to Needles,” on page 12](#_bookmark310)0.

##### Setting the Style

The indicatorStyle property controls the style used for indicators and needles. There are seven possible built-in shapes, as shown in the following figure:



*Figure 68 The built-in indicator shapes*

To specify the indicator style, set it in the constructor or set the indicatorStyle property. The valid values are: JCIndicatorStyle.ARROW (*needle default*), JCIndicatorStyle.CIRCLE, JCIndicatorStyle.POINTER, JCIndicatorStyle.RECTANGLE (*indicator default*), JCIndicatorStyle.TAILED\_ARROW, JCIndicatorStyle.TAILED\_POINTER, and JCIndicatorStyle.TRIANGLE.

##### Setting the Width

The width of an indicator is measured in pixels. The width is controlled using the indicatorWidth property, or the needleWidth property in the case of a needle. To specify the width, set it in the constructor or set the indicatorWidth or needleWidth properties (as appropriate). Both properties are of type double.

##### Setting the Size of an Indicator

The indicator’s length is based on the associated scale and is set as a decimal fraction of the scale’s dimensions using its innerExtent and outerExtent properties. In the circular case, an indicator begins at the center of the circular scale and extends outwards. For example, if its outerExtent property is set to 1.0, the indicator’s tip lies on the circumference of the associated circular scale. In the linear case, an indicator’s extents are measured from the top of the gauge area when the orientation is horizontal, or from the left-hand edge when the scale is vertical.

##### Setting the Size of a Needle

As a subclass of JCAbstractIndicator, a needle has inner and outer extents, and because it is a subclass of JCAbstractNeedle, it has a length property as well. Setting the needle’s length is equivalent to setting its outer extent. If you want to have the needle begin away from the center of a circular gauge, set its inner extent to some positive value. The value is expressed as a ratio based on the radius. Likewise, you can offset a linear needle from the top of a linear horizontal scale by setting its inner extent. For a linear needle, the inner extent can be positive or negative, but if it is negative, you may need to set a border on the scale to prevent the needle from being clipped.

##### Reversing an Indicator or Needle

An indicator that lacks longitudinal symmetry may be reversed. For example, the default JCIndicatorStyle.ARROW points outwards on a circular scale. To reverse the direction of an indicator, set the reversed property to true. The arrow points toward the center rather than to the circumference.

##### Setting the Color

By default, indicators and needles are black. To change the color, you can either specify it in the indicator or needle constructor or set the color property. The value is a Color object.

For example:

// Set the color of an indicator indicator.setColor(Color.green);

##### Hiding an Indicator or Needle

By default, indicators and needles are displayed on the gauge. To hide a needle or indicator, set its visibility property to false.

##### Defining a Custom Indicator Style

You can provide your own indicator style if you require a custom shape. The method is the same as that used in [Section 6.4.5, “Defining a Custom Tick Style,” on page 10](#_bookmark265)8. The simplest way is to use the JCIndicatorStyle constructor that allows you to define a new shape. If you want to keep your new indicator styles for general reuse as class constants, extend JCIndicatorStyle and define a shape as arrays of coordinate points using the same format as you would for java.awt.Rectangle. Here is an example:

import com.klg.jclass.swing.gauge.JCIndicatorStyle;

public class MyIndicatorStyle extends JCIndicatorStyle {

// Create a needle in the form of a diamond using these array values public static final JCIndicatorStyle DIAMOND = new JCIndicatorStyle(

new int[] {0, 100, 200, 100},

new int[] {0, 100, 0, -100},

4);

}

The JCIndicatorStyle constructor that allows you to define a new shape has been used here to define the class constant.

The indicator is positioned so that the origin of the polygon is drawn at the indicator’s inner extent position and that the largest x direction extent projected onto the x-axis is drawn at the indicator’s outer extent position. The polygon is scaled accordingly.

Indicators can be sized however you want, but if the inner or outer extent specification causes the indicator to be drawn outside the component’s boundary the indicator will be clipped. You may increase the border size to compensate, but borders are not scaled when components are resized. Since the indicator’s length is defined as a fraction of its associated scale’s radius, it may still elongate past the border if the component is expanded too much. An alternative approach is to use the scale’s zoomFactor property, set to some value less than one. This has the effect of shrinking the scale so that its boundary is less than its true radius. An indicator whose length is greater than 1.0 appears to extend beyond the scale. Because it is really inside the scale’s actual boundary it can be resized without clipping.

#### Adding User Interaction to Needles

The main difference between needles and indicators is that users can interact with needles. By default, interaction is disabled. You can choose to have a needle respond to a user’s mouse clicks, mouse drags, or both.

To add interaction to a needle, set the interactionType property of the needle. The following table summarizes the valid values:

|  |  |
| --- | --- |
| **InteractionType** | **Description** |
| JCAbstractNeedle.InteractionType.NONE *(default)* | No interaction. |
| JCAbstractNeedle.InteractionType.CLICK | Needle responds to mouse clicks, but not mouse drags. |
| JCAbstractNeedle.InteractionType.DRAG | Needle responds to mouse drags, but not mouse clicks. |
| JCAbstractNeedle.InteractionType.CLICK\_DRAG | Needle responds to both mouse clicks and mouse drags. |

See *GaugeInteractionExample.java* for a demonstration of the use of these interaction types.

When user interaction is enabled, a mouse click and/or mouse drag on the gauge generates a pick event. For more information, see [Section 5.7, “Enabling User Interaction](#_bookmark237) [for Scale Values,” on page 100.](#_bookmark237)

The needle value is updated to point to the value selected by the user. For more information, see [Section 7.5.1, “Updating the Value,” on page 11](#_bookmark296)7.

# 8

## Defining Text Components

[*Overview of Text-based Objects*](#_bookmark314)■ [*Summary of Properties Used*](#_bookmark315)[*Defining Headers and Footers*](#_bookmark316)■ [*Defining Legends*](#_bookmark318)■ [*Defining Labels*](#_bookmark340)

[*Understanding the Constraint Mechanism*](#_bookmark346)

The header, footer, legend, and labels are the text-based components of a gauge. Headers and footers can be used to add titles or other explanatory to your gauge. Legends are used to summarize elements in a gauge. Labels are used to annotate a gauge. This chapter describes the behavior of these objects and explains how to customize them.

#### Overview of Text-based Objects

This section describes the object inheritance for each of the text-based objects.

Legend

The following diagram shows the inheritance for the legend objects.

JCMultiColLegend

JCGridLegend

com.klg.jclass.util.legend.JCLegend

java.io.Serializable

javax.swing.JComponent

Changeable

*Figure 69 Legend objects inheritance*

JCLegend is an abstract class that outlines the properties and methods required to create a legend. It implements the Changeable and java.io.Serializable interfaces.

JCGridLegend extends JCLegend and provides properties and methods to create a single column legend.

JCMultiColLegend extends JCLegend and provides properties and methods to create a legend with multiple columns.

Headers and Footers

Headers and footers are JLabels by default, but can be any JComponent.

Labels

Labels are JLabels that are placed on the gauge using a constraints mechanism. The following diagram shows the inheritance for the constraint and layout objects.

LinearLayout

LinearConstraint

RadialLayout

RadialConstraint

GaugeLayout

GaugeConstraint

LayoutManager2

java.io.Serializable

*Figure 70 Constraint and layout objects inheritance for label placement*

GaugeConstraint is an abstract class that outlines the properties and methods required to describe how components can be placed on a gauge relative to the gauge. It implements the java.io.Serializable interface.

RadialConstraint extends GaugeConstraint and provides the properties and methods required to describe how a component should be placed in relation to a circular gauge.

LinearConstraint extends GaugeConstraint and provides the properties and methods required to describe how a component should be placed in relation to a linear gauge.

GaugeLayout is an abstract class that outlines properties and methods to place a label that uses a constraint. It implements the LayoutManager2 and java.io.Serializable

interfaces. RadialLayout extends GaugeLayout and is used for components that lay out their child components in a circular manner, such as JCCircularScale, JCCircularTick and JCGaugeArea (when it is a child of JCCircularGauge). LinearLayout extends GaugeLayout and is used for components that lay out their child components in a linear fashion, such as JCLinearScale, JCLinearTick and JCGaugeArea (when it is a child of JCLinearGauge). Constraints are used in conjunction with a layout manager and are supplied when adding child components to a parent with the corresponding layout. For example, when adding a label to JCCircularScale, you would use a RadialConstraint to position the label within the scale because JCCircularScale uses a RadialLayout. For more information, see [Section 8.6, “Understanding the Constraint Mechanism,” on page](#_bookmark346) [131](#_bookmark346).

#### Summary of Properties Used

The following table summarizes the properties (grouped where appropriate) used in this chapter and provides links to the associated procedures. For a list of all the properties and methods available for text components, see the *API documentation*.

|  |  |
| --- | --- |
| **Properties** | **Procedures** |
| anchor | [Section 8.4.2.3, “Positioning the Legend,” on page 125](#_bookmark330) |
| extent angle | [Section 8.6.1, “RadialConstraint and RadialLayout,” on](#_bookmark348) [page 131](#_bookmark348) |
| extent position | [Section 8.6.2, “LinearConstraint and LinearLayout,”](#_bookmark350) [on page 131](#_bookmark350) |
| header footer visible | [Section 8.3, “Defining Headers and Footers,” on page](#_bookmark316) [124](#_bookmark316) |
| itemTextAlignment itemTextToolTipEnabled maxItemTextWidth truncateMode useEllipsisWhenTruncating | [Section 8.4.2.4, “Setting the Width of the Legend and](#_bookmark332) [its Columns,” on page 126](#_bookmark332) |
| legendPopulator legendRenderer | [Section 8.4.4, “JCLegend Interfaces,” on page 128](#_bookmark337) |
| rangeName visible | [Section 8.4, “Defining Legends,” on page 124](#_bookmark318) |
| orientation | [Section 8.4.2.2, “Setting the Legend Orientation,” on](#_bookmark328) [page 125](#_bookmark328) |

#### Defining Headers and Footers

When you create an instance of a gauge object, a header and a footer are created automatically. The default header and footer are JLabel objects. By default, the header and footer are hidden and contain no text.

To show the header and footer, set their Visible properties to true.

// Show the header and the footer myGauge.getHeader().setVisible(true); myGauge.getFooter().setVisible(true);

To add text, set the text property of the JLabel.

// Supply the header with text

JLabel label= (JLabel)myGauge.getHeader(); label.setText("Header text");

If you want to choose a header or footer other than the default one, use the methods

setHeader() and setFooter() and specify which JComponent to use.

#### Defining Legends

When you create an instance of a gauge object, a legend object is created automatically. The default legend is a JCGridLegend object. By default, the legend is hidden.

To show the default legend, set its visible property to true.

// Show the legend myGauge.getLegend().setVisible(true);

The default legend uses a DefaultLegendPopulatorRenderer to populate and render the legend. The legend items of the default legend correspond to the current list of ranges within the gauge scale.

Ranges have default names like *range 0*, *range 1*, and so on, which is what is displayed in the legend if the ranges are not named.

To specify a range name, set the range’s rangeName property to a String. For more information, see [Chapter 9, “Defining Ranges.”](#_bookmark355)

// Name a range range.setRangeName("Danger zone");

##### Types of Legends

There are two types of legend objects: JCGridLegend (the default) for a single-column layout and JCMultiColLegend for a multiple-column layout. If these legends do not provide the desired functionality, you can customize the legend using the JCLegend Toolkit. For more information, see [Section 8.4.3, “Creating Custom Legends,” on page](#_bookmark335) [128](#_bookmark335).

* + - 1. Single-Column Legends

The classic single-column legend layout is provided by JCGridLegend. This is the default legend class used by JClass Gauge.

* + - 1. Multi-Column Legends

Multi-column legend layout is available using JCMultiColumnLegend. To use this class, follow these steps:

* + - * 1. Create an instance.
        2. Set the number of rows and columns.
        3. Set the legend property of the gauge to this instance. For example:

JCMultiColLegend mcl = new JCMultiColumnLegend();

mcl.setNumColumns(2); myGauge.setLegend(mcl);

This example creates a legend for the current gauge that has two columns. The number of rows depends on the number of items in the legend. To fix the number of rows, set the NumRows property. Both the number of rows and the number of columns are variable by default. To reset the number of rows and columns to a variable state after they have been fixed, call the appropriate set method with a negative value. If both the NumRows and NumColumns properties are set to fixed values, the legend will be of that exact size and will ignore any extra items.

##### Customizing Legends

The legend is a JComponent, and all properties such as border, colors, font, and so on, apply.

* + - 1. Displaying Range Labels in the Legend

The default legend displays the current list of ranges. To omit a range from the legend, set the range’s visibleInLegend property to false.

* + - 1. Setting the Legend Orientation

Use the legend orientation property to lay out the legend horizontally or vertically.

* + - 1. Positioning the Legend

You can use the legend anchor property to specify where to position the legend relative to the JCGaugeArea. Valid values are: JCLegend.NORTH, JCLegend.SOUTH, JCLegend.EAST, JCLegend.WEST, JCLegend.NORTHWEST, JCLegend.SOUTHWEST, JCLegend.NORTHEAST, and JCLegend.SOUTHEAST. The default value is JCLegend.EAST.

To specify an absolute position for the legend, you set the LayoutHints property from JCGauge and provide coordinates. For more information, see [Section 8.7, “Positioning](#_bookmark352) [Elements on the Gauge Object,” on page 13](#_bookmark352)2.

* + - 1. Setting the Width of the Legend and its Columns

If the legend text is very long, you may find that by default the legend becomes very wide, leaving proportionally less room for the gauge itself. You can improve the balance between gauge and legend by controlling the width of the legend. You have two choices for setting the width. You can set the width of the legend explicitly and allow the columns within the legend to be sized automatically, or you can set the column widths and allow the legend width to be calculated.

Specifying the Legend Width

To set the width of the entire legend, you set the LayoutHints property from JCGauge and provide the width of the legend rectangle. For example, the following code snippet sets the width of the legend to 200 pixels:

gauge.setLayoutHints(chart.getLegend(),

new Rectangle(Integer.MAX\_VALUE, Integer.MAX\_VALUE, 200, Integer.MAX\_VALUE));

Integer.MAX\_VALUE means that the dimension is dynamic. In the above example, there are no restrictions on the positioning of the legend or on the height dimension. For more information, see [Section 8.7, “Positioning Elements on the Gauge Object,” on page 13](#_bookmark352)2.

Specifying Column Widths

To set the width of columns within the legend, you set the maxItemTextWidth property from JCLegend and specify the width in pixels as a non-negative int. By default, the value is Integer.MAX\_VALUE, which means the width is dynamic.

For example, the following code sets the width of each of the columns in the legend to be 100 pixels.

legend.setMaxItemTextWidth(100);

To specify different widths for columns in a multicolumn legend, you need to provide an additional parameter that specifies the column number. For example, the following code specifies column widths of 50, 100, and 75 pixels for consecutive columns in a three- column legend:

legend.setMaxItemTextWidth(50, 0);

legend.setMaxItemTextWidth(100, 1);

legend.setMaxItemTextWidth(75, 2);

Handling Truncated Text

Whichever way you choose to restrict the width of a legend, you can set properties to control what happens when the length of the text exceeds the width of a column. By

default, column text is aligned with the leading edge of the column (for example, it is aligned left in a left-to-right orientation). When text is truncated, the trailing text (the rightmost text in a left-to-right orientation) is hidden and an ellipsis is displayed in its place. You can modify this behavior by setting the JCLegend properties described below.

To change the text alignment, you set the itemTextAlignment property and specify the value using one of the following enumerations: SwingConstants.LEFT, SwingConstants.RIGHT, SwingConstants.CENTER, SwingConstants.LEADING (*default*), or SwingConstants.TRAILING. For example, the following code causes text to be right aligned for all columns except the second column (column 1), where the text is centered:

legend.setItemTextAlignment(SwingConstants.RIGHT); legend.setItemTextAlignment(SwingConstants.CENTER, 1);

To change how the text is truncated, you set the truncateMode property. The following table shows the possible values followed by how the text would appear:

|  |  |
| --- | --- |
| **Values for setTruncateMode()** | **Result** |
| JCLegend.TRUNCATE\_LEFT | ...text |
| JCLegend.TRUNCATE\_MIDDLE | text...text |
| JCLegend.TRUNCATE\_RIGHT | text... |
| JCLegend.TRUNCATE\_END | ...text... |
| JCLegend.TRUNCATE\_LEADING | In a left-to-right orientation, same as  JCLegend.TRUNCATE\_LEFT.  In a right-to-left orientation, same as  JCLegend.TRUNCATE\_RIGHT. |
| JCLegend.TRUNCATE\_TRAILING (*default*) | In a left-to-right orientation, same as  JCLegend.TRUNCATE\_RIGHT.  In a right-to-left orientation, same as  JCLegend.TRUNCATE\_LEFT. |

For example, the following code causes text to be truncated on the right for all columns, except for the third column (column 2), where the ends are truncated:

legend.setTruncateMode(JCLegend.TRUNCATE\_RIGHT); legend.setTruncateMode(JCLegend.TRUNCATE\_END, 2);

To stop the ellipsis from being displayed, you set the UseEllipsisWhenTruncating property to false. There will be no visual indication that text is hidden. This property always applies to all columns.

You can also choose to display the entire legend item text in a tooltip whenever the mouse hovers over a legend item. The tooltip appears whether or not the legend text is truncated. To activate the tooltips, set the ItemTextToolTipEnabled property to true. This property always applies to all columns.

##### Creating Custom Legends

If you need a legend that itemizes other things, like needles or ticks, you can use the JCGaugeLegendEntry interface. Each item to be included in the legend is wrapped in an object that implements this interface. These items are then aggregated into a List. You can then create a new instance of DefaultLegendPopulatorRenderer with the list.

For example:

List items = ... //create your list of items JCLegend legend = myGauge.getLegend(); DefaultLegendPopulatorRenderer legPop =

new DefaultLegendPopulatorRenderer(legend, items); legend.setLegendPopulator(legPop); legend.setLegendRenderer(legPop);

See the *GaugeInteractionExample* for a full example.

The JCGaugeLegendEntry interface includes methods for getting the legend color, legend label, and visibility properties. For more information, look up JCGaugeLegendEntry in the *API documentation*.

Note: JCAbstractRange implements JCGaugeLegendEntry and therefore the scale’s list of ranges is a valid legend item list.

##### JCLegend Interfaces

There are two interfaces associated with JCLegend. JCLegendPopulator is an interface implemented by classes that want to populate a legend with data, and JCLegendRenderer is an interface implemented by a class that wants to help render the legend.

DefaultLegendPopulatorRenderer implements both interfaces and provides a built-in mechanism for itemizing range objects in a legend.

##### Enabling User Interaction in Legends

You can use the pick mechanism to react to user clicks on legend items. The JCGaugePickEvent passed to the pick() method of a pick listener has an info property that gets filled with extra information when the situation warrants it. When a user clicks on a legend term, the info field is filled in with the legend item object. You can query this info object to detect which item as been selected.

Note: The legend title cannot be picked; its info property remains null.

See the *GaugeInteractionExample* for an example. For more information, see [Section 5.7,](#_bookmark237) [“Enabling User Interaction for Scale Values,” on page 100.](#_bookmark237)

#### Defining Labels

You can place any number of labels anywhere within the boundaries of the gauge area using a gauge’s addLabel() method, which in turn uses a RadialConstraint or a LinearConstraint. Because it is a JLabel, it has user-controllable text, position, background and foreground color, images, and borders.

The RadialConstraint class, whose constructor is RadialConstraint(JCGauge gauge, double extent, double angle), lets you specify a label’s position by giving a distance from the center of the scale and an angle. The constructor for a linear constraint is LinearConstraint(JCGauge gauge, double extent, int position), which lets you specify a label’s position by giving the extent in the transverse direction to the scale and a position along the scale in pixels. For more information, see [Section 8.6, “Understanding](#_bookmark346) [the Constraint Mechanism,” on page 13](#_bookmark346)1.

Note that there is an automatic mechanism for providing numeric labels on tick objects or for specifying labeled ticks in user-specified formats. For more information, see [Section](#_bookmark269) [6.5, “Customizing Tick Labels,” on page 11](#_bookmark269)0.

You can choose a location within the gauge area by specifying the location of the center of the rectangle containing the RadialConstraint or LinearConstraint class.

You can take advantage of JLabel’s ability to process HTML tags to display text with mixed fonts and multiple lines of text. Set a font using the <font> tag and the color using color = HTMLcolorValue within the tag. If you are adding text to a circular gauge, you can do the following:

JLabel l1 = new JLabel("<html><font color=black>

Start Angle = 90\u00B0"); l1.setToolTipText("Start Angle = 90\u00B0"); gauge.addLabel(l1, new RadialConstraint(gauge, 0.60, 15), 0);

You can also control whether a label is drawn using its visible property, set a border, or set other properties of the JLabel.

Even though the method is called addLabel(), the method actually allows for any

JComponent to be added in this fashion.

##### Aligning Text

The lines of text within a label can be either centered or right-justified. If text is not centered, it may appear that RadialConstraint is not positioning the text at the correct angle. In [Figure 71](#_bookmark344) both text areas are aligned vertically, but without the borders on the components it appears that the lower label is not vertically aligned. Here is the code that produces this layout:

JLabel l1 = new JLabel("<html><font color=black>

<P ALIGN=CENTER>Start Angle <br>= 0\u00B0"); l1.setToolTipText("Start Angle = 0\u00B0");

l1.setBorder(new BevelBorder(BevelBorder.RAISED)); gauge.getGaugeArea().add(l1,

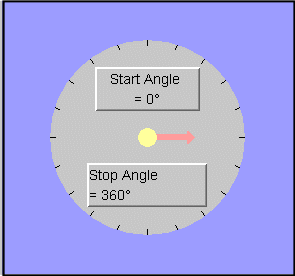
new RadialConstraint(gauge, 0.50, 90), 0);

JLabel l2 = new JLabel(

"<html><font color=black><P>Stop Angle <br>= 360\u00B0"); l2.setToolTipText("Stop Angle = 360\u00B0");

l2.setBorder(new BevelBorder(BevelBorder.RAISED)); gauge.getGaugeArea().add(l2,

new RadialConstraint(gauge, 0.50, 270), 0);



*Figure 71 If text is not centered it may appear to be placed at the wrong angle*

The center of the label is place at the position indicated by the constraint.

##### Sample Code

The following code snippet shows the addition of a label positioned halfway from the center of a gauge to its circumference at an angle of 45°.

RadialConstraint rConstraint = new RadialConstraint(gauge, 0.50, 45) JLabel label = new JLabel("Start Angle = 0"); label.setToolTipText("Start Angle = 0\u00B0"); gauge.getGaugeArea().add(label, rContstraint, 0);

Usage is the same for a linear constraint. This one puts a label at (0, 0) on a linear scale:

JLabel l0 = new JLabel("<html><font color=black>0"); l0.setToolTipText("0 marks the spot!"); gauge.getGaugeArea().add(l0, new LinearConstraint(gauge, 0.0, 0));

#### Understanding the Constraint Mechanism

JCGauge uses a constraint mechanism to place labels and other components on a gauge. RadialConstraint is used for circular gauges, while LinearConstraint is for linear gauges.

##### RadialConstraint and RadialLayout

Class RadialLayout uses an instance of RadialConstraint to position a component at a given angle and at a specified proportional distance from the center of the associated circular gauge. Thus, the gauge employs RadialConstraint classes to facilitate laying out gauge objects in such a way that the objects’ angular positions are maintained as the gauge is resized, as well as maintaining their proper radial proportions.

RadialConstraint supports the placement of any component on the gauge area, not just indicators, needles, ranges, and ticks. Usually these are labels used to annotate a circular gauge, but they can be any JComponent, even another gauge. For more information, see [Section 1.4.3, “Layering Gauges,” on page 26](#_bookmark39).

Constructors

RadialConstraint has a single constructor to which is passed a *gauge*, an *extent*, and an

*angle*.

* + - * The extent parameter specifies the radial distance for the placement of the component.
      * The angle parameter specifies the angle.

The center of component’s bounding rectangle is placed on the gauge at the point defined by the two parameters. Typically an instance of RadialConstraint is passed via the addLabel() method in JCCircularGauge, which passes it to an add() method that knows how to use RadialLayout to position the component.

Here’s an example:

JCCircularGauge gauge = new JCCircularGauge();

JLabel label = new JLabel("<html>Pressure (lbs/in&sup2;)"); gauge.addLabel(label, new RadialConstraint(gauge, 0.35, 90));

##### LinearConstraint and LinearLayout

Class LinearLayout uses an instance of LinearConstraint to position a component at a given extent and at a specified pixel distance from the origin of the linear gauge. Thus, the gauge employs LinearConstraint classes to facilitate laying out gauge objects in such a way that the objects’ relative positions are maintained as the gauge is resized.

It supports the placement of any component on the gauge area, not just indicators, needles, ranges, and ticks. Usually these are labels used to annotate a linear gauge, but they may be any JComponent, even another gauge.

Constructors

LinearConstraint has a single constructor which is passed a *gauge*, an *extent*, and a *position*. The extent parameter specifies the proportional distance from the top left of the rectangle enclosing the gauge. The distance is vertical for horizontal scales and horizontal for vertical scales, and is specified as a ratio of this distance to the height or width of the scale. The position parameter specifies the distance as an integer representing a percentage of the height or width from the top or left of the scale.

The center of the positioned component’s bounding rectangle is placed on the gauge at the point defined by these two parameters. Typically an instance of LinearConstraint is passed via the addLabel() method in JCLinearGauge, which passes it on an add() method that knows how to use LinearLayout to position the component.

Here’s an example:

JCLinearGauge gauge = new JCLinearGauge(); JLabel label = new JLabel("Pressure Point");

gauge.addLabel(label, new LinearConstraint(gauge, 0.35, 90));

#### Positioning Elements on the Gauge Object

Each of the main gauge elements (header, footer, legend, and gauge area) has properties that control its position and size. While the gauge can automatically control these properties, you can also customize the following:

* positioning of any element
* size of any element

When the gauge controls positioning, it first allows space for the header, footer, and legend, if they exist (size is determined by contents, border, and font). The gauge area is sized and positioned to fit into the largest remaining rectangular area. Positioning adjusts when other gauge properties change.

Changing the Location and Size

To specify the absolute location and size of a gauge element, call setLayoutHints() in JCGauge with the object you wish to move and a rectangle containing its desired X and Y location, width, and height. If you desire any of those values to be calculated rather than set, make them equal to Integer.MAX\_VALUE.

For example, the following code sets the legend to be 200 pixels wide and 300 pixels high and places the top-left corner of the legend at the x,y coordinate (0,150):

myGauge.setLayoutHints(legend, newRectangle(0,150,200,300));

Whereas this code allows the legend size to be dynamic, but places the legend at (0,150):

myGauge.setLayoutHints(legend, Rectangle(0,150, Integer.MAX\_VALUE,Integer.MAX\_VALUE, Integer.MAX\_VALUE));

# 9

## Defining Ranges

[*Overview of the Range Objects*](#_bookmark357)■ [*Summary of Properties Used*](#_bookmark358)

[*Defining a Range*](#_bookmark359)■ [*Customizing a Range*](#_bookmark366)[*Specifying a Foreground Image in a Linear Range*](#_bookmark375)■ [*Creating an Offset Range in a Circular Gauge*](#_bookmark377)

You can add range objects to your gauge. Ranges enable you to highlight sections of the scale that hold some meaning, such as, a value that falls within a certain range is within expected limits. On circular scales, ranges are represented as slices (as in a pie chart) or arcs. On linear scales, ranges are rectangular bars. This section describes the range objects, and then shows you how to add ranges and customize them.

#### 9.1 Overview of the Range Objects

The following diagram shows the inheritance for range objects.

JCRange

JCGaugeLegendEntry

JCAbstractRange

Serializable

javax.swing.JComponent

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | |
| JCCircularRange |  |  | JCLinearRange |
|  | |

*Figure 72 Range object inheritance*

JCRange is the interface that represents a range. If you create a custom gauge object, you can use this interface to define a custom range.

JCAbstractRange extends JComponent and implements JCRange, JCGaugeLegendEntry, and java.io.Serializable. JCAbstractRange encapsulates the properties associated with a range, including the scale, the stop and start values of the range, the extents

133

required to draw the range, the range name, and whether or not to include the range name in the legend. It also defines methods to set and get the values of these properties.

JCCircularRange extends JCAbstractRange. It inherits the properties encapsulated in

JCAbstractRange and includes methods to draw a range on a circular scale.

JCLinearRange extends JCAbstractRange. JCLinearRange inherits the properties encapsulated in JCAbstractRange and includes methods to draw a range on a linear scale. It also enables the use of an image in the background of the range area.

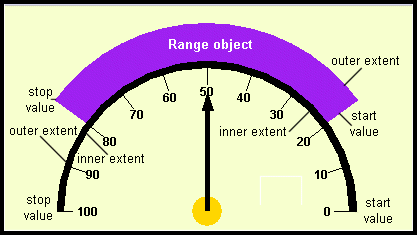
#### Summary of Properties Used

The following table summarizes the properties (grouped where appropriate) used in this chapter and provides links to the associated procedures. For a list of all the properties and methods available for text components, see the *API documentation*.

|  |  |
| --- | --- |
| **Properties** | **Procedures** |
| background  image imageScaled  innerExtent outerExtent  startValue stopValue | [Section 9.4.2, “Setting the Range Color,” on page 137](#_bookmark371)  [Section 9.4.3, “Specifying and Scaling an Image,” on page 137](#_bookmark373)  [Section 9.4.1, “Setting the Breadth and Location of a Range,”](#_bookmark368) [on page 136](#_bookmark368)  [Section 9.3.2, “Setting Range Values,” on page 135](#_bookmark364) |

#### Defining a Range

This section describes how to create a range object. [Figure 73](#_bookmark361) shows two range objects on a circular scale. The thinner one spans the entire scale and appears as the circumference of the scale. The thicker one spans the region between tick marks 20 and 80.



*Figure 73 Using ranges to show the scale circumference and to highlight a portion of the scale*

##### Constructing a JCCircularRange or JCLinearRange Object

The constructors require the scale parameter, that is, a range object needs to know the parent scale to which it belongs. All other properties are assigned default values.

// Create a range object and associate it with the scale

// For a linear gauge, substitute JCLinearRange for JCCircularRange JCAbstractRange range = new JCCircularRange(myGauge.getScale());

The following example shows a complete constructor with default values for all the properties. To learn how to customize these properties, see [Section 9.4, “Customizing a](#_bookmark366) [Range,” on page 13](#_bookmark366)6.

// Create a range object and associate it with the scale

// For a linear gauge, substitute JCLinearRange for JCCircularRange JCAbstractRange range = new JCCircularRange(

Color.black, // background color of range scale, // associated scale

0.0, // innerExtent of range area

1.0, // outerExtent (at the radius)

0, // startValue to begin the range

0); // stopValue (set > 0 to show the range)

Now you need to add the range to the scale specified in the constructor.

// Add the range to the scale associated with ‘myGauge’ scale.add(range);

Recall that components are drawn on the gauge in the reverse order in which they are added. [Figure 74](#_bookmark370) shows the result of one such ordering. The circumference range is drawn first, followed in succession by the 20–80 range, the tick marks, the needle, and finally the center. For more information, see [Section 1.5.1, “How Components are](#_bookmark44) [Rendered on a Gauge,” on page 28](#_bookmark44).

##### Setting Range Values

The startValue and stopValue properties control where the range begins on the scale and where it ends. By default, a range begins at the scale’s minimum value and ends at the scale’s maximum value.

To specify the span of the range, set the values in the constructor or set the startValue and stopValue properties. Both properties are of type double. The values should fall within the scale’s minimum and maximum values, unless you are purposely creating an offset range. For more information, see [Section 9.6, “Creating an Offset Range in a](#_bookmark377) [Circular Gauge,” on page 13](#_bookmark377)9.

For example,

// Specify the start and end values of the range range.setStartValue(20);

range.setStopValue(80);

#### Customizing a Range

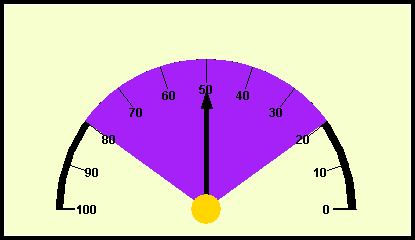
You can change the color and breadth of the range. You can also choose to specify an image for a range area.

##### Setting the Breadth and Location of a Range

The innerExtent and outerExtent properties control the breadth of the range, that is, the thickness of the band that represents the range, and for circular ranges, where that band is located in relation to the scale. As with indicators and needles, the values for the range extent properties are specified as a percentage of the scale’s radius; if the gauge is resized, the range retains its size proportionate to the scale.

In a linear range, the default values for the innerExtent and outerExtent properties cause the breadth of the range to cover the width of the scale in a vertical orientation or the height of the scale in a horizontal orientation.

In a circular range, the default inner extent is 0.0 and the default outer extent is 1.0. The following figure shows a circular range that spans the values 20 to 80 and uses the default extents.



*Figure 74 A circular gauge with two ranges: a 20–80 range and a range marking the scale’s circumference*

To specify different extent values, set the values in the constructor or set the innerExtent and outerExtent properties. Both properties are of type double. The values represent a percentage of the scale’s radius expressed as a decimal fraction.

For example, in [Figure 74](#_bookmark370) the arc that marks the circumference of the scale is a range. The following code snippet shows how the range is defined.

// Specify the extents of the range after it has been instantiated range.setInnerExtent(0.99);

range.setOuterExtent(1.01);

This range straddles the circumference (1.0) of the circular scale, starting just before the circumference (0.99) and extending the same distance past it (1.01). The breadth of the range is calculated to be 1.01 - 0.99 = 0.02, which means that the arc represents 2% of the scale’s radius.

##### Setting the Range Color

To set the color of the range, you can either define it in the constructor or set the background property inherited from JComponent. The opaque property is ignored for ranges.

// Specify a background color range.setBackground(Color.blue);

##### Specifying and Scaling an Image

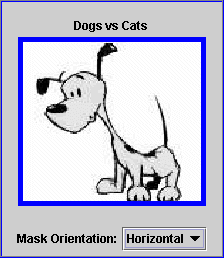
You can set an image in the background of the range. By default, the image scales to fill its container. If the image has transparent pixels and a background color is specified for the container, the background color will show through the transparent pixels.

To specify an image in the background, set the image property to an Image or a URL.

By default, the image scales with its container. If you prefer not to have the image scale, set the imageScaled property to false. In this case, the image is centered within the container. If the image is smaller than its container, the background (if drawn) shows around the image. If the image is larger than its container, the image is clipped.

For example:

// Specify an image and set it to scale with the container range.setImage(this.getClass().getResource("dog.gif")); range.setImageScaled(true);



*Figure 75 Range that uses an image in the background*

The image, including its imageScaled setting, is stored as a PortableImage object. For more information, see [Section 1.4.2, “Images and the PortableImage Object,” on page 25](#_bookmark37).

XML Users: If you want to save image information, you also need to set the imageOutputProperties property. For more information, see [Section 11.5, “Preparing to](#_bookmark439) [Save Information About Images to XML,” on page 15](#_bookmark439)7.

#### Specifying a Foreground Image in a Linear Range

For linear gauges only, you can use a foreground image in the range. A foreground image is useful when you want to show coverage. For example, in a linear gauge that looks like a thermometer, you can use a background image of an empty thermometer and a foreground image of a full thermometer. You can then set, as a percentage, how much of the foreground image covers the background image.

To set the foreground image, set the foregroundImage property to an Image or a URL. If a background image is also specified, the foreground image uses the imageScaled property of the background image. This means that the images scale (or do not scale) together.

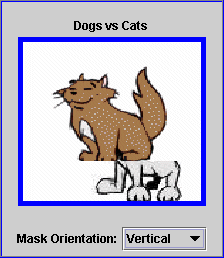
To specify the how much of the foreground image covers the background image, you set the foregroundCoverage property using a decimal fraction that represents percentage of coverage. The default is 0.0. If you specify a coverage but no image, a rectangle is used instead. The rectangle is filled with the foreground color.

By default, the foreground image covers the background image in a horizontal orientation from left to right. To change the orientation, you set the maskOrientation property. Valid values are: JCLinearScale.Orientation.Horizontal (*default*) and JCLinearScale.Orientation.Vertical. When the orientation is vertical, the image covers the background image from top to bottom.

For example, the following code snippet sets up a foreground image that covers 75% of the background image from top to bottom.

// Create the foreground image range.setForegroundImage(this.getClass().getResource("cat.gif"));

// Set the coverage and orientation range.setForegroundCoverage(0.75); range.setMaskOrientation(JCLinearScale.Orientation.Vertical);



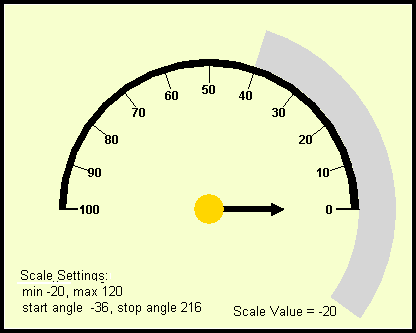
*Figure 76 Foreground image overlaying a background image by 75%*

The image, including its imageScaled setting, is stored as a PortableImage object. For more information, see [Section 1.4.2, “Images and the PortableImage Object,” on page 25](#_bookmark37).

XML Users: If you want to save a foreground image to XML, you also need to set the foregroundImageOutputProperties property. For more information, see [Section 11.5,](#_bookmark439) [“Preparing to Save Information About Images to XML,” on page 15](#_bookmark439)7.

#### Creating an Offset Range in a Circular Gauge

If you want to offset a range from the scale markings, you can create a circular scale spanning all the values that you need to display, but supply tick marks for one portion of the scale and a range for another portion.



*Figure 77 An offset range extends beyond the scale’s tick marks*

As shown in [Figure 77](#_bookmark379), an offset appears to begin before the start of the scale. The thick, offset range has a startValue of -20 and a stopValue of 40. The scale’s settings are max=120, min=-20, startAngle=-36°, stopAngle=216°. These were calculated to be consistent with the offset given to the range, and to maintain a semicircular appearance for the labeled part of the scale.

# 10

## Defining the Center Object in a

**Circular Gauge**

[*Overview of the Center Object*](#_bookmark383)■ [*Summary of Properties Used*](#_bookmark385)[*Defining the Center*](#_bookmark386)■ [*Customizing the Center*](#_bookmark391)

The center of a circular gauge is a disc that is displayed at the origin of the circle (*(0,0)* in a coordinate system), whether the gauge is a full circle or a partial one. There can be one center object per gauge. This chapter describes how to create and customize a center object.



*Figure 78 Center object displaying an image*

#### Overview of the Center Object

The following diagram shows the inheritance for the center object.

JCCenter

java.io.Serializable

javax.swing.JComponent

*Figure 79 Center object inheritance*

JCCenter extends JComponent and implements java.io.Serializable. It contains the properties and methods required to define the center of a circular gauge.

#### Summary of Properties Used

The following table summarizes the properties (grouped where appropriate) used in this chapter and provides links to the associated procedures. For a list of all the properties and methods available for text components, see the *API documentation*.

|  |  |
| --- | --- |
| **Properties** | **Procedures** |
| background  image imageScaled  radius | [Section 10.4.2, “Setting the Background Color,” on page 144](#_bookmark395) [Section 10.4.3, “Specifying and Scaling an Image,” on page 144](#_bookmark397)  [Section 10.4.1, “Setting the Center Radius,” on page 143](#_bookmark393) |

#### Defining the Center

This section describes how to access the default center object and how to create a new center object.

##### Accessing the Preconstructed Center Object

If you created your gauge using the no-arguments circular gauge constructor, you already have a center object associated with the scale. The following example shows how to access the center.

// Access the default center object

JCCenter center = myCircularGauge.getCenter();

The center object has default values for all properties. You can change the value of any of the properties. For more information, see [Section 10.4, “Customizing the Center,” on](#_bookmark391) [page 14](#_bookmark391)3.

##### Constructing a JCCenter Object

If your gauge does not have a center object, you can create one using the JCCenter constructor. The constructor requires the scale parameter, that is, it needs to know its parent scale. All other properties are assumed to be default values.

// Get the scale for the circular gauge "myGauge" JCAbstractScale scale = myGauge.getScale();

// Create the center object and associate it with the scale JCCenter center = new JCCenter(scale);

If you want to change the default values, you can use one of the other constructors provided. The following example shows how to create a JCCenter object as a colored disc.

// Create the center object and set the color and radius JCCenter center = new JCCenter(

scale, // the parent scale Color.black, // background color 0.1); // radius expressed as a

fraction of the scale radius

The following example shows how to create a JCCenter object with a background image.

// Create the center object and specify an image

Image image = Toolkit.getDefaultToolkit().getImage("stopwatch.gif"); JCCenter center = new JCCenter(

scale, // the parent scale Color.black, // background color

image); // the image to use in the center

When you create a new JCCenter object, you need to set it as the center of your gauge.

// Set the object as the center of the gauge myCircularGauge.setCenter(center);

#### Customizing the Center

After a center has been created, you can change the background color and radius used for the center object. You can also choose to add a background image for the center.

##### Setting the Center Radius

The size of the center object is controlled by the radius property. When an image is used, this property is ignored unless the setScaleImage property is true. For more information, see [Section 10.4.3, “Specifying and Scaling an Image,” on page 14](#_bookmark397)4.

To change the radius, set the radius property to a double. The value needs to be expressed as a decimal fraction of the scale’s radius. Using a fraction allows the center to retain its size proportionate to the scale when the gauge is resized. For example, if you want the center radius to be one-fifth the size of the scale radius, you would set the value as 0.2.

// Set the center radius to be one-fifth of the scale radius myCircularGauge.getCenter().setRadius(0.2);

Recall that, if you are not using an image, you can also specify the radius in the JCCenter constructor. For more information, see [Section 10.3.2, “Constructing a JCCenter Object,”](#_bookmark389) [on page 14](#_bookmark389)2.

##### Setting the Background Color

To set the color of the center, you can either define it in the JCCenter constructor or set the background and opaque properties inherited from JComponent. By default, the opaque property is set to true.

// Specify a background color and make it opaque center.setBackground(Color.blue); center.setOpaque(true);

##### Specifying and Scaling an Image

You can set an image in the background of the center disk. By default, the image scales to fill its container. If the image has transparent pixels and a background color is specified for the container, the background color will show through the transparent pixels.

To specify an image in the background, you can specify it in the JCCenter constructor or set the image property to an Image or a URL.

By default, the image scales with its container. If you prefer not to have the image scale, set the imageScaled property to false. In this case, the image is centered within the container. If the image is smaller than its container, the background (if drawn) shows around the image. If the image is larger than its container, the image is clipped.

For example:

center.setImage(this.getClass().getResource("center.gif")); center.setImageScaled(true);



*Figure 80 Center with an image in background*

The image, including its imageScaled setting, is stored as a PortableImage object. For more information, see [Section 1.4.2, “Images and the PortableImage Object,” on page 25](#_bookmark37).

XML Users: If you want to save image information, you also need to set the imageOutputProperties property. For more information, see [Section 11.5, “Preparing to](#_bookmark439) [Save Information About Images to XML,” on page 15](#_bookmark439)7.

*Part III*

*Using XML*

***11***

## Creating Gauges and Graphs with XML

[*Overview of XML for JClass Gauge*](#_bookmark405)■ [*Creating a Gauge or Graph From XML*](#_bookmark416)[*Updating a Gauge or Graph From XML*](#_bookmark427)■ [*Saving a Gauge or Graph to XML*](#_bookmark433)

[*Preparing to Save Information About Images to XML*](#_bookmark438)■ [*Internationalizing Your XML-based Gauge*](#_bookmark448)

This chapter describes how to create, update, and save your gauge or graph using XML. It assumes that you have a working knowledge of XML.

#### Overview of XML for JClass Gauge

This section identifies the XML-related files and classes for JClass Gauge.

##### DTDs

DTD files are located in *JCLASS\_HOME/xml-dtd*.

* + - * *JCGauge.dtd* – Defines common elements required by all the gauge types, such as the

<gauge> tag. It also defines circular and linear gauge properties and graph properties.

* + - * *IndicatorGauge.dtd* – Defines indicator gauge and indicator gauge panel properties. Requires *JCGauge.dtd*.

In JClass Gauge, the elements, sub-elements, and attributes in the DTDs coincide, for the most part, with the objects, sub-objects, and properties within the gauge components.

Properties are specified as Strings in the XML file. The Strings are converted to the appropriate type by the JClass Gauge XML handler. For details, see [Chapter 12, “XML](#_bookmark454) [DTDs.”](#_bookmark454)

Note: The Color type is used in all the DTDs. Values for Colors can be specified in hexadecimal (#RRGGBB), as an RGB value (RRR-GGG-BBB), or as a color enum (such as “black” or “blue”).

##### Factory Classes

There are three JClass Gauge factory classes for use with XML:

* + - * com.klg.jclass.sgauge.JCGaugeFactory
      * com.klg.jclass.gauge.indicator.JCIndicatorGaugeFactory
      * com.klg.jclass.gauge.graph.JCGraphFactory

Each of them provides methods to create, update, and save a gauge or graph using properties encoded in XML. The factories are designed to be able to interpret multiple markup languages, but only XML is supported at this time. For more information, look up the factories in the *API documentation*.

##### LoadProperties Class

The LoadProperties class is responsible for the following tasks:

* + - * Telling the gauge or graph how to access image files based on the fileAccess properties of the <image-file> elements defined in the XML source. Access information for other external files can be supplied as well.
      * Passing user-defined objects to an external Java class when <external-java-code>

elements are defined in the XML source.

* + - * Identifying what to do when there is an error in reading an image from its source. Normally, the gauge or graph throws a JCIOException when this happens. However, you can ignore these exceptions and continue loading the gauge or graph by setting the ignoreExternalResourceExceptions property to true.

To use LoadProperties class, you create an instance of it and then pass the instance to the make\*() method. For more information, see [Section 11.2.3, “Constructing a](#_bookmark425) [LoadProperties Object,” on page 153](#_bookmark425) and look up the com.klg.jclass.util.io.LoadProperties class in the *API documentation*.

##### Images and the OutputProperties Class

The OutputProperties class encapsulates properties that are needed to save an image file to XML. To use the OutputProperties class, you create an instance of OutputProperties and then pass it to setImageOutputProperties() method of the container that holds the image. For example, if you have an image in the background of a gauge that you know you want to save to XML, you would set the imageOutputProperties property of the gauge. For more information, see [Section 11.4, “Saving a Gauge or Graph to XML,” on](#_bookmark433) [page 156](#_bookmark433) and look up the com.klg.jclass.util.io.OutputProperties class in the *API documentation*.

##### XML Examples

Examples are located in *JCLASS\_HOME/examples/gauge/xml*. Each example includes a sample XML file, a JAVA file that contains configuration details to create the gauge and load the properties from the XML file, and a compiled CLASS file.

#### Creating a Gauge or Graph From XML

Each of the factories has a make\*() method that creates a gauge or graph from a file, reader, stream, or URL. For example, JCGaugeFactory has a makeGauge() method. To create a gauge or graph, you encode your gauge or graph properties in XML. Then you create an instance of LoadProperties, which handles how properties are loaded from the XML source. Finally, you use the appropriate factory to create the gauge or graph from the XML and the LoadProperties instance.

The following subsection shows you how to use the factories. Subsequent subsections help you encode JClass Gauge properties to XML and specify the properties of the LoadProperties object.

##### Using the Factories to Make a Gauge or Graph

This section shows you how to use the make\*() method to create a gauge or graph. A complete example is shown for circular and linear gauges; the indicator gauge and graph sections describe how to modify that example.

* + - 1. Creating Circular and Linear Gauges from XML

The following code uses JCGaugeFactory to create a gauge from an XML file called *gauge.in.xml*. In the example, the XML file is specified as a String and assigned to inputSource. Alternatively, the inputSource could be an InputStream, a Reader, or a URL. Within the XML, you can define either a circular or linear gauge; JCGaugeFactory creates and returns the appropriate gauge based on the contents of the inputSource.

The LoadProperties object is called loadProps. Note that the factory requires a third parameter, JCGaugeFactory.XML, which is a static int that indicates the markup language used. Some typical error catching is included in the code sample.

// Specify where to look for the XML-based gauge properties String inputSource = "gauge.in.xml";

// Create the LoadProperties object LoadProperties loadProps = new LoadProperties();

// Create the gauge JCGauge myGauge = null;

try {

myGauge = JCGaugeFactory.makeGauge(

inputSource, loadProps, JCGaugeFactory.XML);

}

catch (JCIOException e) {

System.out.println("Error accessing external file:" + e.getMessage());

}

catch (JCParseException e) {

System.out.println("Error parsing file:" + e.getMessage());

}

catch (IOException e) {

System.out.println("Error reading " + inputSource +

":" + e.getMessage());

}

* + - 1. Creating Indicator Gauges and Panels from XML

Use the same code as in “[Creating Circular and Linear Gauges from XML](#_bookmark419),” but substitute JCIndicatorGauge or JCIndicatorPanel (for JCGauge) and JCIndicatorGaugeFactory (for JCGaugeFactory).

For example:

// Create an indicator gauge JCIndicatorGauge myGauge = null;

try {

myGauge = JCIndicatorGaugeFactory.makeIndicatorGauge( inputSource, loadProps, JCIndicatorGaugeFactory.XML);

}

* + - 1. Creating Graphs from XML

Use the same code as in “[Creating Circular and Linear Gauges from XML](#_bookmark419),” but substitute JCGraph (for JCGauge), JCGraphFactory (for JCGaugeFactory), and use makeGraph() (instead of makeGauge()).

For example:

// Create the graph JCGraph myGraph = null;

try {

myGraph = JCGraphFactory.makeGraph(

inputSource, loadProps, JCGraphFactory.XML);

}

##### Setting JClass Gauge Properties Using XML

JClass Gauge properties are enclosed in the <gauge> tag from *JCGauge.dtd*. For a list of elements, including their expected types or values, see [Chapter 12, “XML DTDs](#_bookmark454)”.

* + - 1. Sample XML File

The following example is taken from the *circular-gauge.xml* file, located in *JCLASS\_HOME/examples/gauge/xml.* Sample XML files for the other gauge types can be found in the same location.When the properties are loaded into the gauge, any properties that are not specified use default values.

<?xml version="1.0"?>

<!DOCTYPE gauge SYSTEM "JCGauge.dtd">

<gauge>

<circular-gauge gaugeType="Top\_Half"

width="400" height="300" background="#8FBC8F" opaque="true" name="Decibel Level">

<header text="Decibel Level"

background="#8B6914" opaque="true" visible="true" foreground="#8FBC8F">

<font name="Dialog"

style="Bold" size="16"/>

<etched-border type="Raised"/>

</header>

<footer text="Never Loud Enough?" opaque="false" visible="true" foreground="#8B6914">

<font style="Bold\_Italic"/>

</footer>

<legend type="MultiCol"

visible="true" useEllipsisWhenTruncating="True" itemTextToolTipEnabled="True" >

<multi-col numColumns="2"/>

<legend-column itemTextAlignment="Center" />

<font style="Bold"/>

<layout-hints x="25" y="240"/>

</legend>

<circular-scale direction="Backward" min="0" max="120">

<circular-range name="What?"

background="white" innerExtent="0.25" outerExtent="0.75" startValue="0" stopValue="30"/>

<circular-range name="Optimum"

background="#006400" innerExtent="0.25" outerExtent="0.75" startValue="30" stopValue="60"/>

<circular-range name="Minimize"

background="#FFD700" innerExtent="0.25" outerExtent="0.75" startValue="60" stopValue="85"/>

<circular-range name="Earsplitting"

background="#8B0000" innerExtent="0.25" outerExtent="0.75" startValue="85" stopValue="120"/>

<tick style="Line"/>

</circular-scale>

<center background="red"/>

<needle style="Arrow"

value="35" innerExtent="0.15" outerExtent="0.65" background="#8B6914" interactionType="Drag"/>

</circular-gauge>

</gauge>

* + - 1. Specifying Images with XML

You can add images using the <image-file> element. For example, the following code sets an image file on the center of the gauge defined in the preceding example.

<gauge>

<circular-gauge...>

...

<center background="red">

<image-file fileName[="http://www.mysite.com/centerbgimage.jpg"](http://www.mysite.com/centerbgimage.jpg) fileAccess="Url"

imageScaled="true" isBackground="true"/>

</center>

...

</circular-gauge>

</gauge>

The fileAccess attribute of the <image-file> element is used by the LoadProperties object to determine how to access the image file named in the fileName attribute. The following table summarizes the valid values for the fileAccess property.

|  |  |
| --- | --- |
| **fileAccess Value** | **Description** |
| Default | The default access is ABSOLUTE. |
| Absolute | Interprets the filename as an absolute name. |
| Url | Interprets the filename as a URL. |
| Relative\_Url | Interprets the filename as a URL after adding a prefix to the beginning of it. You specify the prefix by setting the relativeURLPrefix property of the LoadProperties object. For more information, see [Section 11.2.3, “Constructing a](#_bookmark425) [LoadProperties Object,” on page 153](#_bookmark425). |
| Resolving\_Class | Requires a resolving class Class object to load the file. The ClassLoader of the resolving class is used to resolve the name through a call to getResource(*filename*). In the resolution process, if the filename starts with “/”, it is unchanged; otherwise, the package name of the resolving Class is added to the beginning of the filename, after converting “.” to “/”. You specify the resolving class by setting the resolvingClass property of the LoadProperties object. For more information, see [Section 11.2.3,](#_bookmark425) [“Constructing a LoadProperties Object,” on page 15](#_bookmark425)3. |

The imageScaled attribute maps to the imageScaled property of the container. It is a boolean that determines whether or not the image is scaled. The isBackground attribute determines whether the image is in the background or the foreground of the container. In most cases, you set isBackground to true (*default*). When you are using a foreground image (enabled for linear ranges only), you need to set isBackground to false.

##### Constructing a LoadProperties Object

In simple cases (see [Section 11.2.1.1, “Creating Circular and Linear Gauges from XML,”](#_bookmark419) [on page 14](#_bookmark419)9), you can use the no-arguments LoadProperties constructor to create a LoadProperties object that uses null values for all its properties.

In the following circumstances, you may need to specify some properties for your

LoadProperties object:

* + - * When you have an <image-file> tag with fileAccess=Resolving\_Class, you use the resolvingClass property to specify the Class object that is used to resolve the location of file.
      * When you have an <image-file> tag with fileAccess=Relative\_Url, you use the relativeURLPrefix property to specify the String to prepend to the URL. (Recall that in this case the fileName attribute of the <image-file> is interpreted as a URL.)
* When you specify an <external-java-code> tag, you use the userObject property to specify the Object and the storeUserObject property to determine if the Object is stored with the gauge.

The following example shows a LoadProperties constructor with some properties set.

Class myResolvingClass = new Class(...); Object myObject = new myObject(...);

// Create a LoadProperties object and set properties LoadProperties loadProps = new LoadProperties(

myResolvingClass, // resolvingClass

"", // relativeURLPrefix (default is empty String)

myObject, // userObject

true); // storeUserObject

Alternatively, you can set these property using the LoadProperties object’s set\*() methods. For more information, look up com.klg.jclass.util.io.LoadProperties in the *API documentation*. See also the <image-file> and <external-java-code> elements in [Chapter 12, “XML DTDs](#_bookmark454)”.

#### Updating a Gauge or Graph From XML

Each of the factories has an update\*() method that updates an existing gauge or graph from a file, reader, stream, or URL. It is very similar to the make\*() method, except that it takes as its first parameter the name of the gauge or graph to update. If the name is null, the method creates the gauge or graph. The values of any properties not specified in XML remain unchanged when the properties are loaded into the gauge or graph.

A complete example is shown for circular and linear gauges; the indicator gauge and graph sections describe how to modify that example.

##### Updating a Circular or Linear Gauge from XML

The following code updates an existing circular gauge called myGauge with properties contained in a file called *gauge.in.xml*. In the example, the XML file is specified as a String and assigned to inputSource. Alternatively, the inputSource could be an InputStream, a Reader, or a URL.

Note that a LoadProperties object is created in this example. If null is passed to the

updateGauge() method instead, a default LoadProperties object is used.

// Specify where to look for the XML-based gauge properties String inputSource = "gauge.in.xml";

// Create the LoadProperties object LoadProperties loadProps = new LoadProperties();

// Update the existing gauge called myGauge try {

myGauge = JCGaugeFactory.updateGauge(myGauge,

inputSource, loadProps, JCGaugeFactory.XML);

}

catch (JCIOException e) {

System.out.println("Error accessing external file:" + e.getMessage());

}

catch (JCParseException e) {

System.out.println("Error parsing file:" + e.getMessage());

}

catch (IOException e) {

System.out.println("Error reading " + inputSource +

":" + e.getMessage());

}

##### Updating an Indicator Gauge or Panel from XML

Use the same code as in “[Updating a Circular or Linear Gauge from XML](#_bookmark429),” but substitute JCIndicatorGaugeFactory for JCGaugeFactory, and use updateIndicatorGauge() (instead of updateGauge()).

For example:

// Update the existing gauge called myGauge try {

myGauge = JCIndicatorGaugeFactory.updateIndicatorGauge( myGauge, inputSource, loadProps,

JCIndicatorGagueFactory.XML);

}

##### Updating Graphs from XML

Use the same code as in “[Updating a Circular or Linear Gauge from XML](#_bookmark429),” but substitute

JCGraphFactory for JCGaugeFactory, and use updateGraph() (instead of updateGauge()). For example:

// Update the existing graph called myGraph

try {

myGraph = JCGraphFactory.updateGraph(myGraph, inputSource, loadProps, JCGraphFactory.XML);

}

#### Saving a Gauge or Graph to XML

Each of the factories has a save\*() method that saves a gauge or graph to a file, stream, or writer. If the specified gauge or graph does not exist, there is no XML output. Gauge properties are saved to corresponding XML elements. Images are not saved, but information about how to reload them is saved. For more information, see [Section 11.5,](#_bookmark438) [“Preparing to Save Information About Images to XML,” on page 15](#_bookmark438)7.

This section shows you how to use the save\*() method to save gauge or graph to an XML file. A complete example is shown for circular and linear gauges; the indicator gauge and graph sections describe how to modify that example.

##### Saving a Circular or Linear Gauge to XML

In the following example, the properties that define myGauge are saved to an XML file. The file name *gauge.out.xml* is specified as a String and assigned to outputTarget.

Alternatively, outputTarget could also be an OutputStream or a Writer.

// Specify where to save the gauge properties as XML String outputTarget = "gauge.out.xml";

// Save the gauge to a file try {

JCGaugeFactory.saveGauge(myGauge, outputTarget,

JCGaugeFactory.XML);

}

catch (IOException e) {

System.out.println("Error writing to " + outputTarget + ":" +

e.getMessage());

}

##### Saving an Indicator Gauge or Panel to XML

Use the same code as in “[Saving a Circular or Linear Gauge to XML](#_bookmark435),” but substitute JCIndicatorGaugeFactory (for JCGaugeFactory), and use saveIndicatorGauge() (instead of saveGauge()).

For example:

// Save the gauge to a file try {

JCIndicatorGaugeFactory.saveIndicatorGauge(myGauge,

outputTarget, JCIndicatorGaugeFactory.XML);

}

##### Saving Graphs to XML

Use the same code as in “[Saving a Circular or Linear Gauge to XML](#_bookmark435),” but substitute

JCGraphFactory (for JCGaugeFactory), and use saveGraph() (instead of saveGauge()).

For example:

// Save the graph to a file try {

JCGraphFactory.saveGraph(myGraph, outputTarget,

JCGraphFactory.XML);

}

#### Preparing to Save Information About Images to XML

Images are not saved when a gauge or graph is saved to XML. You can, however, choose to save information about the images in the XML so that if you reload the gauge or graph the images can be located and displayed. Image information includes the file name, where the image can be found, and how to access it.

If your gauge or graph contains images, and you want to be able to save image information, you need to set the imageOutputProperties property on each container that has an image. The imageOutputProperties property takes an OutputProperties object. The following sections describe how to create and use an OutputProperties object.

##### Constructing an OutputProperties Object

Every image in your gauge or graph whose information you want to save to XML requires an OutputProperties object. The following code creates an OutputProperties object for use with the image in the background of the gauge.

// Create an instance of OutputProperties OutputProperties imageOutputProps = new OutputProperties(

null, // outputFileName (not used) "images/gaugebgimage.jpg", // propertyName

null, // saveType (not used) Properties.RELATIVE\_URL); // fileAccess

The propertyName property specifies the file name and location of the image. In the XML, the value of propertyName is saved to the <image-file> fileName attribute. The fileAccess property specifies how to interpret the propertyName. It is saved to the

<image-file> fileAccess attribute.

The outputFileName and saveType properties are null, because this class has other uses and these properties are not required for images. For more information, see [Section](#_bookmark423) [11.2.2.2, “Specifying Images with XML,” on page 15](#_bookmark423)2.

For more information, look up the com.klg.jclass.util.io.OutputProperties class in the *API Documentation*.

##### Setting Output Properties on a Background Image

Background images are enabled for gauge containers and some gauge components, as well as the graph container. To prepare to save information about a background image to XML, you need to set the imageOutputProperties property of the container and specify its OutputProperties object.

// Define an image

String URLString = “http://www.my\_site.com/snowflakes.jpg”; URL url = new URL(URLString);

// Load the image (where loadImageFromURL is some method

// that creates an image from a URL) Image inputImage = loadImageFromURL(url);

// Set the image on the gauge container myGauge.setImage(inputImage); myGauge.setImageScaled(true);

// Set the output properties for the image OutputProperties imageOutputProps = new OutputProperties(

null, URLString, null, Properties.URL); myGauge.setImageOutputProperties(imageOutputProps);

##### Setting Output Properties on a Foreground Image

Foreground images are enabled for ranges in linear gauges. To prepare to save information about a foreground image to XML, you need to set the foregroundImageOutputProperties property of the range and specify its OutputProperties object.

For example:

// Define a foreground image in a linear range String URLPrefix = “file:///c:/jclass”;

String imageString = "images/rangefgimage.jpg";

// Load the image (where loadImageFromURL is some method

// that creates an image from a URL)

Image inputFgImage = loadImageFromURL(URLPrefix + imageString);

// Create an instance of OutputProperties

OutputProperties fgImageOutputProps = new OutputProperties( null, imageString, null, Properties.RELATIVE\_URL);

// Set the image and its properties on the range container range1.setForegroundImage(inputFgImage); range1.setForegroundImageOutputProperties(fgImageOutputProps);

When loading the image from XML, set the above relative URL prefix in the

LoadProperties object.

##### How Image Information is Stored and Accessed

Recall that images are stored as PortableImage objects (see [Section 1.4.2, “Images and](#_bookmark37) [the PortableImage Object,” on page 25](#_bookmark37)). For example, the image and its properties specified in [Section 11.5.2, “Setting Output Properties on a Background Image,” on page](#_bookmark442) [15](#_bookmark442)8, maps to the following PortableImage object definition:

PortableImage image = new PortableImage( inputImage, // image

true, // imageScaled imageOutputProps); // imageOutputProperties

Hence, the PortableImage has as one of its properties the name of the OutputProperties object for the image. When a gauge or graph containing an image is saved to XML, the information about the image that is contained in the OutputProperties object is stored in an <image-file> element. An <image-file> element is nested within its container’s tag.

For example, a background <image-file> element for a circular gauge would be inside the <circular-gauge> tag.

<gauge>

...

<circular-gauge...>

<image-file fileName="images/gaugebgimage.jpg" fileAccess="Relative\_Url" imageScaled="true" isBackground="true"/>

...

</circular-gauge>

</gauge>

The attributes in the <image-file> tag map to properties as follows:

* + - * fileName maps to the propertyName property of the OutputProperties object. It is the name of the image file.
      * fileAccess maps to fileAccess property of the OutputProperties object. It used when creating a gauge from the saved XML. For more information, see [Section 11.2.2.2, “Specifying Images with XML,” on page 15](#_bookmark423)2.
      * imageScaled maps to the imageScaled property of the gauge container. It is a

boolean that records whether or not the image is scaled.

* + - * isBackground is set automatically and should not be changed. When the image property of the container is set, isBackground is true (*default*), which means the image is in the background. When the foregroundImage property is set (enabled for linear ranges only), isBackground is automatically set to false, which means the image is in the foreground.

Recall that if an image does not have an OutputProperties object associated with it, the image is ignored.

#### Internationalizing Your XML-based Gauge

If you need to offer your XML-based gauge in multiple languages, you can replace any or all of the text strings with variables and provide a resource bundle containing properties files or ResourceBundle classes for each language that you support. The following sections describe how to add variables to your gauge and create the resource bundle.

##### Using Variables

Wherever text appears on your gauge, you can replace the text string with a variable in your XML file. For example, you can use variables for the header, footer, tick label, and range labels. Variables take the form ${KEY}, where KEY is a unique variable name.

Variable names are case-sensitive and can be uppercase, lowercase, or mixed case. Try to use meaningful names so that, when you create your resource bundle, it is easier to map the correct text strings to the variables.

For example, in the following excerpt taken from *linear-gauge.xml* (located in *JCLASS\_HOME/examples/gauge/xml/*) variables replace text strings for the header text (${Header\_Text}), gauge title (${Gauge\_Title}), and range names (${Range1\_Text},

${Range2\_Text}).

<?xml version="1.0"?>

<!DOCTYPE gauge SYSTEM "JCGauge.dtd">

<gauge>

<linear-gauge...>

<header text="${Header\_Text}"

visible="true">

<font size="16"/>

<etched-border type="Raised" />

</header>

...

<gauge-area>

<compound-border>

<titled-border title="${Gauge\_Title}"/>

<empty-border>

<insets bottom="5"/>

</empty-border>

</compound-border>

</gauge-area>

...

<linear-scale,,,>

<linear-range name="${Range1\_Text}"

background="255-180-180" innerExtent="-0.02" outerExtent="1.02" startValue="90" stopValue="119"/>

<linear-range name="${Range2\_Text}"

background="173-216-255" innerExtent="-0.02" outerExtent="1.02" startValue="60" stopValue="79"/>

...

</linear-scale>

...

</linear-gauge>

</gauge>

Note: If you want, you can embed a variable within a text string. For example, you could specify a value such as “This is a ${KEY}”. Mixing text and variables, however, is not generally recommended; you usually want to provide gauges with the text entirely in your client’s language.

##### Creating a Resource Bundle

Depending on your needs, you can use the class ResourceBundle or either of its subclasses – PropertyResourceBundle or ListResourceBundle – to assign text strings to the variables that you used in your XML file. PropertyResourceBundle looks for the localized strings in properties files, while ResourceBundle and ListResourceBundle looks for them in your code. For more information on how to use these classes, locate java.lang.ResourceBundle in the *Java API documentation*.

Whichever method you choose, your resource bundle must include a default locale – the language used when a locale is not specified by the browser – plus a properties file or ResourceBundle class for each of the other languages that you want to support. The default locale uses the base name of your resource bundle, while all other locales should follow the I18N naming conventions for language and country, though the country code is optional if there is no chance of confusion. For example, your base name and default locale could be called *myresources*, while U.S. English would be *myresources\_en\_US*, and German would be *myresources\_de\_DE* (or *myresources\_de*). For more information, see [Section 1.6, “Internationalization,” on page 31](#_bookmark48).

This section assumes that you are already familiar with how to create a properties file. The following examples show a ListResourceBundle implementation that is used by the

*LinearGaugeXMLExample* discussed in the preceding section. The base name is

XMLLocaleInfo. The first example is from *XMLLocaleInfo.java* (located in *JCLASS\_HOME/examples/gauge/xml/resources*) and contains text strings in English US. The second example is from *XMLLocaleInfo\_es.java* and contains text strings in Spanish.

Example: *XMLLocaleInfo.java* with English US text strings

package examples.gauge.xml.resources; import java.util.ListResourceBundle;

/\*\*

\* XMLLocaleInfo

\*/

public class XMLLocaleInfo extends ListResourceBundle

{

protected Object[][] contents = {

{"Header\_Text", "Blood Pressure"},

{"Gauge\_Title", "Your Reading"},

{"Range1\_Text", "Normal Systolic Range"},

{"Range2\_Text", "Normal Diastolic Range"},

};

/\*\*

\* See class description.

\*/

protected Object[][] getContents()

{

return contents;

}

}

Example: *XMLLocaleInfo\_es.java* with Spanish text strings

package examples.gauge.xml.resources; import java.util.ListResourceBundle;

/\*\*

\* XMLLocaleInfo\_es

\*/

public class XMLLocaleInfo\_es extends ListResourceBundle

{

protected Object[][] contents = {

{"Header\_Text", "Presi\u00f3n sangu\u00ednea"},

{"Gauge\_Title", "Su lectura"},

{"Range1\_Text", "Rango sist\u00f3lico normal"},

{"Range2\_Text", "Rango diast\u00f3lico normal"},

};

/\*\*

\* See class description.

\*/

protected Object[][] getContents()

{

return contents;

}

}

##### Using Resource Bundles

To load a localized gauge, you need to implement a locale handler using the com.klg.jclass.util.LocaleHandler interface. You can use the implementation provided with JClass Gauge, called LocaleBundle, or you can create your own locale handle by implementing the LocaleHandler interface.

To use LocaleBundle, you create an instance of LocaleBundle and specify the fully qualified pathname to the base name of your resource bundle and a Locale object for the locale you want to use. To tell the factory how to find your resource bundle, you set the localeHandler property of your LoadProperties object to point to your LocaleBundle object.

For example, in the following condensed version of *LinearGaugeXMLExample.java* (located in *JCLASS\_HOME/examples/gauge/xml/*), the programmer creates a LocaleBundle object called localeBundle and specifies the location of the bundle and a Locale object (which is defined elsewhere and passed as a parameter to the createGauge() method). localeBundle is then passed to the setLocaleHandler() method for the loadProperties object. Finally, the loadProperties object is passed to the factory’s makeGauge() method.

package examples.gauge.xml;

import com.klg.jclass.gauge.JCGaugeFactory; import com.klg.jclass.gauge.JCGauge;

...

import com.klg.jclass.util.LocaleBundle;

...

import java.util.Locale;

public class LinearGaugeXMLExample extends JPanel implements ActionListener

{

...

public JCGauge createGauge(Locale locale)

{

JCGauge newGauge = null;

// Create the gauge try {

LoadProperties loadProperties = new LoadProperties(); LocaleBundle localeBundle = new LocaleBundle(

"examples.gauge.xml.resources.XMLLocaleInfo", locale); loadProperties.setLocaleHandler(localeBundle);

URL url = getClass().getResource("linear-gauge.xml"); newGauge = JCGaugeFactory.makeGauge(

url, loadProperties, JCGaugeFactory.XML);

}

catch (IOException e) { e.printStackTrace();

}

...

}

}

return newGauge;

# 12

## XML DTDs

[*IndicatorGauge DTD*](#_bookmark456)■ [*JCGauge DTD*](#_bookmark468)

JClass Gauge has two DTDs: *IndicatorGauge.dtd* and *JCGauge.dtd*.

* + - * *IndicatorGauge.dtd* contains elements specific to indicator gauges and indicator gauge panels. It requires *JCGauge.dtd*.
      * *JCGauge.dtd* is the main DTD. It contains elements common to all gauges, as well as elements specific to circular gauges, linear gauges, and the JCGraph component.

#### 12.1 IndicatorGauge DTD

The *IndicatorGauge.dtd* contains two primary elements, called indicator-gauge and indicator-gauge-panel, plus a number of subelements. The primary elements are listed first. The rest of the elements are in alphabetical order.

Note: Some elements use subelements from *JCGauge.dtd*, namely the font, image-file, and the border-type subelements. For details, see [Section 12.2, “JCGauge DTD,” on page](#_bookmark468) [171](#_bookmark468).

##### indicator-gauge

Purpose:

Creates a standalone indicator gauge.

Equivalent in JClass Gauge:

JCIndicatorGauge

Sub-Elements: ■ [icon](#_bookmark464)

* + - * [indicator-range](#_bookmark466)
      * [image-file](#_bookmark500)
      * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [external-java-code](#_bookmark488)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background. | Color |
| baseValue | Specifies the base value, which is the lowest value that the gauge can display. | int |
| blinkInterval | Specifies how long to wait between blinks. | int |
| height | Determines the height of the gauge. | int |
| name | Specifies a name for the gauge to be used by the application. | String |
| opaque | Determines whether this component is opaque or transparent. Default is true. | boolean |
| padding | Specifies the padding in pixels. Creates a border-effect around the icon. | int |
| textValueDisplayed | Determines whether or not the value is displayed as text on the gauge. Default is false. | boolean |
| value | Specifies the value that the gauge represents. | int |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| visible | Determines whether the gauge is displayed or hidden. Default is true. | boolean |
| width | Determines the width of the gauge. | int |

##### indicator-gauge-panel

Purpose:

Creates a panel that contains copies of the specified [base-gauge](#_bookmark462).

Equivalent in JClass Gauge:

JCIndicatorGaugePanel

Sub-Elements: ■ [base-gauge](#_bookmark462)

* + - * [indicator-range](#_bookmark466)
      * [image-file](#_bookmark500)
      * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [external-java-code](#_bookmark488)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background. | Color |
| baseValue | Specifies the base value, which is used as the lowest value that the panel can display. | int |
| blinkInterval | Specifies how long to wait between blinks. | int |
| direction | Determines whether gauge values increase in a forward direction (right-to-left in a horizontal panel and bottom-to-top in a vertical panel), or in the reverse direction. Default is Forward. | * Forward * Backward |
| height | Determines the height of the panel. | int |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| name | Specifies a name for the panel to be used by the application. | String |
| numGauges | Specifies the number of copies of the gauge to display in the panel. | int |
| opaque | Determines whether this component is opaque or transparent. Default is true. | boolean |
| orientation | Determines whether the panel is oriented horizontally or vertically. Default is Horizontal. | * Horizontal * Vertical |
| padding | Specifies the padding in pixels. Creates a border-effect around the gauges. | int |
| textValueDisplayed | Determines whether or not the value is displayed as text on the gauge. Default is false. | boolean |
| value | Specifies the value that the panel of gauges represents. | int |
| valueType | Determines how the value is represented in the panel. It can be in binary format, decimal, or shown as magnitude on a scale using a single color or multiple colors.  Default is Binary. | * Binary * Decimal * Multichrome * Unichrome |
| visible | Determines whether the panel is displayed or hidden. | boolean |
| width | Determines the width of the panel. | int |

##### base-gauge

Purpose:

Creates an indicator gauge to be replicated in an [indicator-gauge-panel](#_bookmark460).

Equivalent in JClass Gauge:

JCBaseIndicatorGauge

Sub-Elements: ■ [icon](#_bookmark464)

* + - * [image-file](#_bookmark500)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background. | Color |
| opaque | Determines whether this component is opaque or transparent. Default is true. | boolean |
| padding | Specifies the padding in pixels. Creates a border-effect around the icon. | int |
| visible | Determines whether the gauge is displayed or hidden. Default is true. | boolean |

##### icon

Purpose:

Creates an icon to go on the gauge.

Equivalent in JClass Gauge:

JCIndicatorIcon

Sub-Elements: [font](#_bookmark490)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| foreground | Specifies the text color. | Color |
| shape | Determines the shape of the icon. Default is  Rectangle. | * Rectangle * Oval * Triangle |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| imageScaled | Determines whether or not the icon’s background image is scaled. Used only if the current indicator-range element has displayType=Url. Otherwise, the image’s imageScaled property is used. | boolean |

##### indicator-range

Purpose:

Specifies a attributes for each range.

Equivalent in JClass Gauge:

JCIndicatorGauge, JCIndicatorGaugePanel

Sub-Elements: [image-file](#_bookmark500)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| displayType | Determines what appears in the background of the range. If set to Image, a background image is loaded from the image-file element. Otherwise, this attribute determines how the displayValue attribute should be interpreted. No default; this attribute must be defined. | * Color * Url * Image |
| displayValue | Specifies a Color or URL object to use in the background of a range, depending on the value of displayType. (This attribute is ignored when displayType=Image.) | Color or URL |
| text | Specifies the text for a range. | String |
| toolTip | Specifies the tooltip text for a range. | String |
| value | Specifies the upper value of the indicator range. No default; this attribute must be defined. The lower value is determined by the upper value of the previous range, or if there is no previous range, the baseValue. | int |

#### JCGauge DTD

The circular and linear gauges and the JCGraph component share the same DTD, called *JCGauge.dtd*. Its elements are described in this section. The primary element, gauge, is listed first. The remaining elements are listed in alphabetical order.

##### gauge

Purpose:

The main gauge element. It is a wrapper that expects a gauge-type subelement.

Equivalent in JClass Gauge:

none

Sub-Elements: ■ [circular-gauge](#_bookmark475)

* + - * [linear-gauge](#_bookmark520)
      * [graph](#_bookmark496)
      * [indicator-gauge](#_bookmark458) (from *IndicatorGauge.dtd*)
      * [indicator-gauge-panel](#_bookmark460) (from *IndicatorGauge.dtd*)

##### bevel-border

Purpose:

Bevel or SoftBevel borders.

Equivalent in JClass Gauge:

javax.swing.border.BevelBorder or

javax.swing.border.SoftBevelBorder

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| highlightColor | Specifies the color to use for the bevel highlight. | Color |
| shadowColor | Specifies the color to use for the bevel shadow. | Color |
| soft | Determines if the component uses a soft bevel border or a standard bevel border. Default is false. | boolean |
| type | Specifies the bevel type. | * Raised * Lowered |

##### center

Purpose:

Specify the center of a circular gauge.

Equivalent in JClass Gauge:

JCCenter

Sub-Elements: [image-file](#_bookmark500)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Determines the color of the center area. | Color |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| radius | Specifies the center area radius, expressed as a ratio based on the scale radius. | double |
| visible | Determines whether this component is displayed or hidden. Default is true. | boolean |

##### circular-gauge

Purpose:

Creates a circular gauge. The properties and sub-elements by and large coincide with the properties and sub-objects of the corresponding object within the gauge component.

Equivalent in JClass Gauge:

JCCircularGauge

Sub-Elements: ■ [font](#_bookmark490)

* + - * [header](#_bookmark498)
      * [footer](#_bookmark492)
      * [legend](#_bookmark510)
      * [gauge-area](#_bookmark494)
      * [image-file](#_bookmark500)
      * [needle](#_bookmark530)
      * [indicator](#_bookmark502)
      * [label](#_bookmark506)
      * [center](#_bookmark473)
      * [circular-scale](#_bookmark479)
      * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [external-java-code](#_bookmark488)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background. | Color |
| foreground | Specifies the color used in the foreground. | Color |
| gaugeType | Specifies if the entire circular gauge (Full\_Circle) is to be displayed or the specified subsection. | * Full\_Circle * Top-Half * Bottom\_Half * Left\_Half |
|  |  | * Right\_Half |
|  |  | * Upper\_Right\_Quarter |
|  |  | * Upper\_Left\_Quarter |
|  |  | * Lower\_Left\_Quarter |
|  |  | * Lower\_Right\_Quarter |
| height | Determines the height of the gauge. | int |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| name | Specifies a name for the gauge to be used by the application. | String |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| visible | Determines whether the gauge is displayed or hidden. | boolean |
| width | Determines the width of the gauge. | int |

##### circular-range

Purpose:

Specify ranges in a circular scale.

Equivalent in JClass Gauge:

JCRange, JCCircularRange

Sub-Elements: [image-file](#_bookmark500)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Determines the color of the range. | Color |
| innerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to begin drawing the range. For details, see [Section 1.5.2, “Sizing Components Using](#_bookmark46) [Extent Parameters,” on page 29](#_bookmark46). | double |
| name | Specifies a name for the range. | String |
| outerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to stop drawing the range. For details, see [Section 1.5.2, “Sizing Components Using Extent](#_bookmark46) [Parameters,” on page 29](#_bookmark46). | double |
| startValue | Specifies the value from the associated scale at which to begin the range. | double |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| stopValue | Specifies the value from the associated scale at which to end the range. | double |
| visible | Determines whether this component is displayed or hidden. Default is true. | boolean |

##### circular-scale

Purpose:

Defines the scale used in a circular gauge.

Equivalent in JClass Gauge:

JCCircularScale

Sub-Elements: ■ [font](#_bookmark490)

* + - * [tick](#_bookmark534)
      * [circular-range](#_bookmark477)
      * [image-file](#_bookmark500)
      * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background. | Color |
| direction | Determines the direction in which tick values increment on the scale. | * Forward * Backward * Clockwise * Counter\_ Clockwise |
| max | Specifies the maximum value of the scale. | double |
| min | Specifies the minimum value of the scale. | double |
| opaque | Determines whether this component is opaque or transparent. Default is true. | boolean |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| paintComplete Background | Determines whether the entire area defined by the gaugeType attribute in the circular- gauge element is painted (true) or only the sweep defined by the scale (false). Default is false. | boolean |
| startAngle | Specifies the angle at which to begin drawing the scale. | double |
| stopAngle | Specifies the angle at which to end the scale. | double |
| zoomFactor | Specifies the zoom factor as a percentage of the radius of the scale, expressed in decimal form. The scale is compressed using the formula (zoomFactor\*radius). For details, see [Section 5.4.2, “Setting Start and Stop](#_bookmark198) [Angles,” on page 91](#_bookmark198). | double |

##### compound-border

Purpose:

Specifies an outside border and an inside border. Note that these subborders can also be of the type compound-border.

Equivalent in JClass Gauge:

javax.swing.border.CompoundBorder

Sub-Elements: ■ [empty-border](#_bookmark484)

* + - * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)

Attributes: none

##### dash-array

Purpose:

Specifies an array of values used to create a dashed line.

Equivalent in JClass Gauge:

none

Sub-Elements: [value](#_bookmark538)

Attributes: none

##### empty-border

Purpose:

Empty borders.

Equivalent in JClass Gauge:

javax.swing.border.EmptyBorder

Sub-Elements: [insets](#_bookmark504)

Attributes: none

##### etched-border

Purpose:

Etched borders.

Equivalent in JClass Gauge:

javax.swing.border.EtchedBorder

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| highlightColor | Specifies the color to use for the etched highlight. | Color |
| shadowColor | Specifies the color to use to the etched shadow. | Color |
| type | Specifies the type of etch. | * Raised * Lowered |

##### external-java-code

Purpose:

Specifies a Java class that will be created and called between the creation of a gauge via XML and the return of control to the calling code. This can be used for encapsulating gauge settings that cannot be set with XML.

The Java class must contain an empty constructor, as well as implement the com.klg.jclass.util.property.xml.ExternalCodeHandler interface. The ExternalCodeHandler interface specifies a method named handle() that is called by the JClass Gauge XML parser. The contents of the body of the tag will be passed to the handle() method in the userData parameter. The value of the UserObject property in the current LoadProperties class will be passed to the handle() method when it is called. For more information on LoadProperties, see [Section 11.1.3, “LoadProperties](#_bookmark412) [Class,” on page 148](#_bookmark412). For an example of how to use this element, see *ExternalJavaXMLExample.java* in *JCLASS\_HOME/examples/gauge/xml.*

Note: When a gauge that was created via XML is later saved to XML, the contents of this tag are not written out.

Equivalent in JClass Gauge:

n/a

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| class | Fully qualified name of the class to be created and called by the JClass Gauge XML parser. | String |

##### font

Purpose:

Creates a font object for use in headers, footers, legends, labels, and any other element that contain text.

Equivalent in JClass Gauge:

java.awt.Font

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| name | Specifies the name of the font. | String |
| style | Specifies the style of the font. | * Bold * Italic * Bold\_Italic * Plain |
| size | Specifies the point size of the font. | int |

##### footer

Purpose:

The component used as the footer for the gauge.

Equivalent in JClass Gauge:

JLabel

Sub-Elements: ■ [font](#_bookmark490)

* + - * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [layout-hints](#_bookmark508)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background | Color |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| foreground | Specifies the color used in the foreground. | Color |
| horizontalAlignment | Determines the horizontal alignment of the text in the label. Default is Leading. | * Left * Center * Right * Leading * Trailing |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| text | Specifies the text to be displayed. No default; this attribute must be defined. | String |
| verticalAlignment | Determines the vertical alignment of the text in the label. Default is Center. | * Top * Center * Bottom |
| visible | Determines whether this component is displayed or hidden. | boolean |

##### gauge-area

Purpose:

The area that contains the scale. Note that this does not include the header, footer, and legend.

Equivalent in JClass Gauge:

JCGaugeArea

Sub-Elements: ■ [font](#_bookmark490)

* + - * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [layout-hints](#_bookmark508)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background | Color |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| foreground | Specifies the color used in the foreground. | Color |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| visible | Determines whether this component is displayed or hidden. | boolean |

##### graph

Purpose:

A light-weight graph component.

Equivalent in JClass Gauge:

JCGraph

Sub-Elements: ■ [values](#_bookmark540)

* + - * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [image-file](#_bookmark500)
      * [line-style](#_bookmark516)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background | Color |
| direction | The direction in which the graph travels. Default is Right. | * Left * Right * Up * Down |
| height | Determines the height of the graph. If not specified, the height remains unchanged. | int |
| lineColor | The color of the graph line. | Color |
| maxValue | The initial maximum value used for the graph’s scale. | int |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| minValue | The initial minimum value used for the graph’s scale. | int |
| name | The name of the graph component. | String |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| pixelsPerValue | The number of pixels used to mark a value point on the graph. | int |
| visible | Determines whether this component is displayed or hidden. | boolean |
| width | Determines the width of the graph. If not specified, the width remains unchanged. | int |

##### header

Purpose:

The component used as the header for the gauge.

Equivalent in JClass Gauge:

JLabel

Sub-Elements: ■ [font](#_bookmark490)

* + - * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [layout-hints](#_bookmark508)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background | Color |
| foreground | Specifies the color used in the foreground. | Color |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| horizontalAlignment | Determines the horizontal alignment of the text in the label. Default is Leading. | * Left * Center * Right * Leading * Trailing |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| text | Specifies the text to be displayed. No default; this attribute must be defined. | String |
| verticalAlignment | Determines the vertical alignment of the text in the label. Default is Center. | * Top * Center * Bottom |
| visible | Determines whether this component is displayed (true) or hidden (false). | boolean |

##### image-file

Purpose:

Specifies an image from an external source.

Equivalent in JClass Gauge:

none

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| fileName | Specifies the image file. No default; this attribute must be defined. | String |
| fileAccess | Determines how to interpret the fileName property. For more information, see [Section](#_bookmark424) [11.2.2.2, “Specifying Images with XML,” on page](#_bookmark424)  [15](#_bookmark424)2. Default is Default.  Note: The Servlet value is not available in JClass Gauge. | * Default * Absolute * Resolving\_Class * Url * Relative\_Url * Servlet |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| isBackground | Determines whether or not the image is in the background. When false, the image is in the foreground. Default is true. | boolean |
| imageScaled | Determines whether the image is scaled (true) to fit the size of its parent or unscaled (false).  Default is true. | boolean |

##### indicator

Purpose:

Specify the indicator to use with the scale. Note: If you want to enable interaction, use the needle element instead.

Equivalent in JClass Gauge:

JCIndicator, JCCircularIndicator, JCLinearIndicator

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| color | Determines the color of the indicator. (Formerly called foreground.) | Color |
| innerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to begin drawing the indicator. The percentage is applied differently for circular versus linear gauges. For details, see [Section](#_bookmark46) [1.5.2, “Sizing Components Using Extent](#_bookmark46) [Parameters,” on page 29](#_bookmark46). | double |
| name | Specifies a name for the indicator to be used by the application. | String |
| outerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to stop drawing the indicator. The percentage is applied differently for circular versus linear gauges. For details, see [Section](#_bookmark46) [1.5.2, “Sizing Components Using Extent](#_bookmark46) [Parameters,” on page 29](#_bookmark46). | double |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| reversed | Determines whether or not the direction of the indicator is reversed. Default is false.  If reversed=true in a circular gauge, the indicator is drawn pointing toward the center. For linear gauges, depending on the orientation, the indicator points left instead of right or up instead of down. | boolean |
| style | Specifies the style of the indicator. No default; this attribute must be defined. | * Rectangle * Circle * Pointer |
|  |  | * Tailed\_Pointer |
|  |  | * Arrow |
|  |  | * Tailed\_Arrow |
|  |  | * Triangle |
| value | Specifies a value on the scale. The indicator will point to this value. | double |
| width | Specifies the width of the indicator. | double |

##### insets

Purpose:

A representation of the borders of a container used by the [empty-border](#_bookmark484) and

[matte-border](#_bookmark526) elements.

Equivalent in JClass Gauge:

java.awt.Insets

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| bottom | Specifies the bottom margin. | int |
| left | Specifies the left margin. | int |
| right | Specifies the right margin. | int |
| top | Specifies the top margin. | int |

##### label

Purpose:

Specify a label. If the label is on a circular gauge, use a [radial-constraint](#_bookmark532). If the label is on a linear gauge, use a [linear-constraint](#_bookmark518).

Equivalent in JClass Gauge:

JLabel

Sub-Elements: ■ [font](#_bookmark490)

* + - * [linear-constraint](#_bookmark518)
      * [radial-constraint](#_bookmark532)
      * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background | Color |
| foreground | Specifies the color used in the foreground. | Color |
| horizontalAlignment | Determines the horizontal alignment of the text in the label. Default is Leading. | * Left * Center * Right * Leading * Trailing |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| text | Specifies the text to be displayed. No default; this attribute must be defined. | String |
| verticalAlignment | Determines the vertical alignment of the text in the label. Default is Center. | * Top * Center * Bottom |
| visible | Determines whether this component is displayed or hidden. | boolean |

##### layout-hints

Purpose:

Specifies where and at what size a subcomponent ([footer](#_bookmark492), [gauge-area](#_bookmark494), [header](#_bookmark498), or [legend](#_bookmark510)) is drawn on the gauge. Use this subelement to override some or all of the default values assigned when a subcomponent is created. If any of the attributes for this element are left unspecified, the gauge calculates a default value.

Equivalent in JClass Gauge:

java.awt.Rectangle

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| height | Specifies the height of the subcomponent. | int |
| width | Specifies the width of the subcomponent. | int |
| x | Specifies the x position of the subcomponent. | int |
| y | Specifies the y position of the subcomponent. | int |

##### legend

Purpose:

Creates a component within the gauge to hold the legend.

Equivalent in JClass Gauge:

com.klg.jclass.util.legend.JCLegend

Sub-Elements: ■ [font](#_bookmark490)

* + - * [legend-column](#_bookmark512)
      * [multi-col](#_bookmark528)
      * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [layout-hints](#_bookmark508)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| anchor | Determines the position of the legend relative to the gauge. | * North * South * East * West * Northeast * Northwest * Southeast * Southwest |
| background | Specifies the color used in the background | Color |
| foreground | Specifies the color used in the foreground. | Color |
| itemTextToolTip Enabled | Determines whether or not tooltips are displayed when the mouse hovers over a legend item. This is useful when the legend text has been truncated. Default is false. | boolean |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| orientation | Determines how legend information is laid out. Default is Vertical. | * Horizontal * Vertical |
| type | Determines the legend type. Default is  Grid. | * Grid * MultiCol |
| useEllipsisWhen Truncating | Determines whether or not an ellipsis is used to indicate truncated legend text. Default is True. | boolean |
| visible | Determines whether this component is displayed or hidden. Default is True. | boolean |

##### legend-column

Purpose:

Defines column attributes for the legend defined by the legend tag.

Equivalent in JClass Gauge:

com.klg.jclass.util.legend.LegendColumn

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| column | Specifies a column within the legend to which the other attributes in this element are applied. If omitted, the other legend- column attributes apply to all columns in the legend. | int |
| itemTextAlignment | Determines the alignment for the text in a column. Default is Leading. | * Left * Center * Right |
|  |  | * Leading |
|  |  | * Trailing |
| maxItemTextWidth | Specifies the maximum width of the column in pixels. If the column text exceeds this width, the text is truncated. | int |
| truncateMode | Determines how text is truncated when the | * Left * Right * Middle * End * Leading * Trailing |
|  | length of the text exceeds the maximum |
|  | width of the column. Default is Trailing. |
|  | For more information, see [“Handling](#_bookmark334) |
|  | [Truncated Text” on page 12](#_bookmark334)6. |

##### line-border

Purpose:

Line borders.

Equivalent in JClass Gauge:

javax.swing.border.LineBorder

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| color | Specifies the color of the border. No default; this attribute must be defined. | Color |
| roundedCorners | Determines whether the border corners are rounded or straight. | boolean |
| thickness | Specifies how thick the border will be drawn. | int |

##### line-style

Purpose:

Line borders.

Equivalent in JClass Gauge:

java.awt.BasicStroke

Sub-Elements: [dash-array](#_bookmark482)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| color | Specifies the color of the border. | Color |
| width | Specifies the width of the line in pixels. | int |
| join | Determines the line join style to use. | * Miter * Bevel * Round |
| cap | Determines the line cap style to use. | * Butt * Round * Square |

##### linear-constraint

Purpose:

Specify the position of a component (usually a [label](#_bookmark506)) on a linear gauge. The center of the component is placed at the position specified.

Equivalent in JClass Gauge:

LinearConstraint

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| extent | Specifies the proportional distance from the top left of the rectangle enclosing the gauge. The distance is vertical for horizontal scales and horizontal for vertical scales, and is specified as a ratio of the distance to the height or width of the scale. No default; this attribute must be defined. | double |
| position | Specifies the distance as a percentage of the height from the top of a vertical scale or the width from the left of a horizontal scale. No default; this attribute must be defined. | int |

##### linear-gauge

Purpose:

Creates a linear gauge. The properties and sub-elements by and large coincide with the properties and sub-objects of the corresponding object within the gauge component.

Equivalent in JClass Gauge:

JCLinearGauge

Sub-Elements: ■ [font](#_bookmark490)

* + - * [header](#_bookmark498)
      * [footer](#_bookmark492)
      * [legend](#_bookmark510)
      * [gauge-area](#_bookmark494)
      * [image-file](#_bookmark500)
      * [needle](#_bookmark530)
      * [indicator](#_bookmark502)
      * [label](#_bookmark506)
      * [linear-scale](#_bookmark524)
      * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)
      * [external-java-code](#_bookmark488)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background. | Color |
| foreground | Specifies the color used in the foreground. | Color |
| height | Determines the height of the gauge. If not specified, the height remains unchanged. | int |
| name | Specifies a name for the gauge to be used by the application. | String |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| visible | Determines whether the gauge is displayed or hidden. | boolean |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| width | Determines the width of the gauge. If not specified, the width remains unchanged. | int |

##### linear-range

Purpose:

Specify ranges in a linear scale. If using two images, you need to set one to be a background image and the other to be a foreground image.

Equivalent in JClass Gauge:

JCRange, JCLinearRange

Sub-Elements: ■ [image-file](#_bookmark500) (with isBackground=true)

* + - * [image-file](#_bookmark500) (with isBackground=false)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Determines the background color of the range. | Color |
| innerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to begin drawing the range. For details, see [Section 1.5.2, “Sizing](#_bookmark46) [Components Using Extent Parameters,” on](#_bookmark46) [page 29](#_bookmark46). | double |
| foreground | Determines the foreground color of the range. | Color |
| foregroundCoverage | Specifies the percentage of the background that is covered by the foreground. The percentage is expressed as a decimal fraction. | double |
| maskOrientation | Determines if the foreground covers the background from left to right (Horizontal) or top to bottom (Vertical). Default is Horizontal. | * Horizontal * Vertical |
| name | Specifies a name for the range. | String |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| outerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to stop drawing the range. For details, see [Section 1.5.2, “Sizing](#_bookmark46) [Components Using Extent Parameters,” on](#_bookmark46) [page 29](#_bookmark46). | double |
| startValue | Specifies the value of the associated scale at which to begin the range. | double |
| stopValue | Specifies the value of the associated scale at which to end the range. | double |
| visible | Determines whether this component is displayed or hidden. Default is true. | boolean |

##### linear-scale

Purpose:

Defines the scale used in a linear gauge.

Equivalent in JClass Gauge:

JCLinearScale

Sub-Elements: ■ [tick](#_bookmark534)

* + - * [linear-range](#_bookmark522)
      * [font](#_bookmark490)
      * [image-file](#_bookmark500)
      * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)
      * [titled-border](#_bookmark536)
      * [compound-border](#_bookmark480)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| background | Specifies the color used in the background. | Color |
| direction | Determines the direction in which tick values increment on the scale. Default is Forward. | * Backward * Forward |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| max | Specifies the maximum value on the scale. No default; this attribute must be defined. | double |
| min | Specifies the minimum value on the scale. No default; this attribute must be defined. | double |
| opaque | Determines whether this component is opaque or transparent. Default is false. | boolean |
| orientation | Determines whether the scale is displayed horizontally or vertically. Default is Horizontal. | * Horizontal * Vertical |
| useZoomFactorForMax | Determines whether or not the zoomFactor value is applied to the maximum value on the scale. Default is true. | boolean |
| useZoomFactorForMin | Determines whether or not the zoomFactor value is applied to the minimum value on the scale. Default is true. | boolean |
| zoomFactor | Specifies the zoom factor as a percentage of the size of the scale, expressed in decimal form. The scale’s height in a horizontal scale (or the width in a vertical scale) is compressed by that percentage. For details, see [Section 5.6.5, “Setting the Zoom Factor](#_bookmark230) [on a Linear Scale,” on page 98](#_bookmark230). | double |

##### matte-border

Purpose:

Matte borders.

Equivalent in JClass Gauge:

javax.swing.border.MatteBorder

Sub-Elements: [insets](#_bookmark504)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| color | Determines the color of the border. No default; this attribute must be defined. | Color |

##### multi-col

Purpose:

Attributes to use when the legend element is defined with type=multiCol.

Equivalent in JClass Gauge:

com.klg.jclass.util.legend.JCMultiColLegend

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| numColumns | Specifies the number of columns. | int |
| numRows | Specifies the number of rows. | int |

##### needle

Purpose:

Specify a needle to use with the scale.

Equivalent in JClass Gauge:

JCNeedle, JCCircularNeedle, JCLinearNeedle

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| color | Determines the color of the needle. (Formerly called foreground.) | Color |
| innerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to begin drawing the needle. The percentage is applied differently for circular versus linear gauges. For details, see [Section](#_bookmark46) [1.5.2, “Sizing Components Using Extent](#_bookmark46) [Parameters,” on page 29](#_bookmark46). | double |
| interactionType | Specifies whether or not interaction with the needle is permitted, and if so, what type of interaction. Default is None. | * None * Drag * Click * Click\_Drag |
| name | Specifies a name for the needle to be used by the application. | String |
| outerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to stop drawing the needle. The percentage is applied differently for circular versus linear gauges. For details, see [Section](#_bookmark46) [1.5.2, “Sizing Components Using Extent](#_bookmark46) [Parameters,” on page 29](#_bookmark46). | double |
| reversed | Determines whether or not the direction of the needle is reversed. Default is false.  If reversed=true in a circular gauge, the needle is drawn pointing toward the center. For linear gauges, depending on the orientation, the needle points left instead of right or up instead of down. | boolean |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| style | Specifies the style of the needle. Default is Arrow. | * Rectangle * Circle * Pointer * Tailed\_Pointer * Arrow * Tailed\_Arrow * Triangle |
| value | Specifies a value on the scale. The needle will point to this value. | double |
| width | Specifies the width of the needle. | double |

##### radial-constraint

Purpose:

Specify the position of a component (usually a label) on a circular gauge. The center of the component is placed at the position specified.

Equivalent in JClass Gauge:

RadialConstraint

Sub-Elements: none

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| angle | Specifies the angle. No default; this attribute must be defined. | double |
| extent | Specifies the proportional distance of the label from the center of the scale. No default; this attribute must be defined. | double |

##### tick

Purpose:

Specify attributes for the ticks in a scale.

Equivalent in JClass Gauge:

JCTick, JCCircularTick, JCLinearTick

Sub-Elements: [font](#_bookmark490)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| automatic | Determines whether the tick increments are calculated automatically (true) or set by the incrementValue attribute (false). Default is true. | boolean |
| color | Determines the color of the tick. | Color |
| drawLabels | Determines whether or not tick labels are displayed. Default is true. | boolean |
| drawTicks | Determines whether or not ticks are displayed. Default is true. | boolean |
| fontColor | Specifies the color to use for the text. | Color |
| incrementValue | Specifies the increment to use between ticks when  automatic=false. | double |
| innerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to begin drawing the tick. The percentage is applied differently for circular versus linear gauges. For details, see [Section 1.5.2, “Sizing](#_bookmark46) [Components Using Extent Parameters,” on page](#_bookmark46) [29](#_bookmark46). | double |
| labelExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to place tick labels. The percentage is applied differently for circular versus linear gauges. For details, see [Section 1.5.2, “Sizing Components](#_bookmark46) [Using Extent Parameters,” on page 29](#_bookmark46). | double |

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| outerExtent | Specifies the position (as a percentage of the size of the gauge, expressed in decimal form) at which to stop drawing the tick. The percentage is applied differently for circular versus linear gauges. For details, see [Section 1.5.2, “Sizing Components](#_bookmark46) [Using Extent Parameters,” on page 29](#_bookmark46). | double |
| name | Specifies a name for the tick object. | String |
| precision | Specifies the precision used for creating tick labels. Positive values set the number of places *after* the decimal place. For example, using precision=3 causes the labels to be multiples of  0.001. Negative values set the number of zeros before the decimal place. For example, using precision=–3 the labels will increment in multiples of 1000. A value of zero causes the labels to be integers. | int |
| precisionUse Default | Determines whether or not the default precision value is used. When false, the value assigned to the precision attribute is used. Default is true. | boolean |
| reversed | Determines whether or not the direction of the tick is reversed. Default is false.  If reversed=true in a circular gauge, the tick is drawn pointing toward the center. For linear gauges, depending on the orientation, the tick points left instead of right or up instead of down. | boolean |
| startValue | Specifies the value at which to begin displaying ticks. | double |
| stopValue | Specifies the value at which to stop displaying ticks. | double |
| style | Determines the style used to draw the tick. Default is Line. | * Circle * Diamond * Line * Rectangle * Reverse\_ Triangle * Triangle |
| width | Specifies the width of the tick. | double |

##### titled-border

Purpose:

A titled border. Note that you cannot specify a titled-border or a compound-border as a sub-element.

Equivalent in JClass Gauge:

javax.swing.border.TitledBorder

Sub-Elements: ■ [font](#_bookmark490)

* + - * [empty-border](#_bookmark484)
      * [bevel-border](#_bookmark471)
      * [etched-border](#_bookmark486)
      * [line-border](#_bookmark514)
      * [matte-border](#_bookmark526)

Attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Values or Type** |
| color | The border’s color. | Color |
| title | The string for the title. No default; this attribute must be defined. | String |
| titleJustification | The title justification. Default is Default. | * Default * Left * Center * Right * Leading * Trailing |
| titlePosition | The position or placement of the title. Default is Default. | * Default * Above\_Top * Top * Below\_Top * Above\_Bottom * Bottom * Below\_Bottom |

##### value

Purpose:

Value for the [values](#_bookmark540) or [dash-array](#_bookmark482) elements.

Equivalent in JClass Gauge:

none

Sub-Elements: #PCDATA

Attributes: none

##### values

Purpose:

Values for the [graph](#_bookmark496) component.

Equivalent in JClass Gauge:

none

Sub-Elements: [value](#_bookmark538)

Attributes: none

**A**

API documentation [15](#_bookmark10)

### B

background [color,](#_bookmark177) *See* color

image, specifying for

center of a circular gauge [144](#_bookmark398)

circular gauge [84](#_bookmark179)

graph [74](#_bookmark153)

icon in an indicator gauge [44](#_bookmark73) indicator gauges and panels [62](#_bookmark119) linear gauge [84](#_bookmark179)

range in a circular or linear gauge [137](#_bookmark374)

scale in a circular or linear gauge [94](#_bookmark211)

base value indicator gauge

panel, setting [56](#_bookmark101)

standalone, setting [45](#_bookmark75)

base-gauge tag [169](#_bookmark463)

bevel-border tag [171](#_bookmark472)

blink, making indicator gauges [63](#_bookmark123)

border, setting on a linear scale [98](#_bookmark228)

### C

center

color, setting [144](#_bookmark396)

constructing a JCCenter object [142](#_bookmark390) object inheritance [141](#_bookmark384) preconstructed object, accessing [142](#_bookmark388) radius, setting [143](#_bookmark394)

center tag [172](#_bookmark474)

circular gauge assigning values [83](#_bookmark174)

background color, setting [84](#_bookmark177)

constructing a JCCircularGauge object [78](#_bookmark165)

gauge type, setting [79](#_bookmark166)

indicators, defining [115](#_bookmark285)

introduction [21](#_bookmark27)

needles, defining [116](#_bookmark289)

object inheritance [77](#_bookmark162)

part of a circle, defining [79](#_bookmark166)

range, defining [134](#_bookmark360)

scale, defining [89](#_bookmark187)

ticks, defining [102](#_bookmark245)

circular scale

[*See*](#_bookmark181) scale, circular circular-gauge tag [173](#_bookmark476)

circular-range tag [174](#_bookmark478)

color

circular gauge

background, setting [84](#_bookmark177)

center, setting [144](#_bookmark396)

indicator or needle, setting [118](#_bookmark306)

range, setting [137](#_bookmark372)

scale, setting [94](#_bookmark209)

tick mark, setting [108](#_bookmark263) graph background, setting [73](#_bookmark151) indicator gauge

background, setting [60](#_bookmark117)

ranges, setting in a panel [50](#_bookmark92)

ranges, setting in a standalone [42](#_bookmark71)

linear gauge

background, setting [84](#_bookmark177) indicator or needle, setting [118](#_bookmark306) range, setting [137](#_bookmark372)

scale, setting [100](#_bookmark235)

tick mark, setting [108](#_bookmark263) compound-border tag [176](#_bookmark481) constructor

JCBaseIndicatorGauge [48](#_bookmark83)

JCCenter [142](#_bookmark390)

JCCircularGauge [78](#_bookmark165)

JCCircularNeedle [116](#_bookmark293)

JCCircularRange [135](#_bookmark363)

JCCircularScale [90](#_bookmark192)

JCCircularTick [103](#_bookmark249)

JCGraph [69](#_bookmark135)

JCIndicatorGauge [41](#_bookmark63)

JCIndicatorGaugePanel [49](#_bookmark85)

JCLinearGauge [82](#_bookmark170)

JCLinearNeedle [116](#_bookmark293)

JCLinearRange [135](#_bookmark363)

JCLinearScale [96](#_bookmark218)

JCLinearTick [103](#_bookmark249)

## Index

### D

dash-array tag [177](#_bookmark483)

direction

graph, setting [71](#_bookmark143)

indicator gauge panel, setting [55](#_bookmark97)

linear scale, setting [98](#_bookmark226)

drawing order for components [28](#_bookmark44)

DTD

IndicatorGauge

base-gauge [169](#_bookmark463)

icon [169](#_bookmark465)

indicator-gauge [166](#_bookmark459)

indicator-gauge-panel [167](#_bookmark461)

indicator-range [170](#_bookmark467)

JCGauge

bevel-border [171](#_bookmark472)

center [172](#_bookmark474)

circular-gauge [173](#_bookmark476)

circular-range [174](#_bookmark478)

compound-border [176](#_bookmark481)

dash-array [177](#_bookmark483)

empty-border [177](#_bookmark485)

etched-border [177](#_bookmark487)

external-java-code [178](#_bookmark489)

font [179](#_bookmark491)

footer [179](#_bookmark493)

gauge [171](#_bookmark470)

gauge-area [180](#_bookmark495)

graph [181](#_bookmark497)

header [182](#_bookmark499)

image-file [183](#_bookmark501)

indicator [184](#_bookmark503)

insets [185](#_bookmark505)

label [186](#_bookmark507)

layout-hints [187](#_bookmark509)

legend [187](#_bookmark511), [189](#_bookmark513)

linear-constraint [191](#_bookmark519)

linear-gauge [192](#_bookmark521)

linear-range [193](#_bookmark523)

linear-scale [194](#_bookmark525)

line-border [190](#_bookmark515)

line-style [190](#_bookmark517)

matte-border [196](#_bookmark527)

multi-col [196](#_bookmark529)

needle [197](#_bookmark531)

radial-constraint [198](#_bookmark533)

tick [199](#_bookmark535)

titled-border [201](#_bookmark537)

value [202](#_bookmark539)

values [202](#_bookmark541)

overview [147](#_bookmark408)

**E**

empty-border tag [177](#_bookmark485)

etched-border tag [177](#_bookmark487) extents, sizing components [29](#_bookmark46) external-java-code tag [178](#_bookmark489)

### F

factory classes [148](#_bookmark410)

feature overview [11](#_bookmark2)

font tag [179](#_bookmark491)

footer

setting, in a circular or linear gauge [124](#_bookmark317)

size and position, setting [132](#_bookmark353)

footer tag [179](#_bookmark493)

foreground image, specifying in linear range [138](#_bookmark376)

### G

gauge

circular

[*See*](#_bookmark159) circular gauge

indicator gauge and panel

[*See*](#_bookmark53) indicator gauge

linear

[*See*](#_bookmark159) linear gauge

gauge tag [171](#_bookmark470)

gauge-area tag [180](#_bookmark495)

graph

assigning values [70](#_bookmark137)

background color, setting [73](#_bookmark151) constructing a JCGraph object [69](#_bookmark135) creating [65](#_bookmark125)

direction, setting [71](#_bookmark143)

line attributes, setting [73](#_bookmark149)

minimum and maximum values, setting [72](#_bookmark145)

pixel width for value points, setting [72](#_bookmark147)

graph tag [181](#_bookmark497)

### H

header

setting, in a circular or linear gauge [124](#_bookmark317)

size and position, setting [132](#_bookmark353)

header tag [182](#_bookmark499)

help [13](#_bookmark6)

how to use this guide [13](#_bookmark6)

### I

icon

shape, setting on an indicator gauge [58](#_bookmark111)

text font and color, setting [63](#_bookmark121)

icon tag [169](#_bookmark465)

image

and the PortableImage object [25](#_bookmark37) preparing to save to XML [157](#_bookmark439) specifying for

center background of a circular gauge [144](#_bookmark398)

circular gauge background [84](#_bookmark179)

graph background [74](#_bookmark153)

icon background in an indicator gauge [44](#_bookmark73) indicator gauge and panel backgrounds [62](#_bookmark119) linear gauge background [84](#_bookmark179)

range background in a circular or linear gauge [137](#_bookmark374)

range foreground in a linear gauge [138](#_bookmark376)

scale background in a circular or linear gauge [94](#_bookmark211)

specifying in XML [152](#_bookmark424)

understanding how images are saved to XML [159](#_bookmark447)

understanding how images are stored [25](#_bookmark37)

image-file tag [183](#_bookmark501)

indicator

constructing a JCIndicator object [115](#_bookmark287)

custom style, defining [119](#_bookmark309)

defining [115](#_bookmark285)

object inheritance [113](#_bookmark282)

style, setting [117](#_bookmark299)

indicator gauge

customizing properties [58](#_bookmark109)

introduction [20](#_bookmark25)

overview of objects [35](#_bookmark56), [65](#_bookmark128)

panel of gauges

assigning a value [49](#_bookmark87)

changing the gauge displayed [57](#_bookmark105)

constructing a JCBaseIndicatorGauge object [48](#_bookmark83) constructing a JCIndicatorGaugePanel object [49](#_bookmark85) creating [48](#_bookmark81)

customizing properties [50](#_bookmark90)

standalone gauge assigning a value [42](#_bookmark66) creating [41](#_bookmark63)

customizing properties [42](#_bookmark69), [71](#_bookmark141)

indicator tag [184](#_bookmark503)

IndicatorGauge DTD

[*See*](#_bookmark457) DTD IndicatorGauge indicator-gauge tag [166](#_bookmark459)

indicator-gauge-panel tag [167](#_bookmark461)

indicator-range tag [170](#_bookmark467)

insets tag [185](#_bookmark505)

internationalization [31](#_bookmark48)

### J

Javadoc for JClass DesktopViews [15](#_bookmark10)

JCAbstractIndicator [113](#_bookmark282)

JCAbstractNeedle [113](#_bookmark282)

JCAbstractScale [87](#_bookmark184)

JCAbstractTick [101](#_bookmark242)

JCCenter [141](#_bookmark384)

JCCircularIndicator [113](#_bookmark282)

JCCircularNeedle [113](#_bookmark282)

JCCircularScale [87](#_bookmark184)

JCCircularTick [101](#_bookmark242)

JCGraph [69](#_bookmark135)

JCGridLegend [125](#_bookmark323)

JCIndicator interface [113](#_bookmark282)

JCIndicatorGauge [41](#_bookmark63)

JCIndicatorGaugePanel [49](#_bookmark85)

JCLinearIndicator [113](#_bookmark282)

JCLinearNeedle [113](#_bookmark282)

JCLinearScale [87](#_bookmark184)

JCLinearTick [101](#_bookmark242)

JCMultiColumnLegend [125](#_bookmark325)

JCNeedle interface [113](#_bookmark282)

JCScale interface [87](#_bookmark184)

JCTick interface [101](#_bookmark242)

### L

label tag [186](#_bookmark507)

labels

aligning text [129](#_bookmark343)

defining [129](#_bookmark341)

tick

[*See*](#_bookmark239)tick labels

layering a graph on a gauge [27](#_bookmark41)

layering gauges [26](#_bookmark39)

layout-hints tag [187](#_bookmark509)

legend

column width, setting [126](#_bookmark333)

creating custom [128](#_bookmark336)

customizing [125](#_bookmark327)

JCLegend interfaces [128](#_bookmark338)

multi-column [125](#_bookmark324)

orientation, setting [125](#_bookmark329)

position, setting [125](#_bookmark331)

setting, in a circular or linear gauge [124](#_bookmark319)

single-column [125](#_bookmark322)

size and position, setting [132](#_bookmark353) truncated text, handling [126](#_bookmark334) types [124](#_bookmark321)

width, setting [126](#_bookmark333)

legend tag [187](#_bookmark511), [189](#_bookmark513)

licensing [15](#_bookmark12)

line attributes, setting on a graph [73](#_bookmark149)

linear constraint [131](#_bookmark351)

linear gauge assigning values [83](#_bookmark174)

background color, setting [84](#_bookmark177) constructing a JCLinearGauge object [82](#_bookmark170) indicators, defining [115](#_bookmark285)

introduction [21](#_bookmark29)

needles, defining [116](#_bookmark289)

object inheritance [77](#_bookmark162)

range, defining [134](#_bookmark360)

scale, defining [95](#_bookmark213)

ticks, defining [102](#_bookmark245)

linear scale

[*See*](#_bookmark181) scale, linear

linear-constraint tag [191](#_bookmark519)

linear-gauge tag [192](#_bookmark521)

linear-range tag [193](#_bookmark523)

linear-scale tag [194](#_bookmark525)

line-border tag [190](#_bookmark515)

line-style tag [190](#_bookmark517)

LoadProperties constructing [153](#_bookmark426)

overview [148](#_bookmark412)

using when creating a gauge [149](#_bookmark420)

using when loading a gauge [154](#_bookmark430)

### M

marks, tick

[*See*](#_bookmark239) tick marks matte-border tag [196](#_bookmark527)

metaphor [19](#_bookmark22)

multi-col tag [196](#_bookmark529)

### N

needle

constructing a JCCircularNeedle object [116](#_bookmark293) constructing a JCLinearNeedle object [116](#_bookmark293) custom style, defining [119](#_bookmark309)

defining [116](#_bookmark289)

length, setting [118](#_bookmark303)

object inheritance [113](#_bookmark282) preconstructed object, accessing [116](#_bookmark291) style, setting [117](#_bookmark299)

needle tag [197](#_bookmark531)

nested gauges, creating [26](#_bookmark39)

number of gauges in a panel, changing [57](#_bookmark107)

numbering precision in tick labels [111](#_bookmark273)

### O

offset

needle [118](#_bookmark303)

range in a circular gauge, creating [139](#_bookmark378)

order, drawing [28](#_bookmark44)

orientation

indicator gauge panel, setting [55](#_bookmark97)

legend, setting [125](#_bookmark329)

linear scale, setting [97](#_bookmark224)

OutputProperties constructing [157](#_bookmark441)

overview [148](#_bookmark414)

setting on a background image [158](#_bookmark443)

setting on a foreground image [158](#_bookmark445)

### P

padding

indicator gauge and panel, setting [59](#_bookmark115)

panel of indicator gauges

[*See*](#_bookmark53) indicator gauge

pixel width for value points, setting in a graph [72](#_bookmark147)

placing a gauge on top of a gauge [26](#_bookmark39) placing a graph on a gauge [27](#_bookmark41) PortableImage

overview [25](#_bookmark37)

understanding how images are save to XML [159](#_bookmark447)

precision [111](#_bookmark273)

### R

radial constraint [131](#_bookmark349)

radial-constraint tag [198](#_bookmark533)

radius

and needle length [118](#_bookmark303)

and ranges [136](#_bookmark369)

and sizing components [29](#_bookmark46)

and the center in a circular gauge [143](#_bookmark394)

of the scale [93](#_bookmark207)

range

breadth and location, setting [136](#_bookmark369)

color, setting [137](#_bookmark372)

constructing a JCCircularRange object [135](#_bookmark363) constructing a JCLinearRange object [135](#_bookmark363) customizing [136](#_bookmark367)

defining [134](#_bookmark360)

offset in a circular gauge, creating [139](#_bookmark378)

values, setting [135](#_bookmark365)

range of values

indicator gauge panel [50](#_bookmark92)

standalone indicator gauge [42](#_bookmark71)

related documents [15](#_bookmark14)

relative sizing [29](#_bookmark46)

rendering list [28](#_bookmark44)

### S

saving a gauge or graph to XML [156](#_bookmark434)

scale

circular

color, setting [94](#_bookmark209)

constructing a JCCircularScale object [90](#_bookmark192)

defining [89](#_bookmark187)

direction, setting [92](#_bookmark202)

initial size, setting [93](#_bookmark207)

min and max values, setting [90](#_bookmark196) preconstructed object, accessing [89](#_bookmark190) radius, getting the current [93](#_bookmark207)

start and stop values, setting [91](#_bookmark198)

zoom factor, setting [93](#_bookmark204)

introduction [87](#_bookmark181)

linear

border, setting [98](#_bookmark228)

color, setting [100](#_bookmark235)

constructing a JCLinearScale object [96](#_bookmark218)

defining [95](#_bookmark213)

direction, setting [98](#_bookmark226)

initial size, setting [99](#_bookmark233)

min and max values, setting [97](#_bookmark222) orientation, setting [97](#_bookmark224) preconstructed object, accessing [96](#_bookmark216) zoom factor, setting [98](#_bookmark230)

object inheritance [87](#_bookmark184)

shape of icon, setting on an indicator gauge [58](#_bookmark111) size of indicator gauge container, setting [59](#_bookmark113) sizing components using extents [29](#_bookmark46)

support [16](#_bookmark16)

JClass Community [17](#_bookmark18)

### T

technical support [16](#_bookmark16)

text

indicator gauge

icon font and color, setting [63](#_bookmark121) ranges, setting in a panel [50](#_bookmark92) ranges, setting in a standalone [42](#_bookmark71)

placing on a circular or linear gauge [129](#_bookmark341)

truncated in legend, handling [126](#_bookmark334)

tick

constructing a JCCircularTick object [103](#_bookmark249) constructing a JCLinearTick object [103](#_bookmark249) defining [102](#_bookmark245)

labels

customizing [110](#_bookmark269)

outside of scale bounds [112](#_bookmark277) precision of values, setting [111](#_bookmark273) space between, managing [112](#_bookmark275) text, adding [112](#_bookmark275)

marks

color, setting [108](#_bookmark263)

customizing [104](#_bookmark251)

number and bounds, setting [105](#_bookmark255) outside of scale bounds (clipped) [107](#_bookmark259) placement and length, setting [105](#_bookmark257) style and width, setting [107](#_bookmark261)

style, defining a custom [108](#_bookmark265)

object inheritance [101](#_bookmark242)

preconstructed object, accessing [103](#_bookmark247)

tick tag [199](#_bookmark535)

titled-border tag [201](#_bookmark537)

tooltips

indicator gauge

ranges, setting in a panel [50](#_bookmark92)

ranges, setting in a standalone [42](#_bookmark71)

### U

user interface metaphor [19](#_bookmark22)

### V

value tag [202](#_bookmark539)

value type

binary [54](#_bookmark95), [55](#_bookmark98), [57](#_bookmark103)

decimal [52](#_bookmark94), [55](#_bookmark98), [57](#_bookmark103)

magnitude [51](#_bookmark93), [56](#_bookmark102)

setting for an indicator gauge panel [50](#_bookmark92)

value, base

indicator gauge panel, setting [56](#_bookmark101)

indicator gauge standalone, setting [45](#_bookmark75)

values

circular gauge, assigning [83](#_bookmark174)

graph, assigning [70](#_bookmark137)

indicator gauge panel, assigning [49](#_bookmark87)

linear gauge, assigning [83](#_bookmark174)

standalone indicator gauge, assigning [42](#_bookmark66)

values tag [202](#_bookmark541)

values, range of

indicator gauge panel, setting [50](#_bookmark92)

standalone indicator gauge, setting [42](#_bookmark71)

### X

XML

creating a gauge or graph [149](#_bookmark417) overview for JClass Gauge [147](#_bookmark406) preparing to save images [157](#_bookmark439) sample file [151](#_bookmark422)

saving a gauge or graph [156](#_bookmark434) specifying images in [152](#_bookmark424) updating a gauge or graph [154](#_bookmark428)

### Z

zoom factor

circular scale, setting [93](#_bookmark204)

linear scale, setting [98](#_bookmark230)

z-order [28](#_bookmark44)