

Elements Programmer’s Guide

## 6.4.2

###### © 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.

###### Elements Programmer’s Guide January 2009

**Version 6.4.2**

**Table of Contents**

[Preface 9](#_bookmark0)

[Introducing JClass Elements 9](#_bookmark1)

[Assumptions 10](#_bookmark3)

[Typographical Conventions 10](#_bookmark4)

[Overview of this Guide 10](#_bookmark6)

[API Documentation (Javadoc) 12](#_bookmark7)

[Licensing 12](#_bookmark9)

[Related Documents 13](#_bookmark10)

[About Quest Software, Inc. 13](#_bookmark12)

[Part](#_bookmark17) [I: Components and Layout Managers](#_bookmark15)

1. [Introducing JClass Elements 17](#_bookmark18)
   1. [How the Manual is Organized 17](#_bookmark21)
   2. [Components and Layout Managers 17](#_bookmark24)
   3. [Internationalization 22](#_bookmark69)
2. [CheckBox-List Component 25](#_bookmark70)
   1. [Features of JCCheckBoxList 25](#_bookmark73)
   2. [Classes 26](#_bookmark74)
   3. [Properties 26](#_bookmark76)
   4. [Methods 26](#_bookmark78)
   5. [Examples 26](#_bookmark80)
3. [Date Chooser 29](#_bookmark83)
   1. [Features of JCDateChooser 29](#_bookmark86)
   2. [Classes and Interfaces 30](#_bookmark95)
   3. [Properties 32](#_bookmark110)
   4. [Methods 33](#_bookmark124)
   5. [Examples 34](#_bookmark130)
4. [JCPopupCalendar Component 35](#_bookmark131)
   1. [Features of JCPopupCalendar 35](#_bookmark134)
   2. [Classes 36](#_bookmark135)

3

* 1. [Properties 37](#_bookmark137)
  2. [Constructors and Methods 38](#_bookmark138)
  3. [Listeners and Events 39](#_bookmark140)
  4. [Examples 39](#_bookmark141)

1. [Exit Frame 43](#_bookmark142)
   1. [Features of JCExitFrame 43](#_bookmark145)
   2. [Properties 43](#_bookmark147)
   3. [Methods and Constructors 44](#_bookmark150)
   4. [Examples 44](#_bookmark155)
2. [Font Choosers 47](#_bookmark156)
   1. [Features of JCFontChooser and its Subclasses 47](#_bookmark159)
   2. [Classes 48](#_bookmark162)
   3. [Properties 49](#_bookmark169)
   4. [Methods 49](#_bookmark173)
   5. [Examples 49](#_bookmark176)
3. [HTML/Help Panes 51](#_bookmark177)
   1. [Features of JCHTMLPane 51](#_bookmark180)
   2. [Features of JCHelpPane 51](#_bookmark181)
   3. [Classes 52](#_bookmark183)
   4. [Properties 52](#_bookmark185)
   5. [Constructors and Methods 53](#_bookmark187)
   6. [Examples 54](#_bookmark201)
4. [Sortable Table 59](#_bookmark202)
   1. [Features of JCMappingSort 59](#_bookmark205)
   2. [Features of JCSortableTable 59](#_bookmark207)
   3. [Classes and Interfaces 60](#_bookmark210)
   4. [Constructors and Methods 61](#_bookmark220)
   5. [Cell Renderers for JCSortableTable 63](#_bookmark237)
   6. [Examples 64](#_bookmark239)
5. [Multiple Document Frame 71](#_bookmark243)
   1. [Features of JCMDIPane and JCMDIFrame 71](#_bookmark246)
   2. [Properties 74](#_bookmark252)
   3. [Methods 75](#_bookmark254)
   4. [Examples 78](#_bookmark290)
6. [Multi-Select List 79](#_bookmark291)
   1. [Features of JCMultiSelectList 79](#_bookmark294)
   2. [Properties 80](#_bookmark295)
   3. [Constructors and Methods 81](#_bookmark300)
   4. [Examples 82](#_bookmark314)
7. [Spin Boxes 83](#_bookmark315)
   1. [Features of JCSpinBox and JCSpinNumberBox 83](#_bookmark319)
   2. [Classes and Interfaces 84](#_bookmark320)
   3. [Properties 84](#_bookmark328)
   4. [Constructors and Methods 86](#_bookmark347)
   5. [Examples 87](#_bookmark358)
8. [Splash Screen 89](#_bookmark361)
   1. [Features of JCSplashScreen 89](#_bookmark364)
   2. [Classes and Interfaces 89](#_bookmark366)
   3. [Methods and Constructors 90](#_bookmark368)
   4. [Examples 90](#_bookmark372)
9. [Tree/Table Components 93](#_bookmark373)
   1. [Features of JCTreeExplorer and JCTreeTable 93](#_bookmark377)
   2. [Classes and Interfaces 96](#_bookmark378)
   3. [Properties 98](#_bookmark391)
   4. [Methods 99](#_bookmark401)
   5. [Examples 103](#_bookmark461)
10. [Wizard Creator 105](#_bookmark462)
    1. [Features of JCWizard and JCSplitWizard 105](#_bookmark465)
    2. [Classes 107](#_bookmark466)
    3. [Constructors and Methods 107](#_bookmark472)
    4. [Events 109](#_bookmark489)
    5. [Examples 109](#_bookmark500)
11. [Layout Managers 111](#_bookmark501)
    1. [Features of the Layout Managers in JClass Elements 111](#_bookmark504)
    2. [Interfaces 113](#_bookmark516)
    3. [Properties 113](#_bookmark518)
    4. [Constructors and Methods 114](#_bookmark527)
    5. [Examples 116](#_bookmark540)

[Part](#_bookmark545) [II: Utility Classes](#_bookmark543)

1. [Introduction to the Utility Classes 121](#_bookmark546)
   1. [Utilities 121](#_bookmark550)
2. [Debugging Tools 125](#_bookmark564)
   1. [Features of JCDebug 125](#_bookmark567)
   2. [Class 126](#_bookmark571)
   3. [Methods 126](#_bookmark572)
   4. [Examples 128](#_bookmark586)
3. [JCFileFilter 129](#_bookmark587)
   1. [Features of JCFileFilter 129](#_bookmark591)
   2. [Constructors 129](#_bookmark592)
   3. [Methods 130](#_bookmark594)
   4. [Example 131](#_bookmark602)
4. [Icon Creator 133](#_bookmark603)
   1. [Features of JCIconCreator 133](#_bookmark606)
   2. [Classes 133](#_bookmark609)
   3. [Constructors and Methods 133](#_bookmark611)
   4. [Examples 135](#_bookmark618)
5. [Image Encoder 137](#_bookmark619)
   1. [Features of JCEncodeComponent 137](#_bookmark622)
   2. [Classes and Interfaces 137](#_bookmark624)
   3. [Constructors and Methods 138](#_bookmark626)
   4. [Examples 139](#_bookmark633)
6. [Listener List 141](#_bookmark634)
   1. [Features of JCListenerList 141](#_bookmark637)
   2. [Classes 141](#_bookmark638)
   3. [Methods 141](#_bookmark640)
   4. [Examples 142](#_bookmark645)
7. [Progress Helper 143](#_bookmark647)
   1. [Features of JCProgressHelper 143](#_bookmark651)
   2. [Constructors and Associated Classes 144](#_bookmark653)
   3. [JCProgressHelper Methods 146](#_bookmark664)
   4. [Examples 147](#_bookmark679)
8. [String Tokenizer 149](#_bookmark680)
   1. [Features of JCStringTokenizer 149](#_bookmark683)
   2. [Classes 149](#_bookmark684)
   3. [Methods 150](#_bookmark686)
   4. [Examples 150](#_bookmark695)
9. [Thread Safety Utilities 153](#_bookmark696)
   1. [Features of the Thread Safety Classes 153](#_bookmark699)
   2. [Methods 153](#_bookmark704)
10. [Tree Set 155](#_bookmark712)
    1. [Features of JCTreeSet 155](#_bookmark715)
    2. [Constructors and Methods 156](#_bookmark716)
    3. [Examples 156](#_bookmark718)
11. [Type Converters 157](#_bookmark719)
    1. [Features of JCTypeConverter 157](#_bookmark722)
    2. [Features of JCSwingTypeConverter 158](#_bookmark723)
    3. [Classes 158](#_bookmark725)
    4. [Methods 158](#_bookmark726)
    5. [Examples 160](#_bookmark757)
12. [Word Wrap 165](#_bookmark758)
    1. [Features of JCWordWrap 165](#_bookmark761)
    2. [Methods 165](#_bookmark763)
    3. [Examples 166](#_bookmark767)

[Part](#_bookmark770) [III: Reference Appendices](#_bookmark768)

1. [Bean Properties Reference 169](#_bookmark772)
   1. [Beans in the Swing Package 170](#_bookmark777)
   2. [Beans in the com.klg.jclass.util.swing Package 177](#_bookmark791)
2. [Distributing Applets and Applications 191](#_bookmark813)
3. [Colors and Fonts 193](#_bookmark820)
   1. [Colorname Values 193](#_bookmark823)
   2. [RGB Color Values 193](#_bookmark825)
   3. [Fonts 198](#_bookmark828)

[Index 201](#_bookmark834)

## Preface

[*Introducing JClass Elements*](#_bookmark1)■ [*Assumptions*](#_bookmark3)■ [*Typographical Conventions*](#_bookmark4)[*Overview of this Guide*](#_bookmark6)■ [*API Documentation (Javadoc)*](#_bookmark7)■ [*Licensing*](#_bookmark9)[*Related Documents*](#_bookmark10)■ [*About Quest Software, Inc.*](#_bookmark12)

#### Introducing JClass Elements

The Swing components are the most significant part of the Java Foundation Classes (JFC). Swing components cover basic needs, but some commonly useful items are

missing. For instance, a Color Chooser component is included, but a Font Chooser is not.

JClass Elements is a broad collection of GUI components and utility classes designed to augment Swing’s basic offerings. With JClass Elements, you have an extended set of off- the-shelf user interface components at your disposal. Moreover, because of their open design, it’s easy to adapt them to your own custom needs.

###### Feature Overview

The classes of JClass Elements are distributed over three packages. com.klg.jclass.util contains a collection of utilities and com.klg.jclass.swing contains the more elaborate GUI components. There are additional classes in com.klg.jclass.util.swing. This package contains both utilities and some basic GUI components that add functionality to their Swing ancestors.

JClass Elements is a collection of utilities and GUI components which:

* + - Extends basic Swing functionality by rounding out the list of much-needed components.
    - Simplifies your work by providing built-in functionality.
    - Implements commonly-required utility functions.

JClass Elements may be used in conjunction with all of Quest Software’s other JClass products, as well as with ordinary Swing components.

JClass Elements is compatible with JDK 1.4. If you are using JDK 1.4 and experience drawing problems, you may want to upgrade to the latest drivers for your video card from your video card vendor.

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

#### Assumptions

This manual assumes that you have some experience with the Java programming language. You should have a basic understanding of object-oriented programming and Java programming concepts such as classes, methods, and packages before proceeding with this manual. See [Related Documents](#_bookmark10) later in this section of the manual for additional sources of Java-related information.

#### Typographical Conventions

Typewriter Font ■ Java language source code and examples of file contents.

* + - * JClass Elements and Java classes, objects, methods, properties, constants, and events.
      * HTML documents, tags, and attributes.
      * Commands that you enter on the screen.

*Italic Text* ■ Pathnames, filenames, URLs, programs, and method parameters.

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

**Bold** ■ Keyboard key names and menu references.

#### Overview of this Guide

Part I —Components and Layout Managers – contains information about JClass Elements’s GUI components. Each chapter explains what the component is, and describes how to use it in your development project. There is also a chapter describing the behavior of the layout managers that JClass Elements provides. These functional yet simple-to-use layout managers can ease your layout tasks.

[Chapter 1, Introducing JClass Elements,](#_bookmark19) provides an overview of the components in JClass Elements.

[Chapter 2, CheckBox-List Component,](#_bookmark71) describes the use of a component that associates check boxes with list items.

[Chapter 3, Date Chooser,](#_bookmark84) describes the use of a graphical date chooser component.

[Chapter 4, JCPopupCalendar Component,](#_bookmark132) introduces a component that allows you to edit the date and time using a drop-down calendar.

[Chapter 5, Exit Frame,](#_bookmark143) outlines this subclass of Swing’s JFrame, which is used to detect and react to window-closing events.

[Chapter 6, Font Choosers,](#_bookmark157) details the JCFontChooser class, which gives you an easy way of letting your end users change fonts.

[Chapter 7, HTML/Help Panes,](#_bookmark178) covers the use of this subclass of Swing’s JEditorPane, which provides added HTML, hyperlink, and cursor changing functionality.

[Chapter 8, Sortable Table,](#_bookmark203) offers information about this index map sorting class.

[Chapter 9, Multiple Document Frame,](#_bookmark244) outlines this multiple document interface component, which allows you to put multiple windows in the same pane.

[Chapter 10, Multi-Select List,](#_bookmark292) covers the use of this dual-list component, which handles tasks like specifying file inclusion and exclusion by providing a GUI containing two list areas. Items can be moved from one list area to the other. The names in the *selected* list are marked for the action you designate, while those in the *deselected* list are excluded.

[Chapter 11, Spin Boxes,](#_bookmark316) presents an overview of this incrementing and decrementing component, which is used with java.lang.Number type objects.

[Chapter 12, Splash Screen,](#_bookmark362) shows you how to include a splash screen with your application.

[Chapter 13, Tree/Table Components,](#_bookmark374) provides information about the table component, which presents data as a hierarchical/tree listing or a non-hierarchical grid listing.

[Chapter 14, Wizard Creator,](#_bookmark463) covers the use of a component that manages pages with wizard-like behavior. Typically, these pages are dialogs that assist the end user in setting up custom configurations by organizing the setup procedure.

[Chapter 15, Layout Managers,](#_bookmark502) covers the behavior and use of the JClass Elements layout managers.

Part II— Utilities – describes how to use the utility classes in JClass Elements. Each chapter explains what the class is and describes how to use it in your development project.

[Chapter 16, Introduction to the Utility Classes,](#_bookmark547) describes JClass Elements’s utility classes.

[Chapter 17, Debugging Tools,](#_bookmark565) covers this tool that provides three different types of debug printout control.

[Chapter 18, JCFileFilter,](#_bookmark588) provides a convenient way of passing Windows-style filename extensions to a Swing JFileChooser so that only files of the named types appear in the file chooser dialog.

[Chapter 19, Icon Creator,](#_bookmark604) outlines how you can use String arrays to create icon images. This eliminates the need to supply separate image files for the icons in your class.

[Chapter 20, Image Encoder,](#_bookmark620) describes how to use this class to provide a picture of your component.

[Chapter 21, Listener List,](#_bookmark635) describes how to use this class for keeping track of event listeners.

[Chapter 22, Progress Helper,](#_bookmark648) offers information about this index map sorting class.

[Chapter 23, String Tokenizer,](#_bookmark681) outlines the capabilities of this class, which lets you specify a delimiter and split a String into tokens.

[Chapter 24, Thread Safety Utilities,](#_bookmark697) describes the classes that help with thread safety.

[Chapter 25, Tree Set,](#_bookmark713) outlines the features of this class, which allows you to represent a set’s elements as a sort tree.

[Chapter 26, Type Converters,](#_bookmark720) outlines how to use these classes to convert between data types.

[Chapter 27, Word Wrap,](#_bookmark759) shows how to add word wrapping functionality to a String.

Part III — Reference Appendices – contains detailed technical reference information.

[Appendix A, Bean Properties Reference](#_bookmark773), gives important property details of all JClass Elements’s components.

[Appendix B, Distributing Applets and Applications](#_bookmark814), describes how to package your application for distribution using JClass JarMaster.

[Appendix C, Colors and Fonts](#_bookmark821), provides you with a useful table of Color values.

#### API Documentation (Javadoc)

The *JClass DesktopViews API Documentation* (Javadoc) is installed automatically when you install JClass Elements and is found in the *JCLASS\_HOME/docs/api/* directory.

#### Licensing

In order to use JClass Elements, you need a valid license. Complete details about licensing are outlined in the *JClass DesktopViews Installation Guide*, which is automatically installed when you install JClass Elements.

#### Related Documents

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

* + - “*Java Platform Documentation*” at [*http://java.sun.com/docs/index.html*](http://java.sun.com/docs/index.html) and the “*Java Tutorial*” at [*http://java.sun.com/docs/books/tutorial/index.html*](http://java.sun.com/docs/books/tutorial/index.html) from Sun Microsystems
    - For an introduction to creating enhanced user interfaces, see “*Creating a GUI with JFC/Swing*” at [*http://java.sun.com/docs/books/tutorial/uiswing/index.html*](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/).
    - Resources for using JavaBeans are at [*http://java.sun.com/beans/resources.html*](http://java.sun.com/beans/resources.html)

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

#### 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/)

*Part I* *Components and Layout*

*Managers*

***1***

## Introducing JClass Elements

[*How the Manual is Organized*](#_bookmark21)[*Components and Layout Managers*](#_bookmark24)[*Internationalization*](#_bookmark69)

#### How the Manual is Organized

For the most part, each chapter is devoted to a single component. This makes it easy to find a component or utility, and makes it easy to review its structure and usage. In very few cases, utilities that are very closely related are covered in the same chapter.

You’re reading Part I right now. You’ll find JClass Elements’s Graphical User Interface (GUI) components and layout managers in Part I and JClass Elements’s utility classes in Part II.

This chapter contains a general description of each GUI component and layout manager in the product.

#### Components and Layout Managers

It’s as easy to use a JClass Elements component in your program as it is to use a Swing component. Where you would reference a JComponent if it were part of Swing, you instead reference a JCComponent, after making sure that the JClass Elements packages are on your CLASSPATH.

Some of JClass Elements’s components, like JCMDIFrame, augment the standard Swing components. Others, like JCFontChooser, provide a new component, one that is not part of the standard Swing package. Either way, they add functionality.

##### 17

Here’s a brief note on each component or layout manager:

|  |  |
| --- | --- |
| **Component Name** | **Description** |
| JCCheckBoxList | This component lets you show a columnar list of items. A check box appears at the left of each item. A mode switch lets you set the selection policy between (a) only one box at a time, (b) a contiguous range, or (c) any group whatsoever.  You can implement the javax.swing.event.ListSelectionListener interface to respond immediately when a user checks a box. |
| JCDateChooser | This component provides a convenient way of viewing and setting calendar information. The days of the month appear as a standard monthly calendar, the year is in a spin box, and you have a choice of a spin box or a pop-down list for the month. |
| JCExitFrame | A frame that responds to window-closing events either by exiting or by becoming invisible. |
| JCFontChooserPane | JCFontChooser is the abstract base class for JCFontChooserBar and JCFontChooserPane. It provides common data and methods for both components. Place one of these in any application where you want to let the end user choose fonts. |
| JCHTMLPane and  JCHelpPane | Ease of use is the key feature for these two components. Simply pass an HTML-encoded file to the pane and you have a mini browser. JCHelpPane supports up to three JCHTMLPanes and provides basic navigation buttons, allowing you to implement a simple HTML-based help system. The left pane functions as a table of contents, and the right pane shows the help pages. |
| JCMDIFrame and  JCMDIPane | Multiple document windows are great where multiple views of some multi-faceted object may be required, or multiple forms need to be simultaneously present in a window. The JClass versions optimize space by managing the active window’s main menu bar, while providing the standard window-management options. |

|  |  |
| --- | --- |
| **Component Name** | **Description** |
| JCMultiSelectList | JCMultiSelectList matches the API for JList except that two lists instead of one appear in the component’s GUI. There are four buttons in between the list areas that move items between the lists. The left-hand list contains non- selected items and the right-hand list contains the selected items. |
| JCSpinBox | Swing provides checkboxes and radio buttons, but no spin boxes. The JClass spin boxes fill the need for components that let the user select a number or a String by clicking on up- or down-arrows. |
| JCSpinNumberBox | Use JCSpinNumberBox for incrementing and decrementing objects of type java.lang.Number. You can select numbers of type Byte, Short, Integer, Long, or Float, and you can set maximum and minimum values for the spin operation. |
| JCSortableTable | A subclass of JTable that internally wraps any TableModel it is given with a JCRowSortTableModel and provides a Comparator that has a adjustable list of the column indexes that it uses for sorting. Clicking on a column header invokes the sorting behavior tied to that column; clicking again reverses the sort. It can be used to sort Dates, Objects that implement Comparable, and wrapped primitive types. For more information, see [Features of JCSortableTable, in](#_bookmark208) [Chapter](#_bookmark208) 8, for a description and examples. |
| JCSplashScreen | A splash screen is an image that appears while an application is loading. It serves both as an indication that the program is being loaded from disk and as a place to put notices, such as copyrights, version or release numbers, and the like. |
| JCTreeExplorer | A subclass of JTable that handles listeners, rendering, editing, and painting of a component that combines tree-like and table-like properties. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | | **Description** | |
| JCPopupCalendar | | JCPopupCalendar is a component that allows you to edit the date and time using a dropdown calendar. | |
| JCTreeTable | | Swing’s JTree and JTable are the two components that do more than merely display data; they attempt to manage the data as well. This becomes important when you need to organize large amounts of data and provide a view that displays a portion of it along with an indication of its relationship to the rest. Information that has a hierarchical structure, like a file system, can be displayed as tree data, while other types of data nicely fit a tabular format. There are a large number of data structures that combine tree-like and a table-like properties. A file system has a hierarchical organization that begs to be represented as a tree, yet the individual directories and files have properties, such as name, size, type, and date modified, that fit nicely in a row-column organization. Obviously there is a need for a component that lets you combine the look and functionality of both a tree and a table. | |
| JCWizard and  JCSplitWizard | | JCWizard and JCSplitWizard let you create and manage a Wizard-style group of dialogs by supplying informative events and special page components with standard buttons. You add a JCWizardListener to your JCWizardPages to invoke the actions that each page needs to perform. | |
| Layout Managers | | The layout managers are JCAlignLayout, JCColumnLayout, JCElasticLayout, JCGridLayout, and JCRowLayout.  JCBorder, JCBox, JCBrace, and JCSpring are the associated components. Use them as enhancements to the AWT layout managers.  Use JCAlignLayout to vertically arrange components with their associated labels, and JCRowLayout to arrange components in a single row.  JCGridLayout improves AWT’s GridLayout by sizing cells more intelligently.  JCBorder lets you place your borders anywhere, not just around components. | |

The following table lists some JClass Elements objects, with their nearest Swing relatives. The accompanying description informs you about the advantages you gain by using the JClass Elements component.

|  |  |  |
| --- | --- | --- |
| **Swing** | **JClass Elements** | **Description** |
| JEditorPane | JCHTMLPane | Its constructor takes either an HTML String or a URL, making it easy to add HTML pages to a pane. Follows hyperlinks without having to add listeners explicitly. |
| JPane | JCMDIPane | A pane that can hold multiple document interface (MDI) frames. |
| JInternalFrame | JCMDIFrame | Supports the multiple document interface paradigm with the automatic addition of a “Windows” menu to the parent menubar. |
| JFrame | JCExitFrame | Automatically responds to window closing events. |
| JTree JTable | JCTreeTable JCTreeExplorer | Components that combine tree and table views of hierarchically ordered data. |
| No Swing equivalent. | JCFontChooser JCFontChooserBar JCFontChooserPane | Choose fonts from a menu or a dialog. |
| (AWT) GridLayout | JCAlignLayout | An easy way to lay out a two-column grid. |
| No Swing equivalent. | JCElasticLayout | For laying out components in a single row or a single column. Any leftover space in the component is divided among the components in the way you specify. |
| JSpinner | JCSpinBox JCSpinNumberBox | These spin boxes are the top half of a combo box. They are useful when you don’t need a drop-down list, and they don’t subclass from JComboBox.  Instead, they inherit from Swing’s  AbstractSpinBox. |

|  |  |  |
| --- | --- | --- |
| **Swing** | **JClass Elements** | **Description** |
| No Swing equivalent. | JCDateChooser | JCDateChooser is a component that displays a calendar in one of four variant forms. Each one displays the days of the month in the familiar form of a calendar, but varies the ways that the month and year are displayed. |
| No Swing equivalent. | JCWordWrap | Wraps lines, given a length and a newline delimiter. |
| JProgressBar | JCProgressHelper | A thread-safe class that reports via a dialog just how far along some time- consuming operation is. |

#### 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 Elements, without the need for you to do any extra work.

To localize your JClass Elements, you need the JClass Elements 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

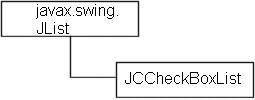
[*http://java.sun.com/j2se/1.4.2/docs/guide/intl/index.html*](http://java.sun.com/j2se/1.4.2/docs/guide/intl/index.html).

# 2

## CheckBox-List Component

[*Features of JCCheckBoxList*](#_bookmark73)[*Classes*](#_bookmark74)[*Properties*](#_bookmark76)[*Methods*](#_bookmark78)[*Examples*](#_bookmark80)

#### Features of JCCheckBoxList



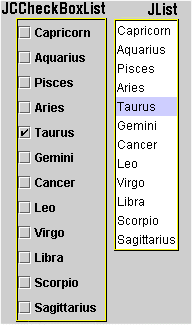
A JCCheckBoxList functions just like a JList, except that a check box appears to the left of the list items. (See [Figure](#_bookmark82) 2.)

It is a subclass of the JList component that implements a JCheckBox as the cell renderer.

To use a com.klg.jclass.util.swing.JCCheckBoxList in your application, simply ensure that the JClass Elements JAR is part of your CLASSPATH.

Constructors are of the no argument type, or a single argument consisting of an instance of a ListModel, an array of Objects (usually Strings), or a Vector of list items.

For comparison purposes, a JCCheckBoxList is shown beside a JList in the following figure.



*Figure 1 Comparison of a JCCheckBoxList with Swing’s default JList.*

##### 25

#### Classes

com.klg.jclass.util.swing.JCCheckBoxList — The component itself.

[javax.swing.event.ListSelectionListener](http://www.javasoft.com/products/jdk/1.2/docs/api/javax/swing/event/ListSelectionListener.html) — for listening to changes in the list.

#### Properties

JCCheckBoxList does not need any extra properties. You are free to use all the properties it inherits from [JList](http://www.javasoft.com/products/jdk/1.2/docs/api/javax/swing/JList.html). For a full listing of the properties, see [Appendix A, Bean Properties](#_bookmark795) [Reference](#_bookmark795).

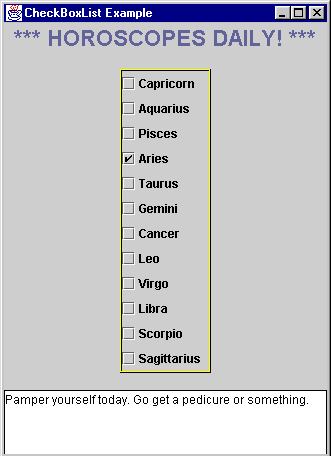
#### Methods

JCCheckBoxList does not need any extra methods. You are free to use all the methods it inherits from [JList](http://www.javasoft.com/products/jdk/1.2/docs/api/javax/swing/JList.html). Selection is handled just like a JList; you can choose one of three selection modes: single, block, or multiple block.

#### Examples

This example shows the use of a JCCheckBoxList, including the action taken when an item is checked. If you look at the full listing in examples.elements.CheckBoxList, you’ll observe that the only modification needed to replace a JList with a JCCheckBoxList is the change of the component’s name and this import line:

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



*Figure 2 A JCCheckBoxList.*

The code that instantiates a JCCheckBoxList is:

public JCCheckBoxList list;

list = new JCCheckBoxList(data);

list.setBorder(new EtchedBorder(Color.black, Color.yellow)); list.setForeground(Color.white);

list.setSelectionMode(0); list.addListSelectionListener(this); add(list);

Since JCCheckBoxList inherits all the properties of JList, it also uses ListSelectionEvent and ListSelectionListener to let you track the list items that have been selected. The code fragment shown below, also taken from examples.elements.CheckBoxList, shows how to use the ListSelectionEvent to react when items in a JCCheckBoxList are selected. In this example, a horoscope based on the selected item is placed in a text area.

//================== ListSelectionListener interface methods ======== public void valueChanged(ListSelectionEvent e) {

int index = list.getSelectedIndex();

switch(index) { case 0:

horoscope.setText(

"A homeless puppy will follow you home. Be good to it."); break;

...

More messages for the JTextArea

...

case 19:

horoscope.setText("You're running low on supplies. Run out &

stock up.");

break;

}

}

}

Chapter 2 CheckBox-List Component **27**

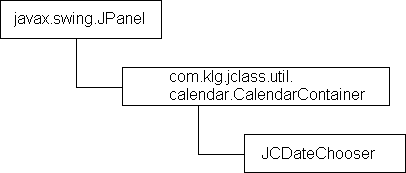
# 3

## Date Chooser

[*Features of JCDateChooser*](#_bookmark86)[*Classes and Interfaces*](#_bookmark95)[*Properties*](#_bookmark110)[*Methods*](#_bookmark124)[*Examples*](#_bookmark130)

#### Features of JCDateChooser

JCDateChooser is a component that displays a calendar in one of four variant forms. Each one displays the days of the month in the familiar form of a calendar, but varies the ways that the month and year are displayed.



The different styles are:

**Spin Popdown**

The year is shown in a spin box; the month is shown in a popdown.



**Dual Spin**

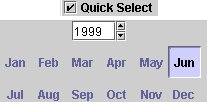
Spin boxes are used to display both the year and the month.



**Quick Select**

The year is shown in a spin box; a table is used to display all twelve months. One of the months may be highlighted to indicate that it has been selected.

##### 29



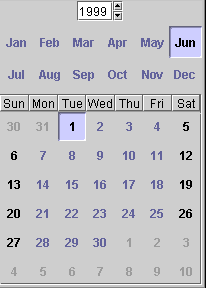
**Read Only**

The year and month are shown in non-editable fields. The table showing the days of the month is read-only as well. Selected “special” dates still appear highlighted.



Like the standard Swing components, JCDateChooser provides for the optional use of a *Tool Tip*.

The accompanying figure shows the full component in its *Quick Select* form, so the year is shown in a spin box, while tables are used to show months and days of the month. Note that special days, in this case Saturdays and Sundays, are distinguishable from the others. You can observe the other calendar styles by running the example called *examples.elements.DateChooser.java*.



*Figure 3 A JCDateChooser.*

#### Classes and Interfaces

###### Classes in the com.klg.jclass.util.calendar Package

|  |  |
| --- | --- |
| AbstractLabel | An abstract class for setting dates in a locale-dependent fashion. It is used for MonthLabels and YearLabels. |
| DayTable | The days of the month. |

|  |  |
| --- | --- |
| JCCalendarContainer | A container that manages CalendarComponent children. That is, any calls to the calendar component interface methods are automatically passed to any calendar component children. |
| JCCalendar | A calendaring utility class that can define special dates and custom date classes, such as “Tuesdays” or “April Fool’s Day” (April 1).  JCCalendar augments java.util.Calendar by providing extra date-specific capabilities. |
| JCDateChooser | A GUI component with four styles of calendar. Special dates display differently from other dates. |
| MonthLabel | Displays a locale-specific table of month labels. |
| MonthPopdown | Encapsulates the months of the year in a popdown. |
| MonthSpin | Encapsulates the months of the year in a spin box. |
| MonthTable | Encapsulates the months of the year in a table. |
| YearLabel | Presents the designated year in a label. |
| YearSpin | Presents the designated year in a spin box. |

* + 1. **The** **CalendarComponent Interface**

The calendar component uses a single model for the day, month, and year. The methods declared in public interface CalendarComponent are:

public void setCalendarModel(JCValueModel model)

Sets the model which provides the current date being used by the calendar component.

public void setSpecialDates(JCCalendar special\_dates

Sets the special dates being used by the calendar component.

public void setLocale(Locale locale);

Sets the locale being used by the calendar component.

public void addActionListener(ActionListener l)

Adds an action listener to detect specific actions on this component.

public void removeActionListener(ActionListener l)

Removes action listener to detect specific actions on this component.

* + 1. **The** **SpecialDate Interface**

This interface has only one method:

boolean isSpecialDate(int year, int month, int date, int week)

You’ll note that the numeric value for the week (1 - 52) is a redundant parameter in isSpecialDate. This is done for efficiency’s sake. If you implement the SpecialDate interface, you will have to supply a numeric value for the week even though it is possible to compute it from the first three parameters in isSpecialDate. Note that you use JCCalendar’s dayofweek() method to calculate this value.

#### Properties

###### Properties of JCDateChooser

|  |  |
| --- | --- |
| chooserType | For specifying the date chooser type, use one of JCDateChooser.DUAL\_SPIN, JCDateChooser.QUICK\_SELECT, JCDateChooser.READ\_ONLY, or JCDateChooser.SPIN\_POPDOWN. |
| days | The days array\* used by the date chooser |
| minimumDate, maximumDate | Bounds between which dates are valid. Any date outside these bounds will be rejected by the validator. |
| months | The months array\* used by the date chooser |
| shortMonths | The months’ short form array\* used by the date chooser |
| toolTipText | The text that appears in the tool tip box when the mouse pointer rests over the component. |
| value | The currently selected date. |

\*This array must be at least as long as what the JCDateChooser’s locale expects. By default, the array is initialized to the locale’s default list.

###### Properties of JCCalendar

|  |  |
| --- | --- |
| addSpecialDate, removeSpecialDate | Mark a special date on the calendar, or remove one that has already been designated as special. |
| isSpecialDate | A Boolean indicating whether the given date is special. |

For a full listing of the properties, see [Appendix A](#_bookmark776).

#### Methods

###### JCDateChooser

There are four visual aspects to the date chooser: *Quick Select*, *Dual Spin*, *Spin Popdown*, and *Read Only*. Use setChooserType(int type) to select the type you want to display. type is one of JCDateChooser.DUAL\_SPIN, JCDateChooser.QUICK\_SELECT, JCDateChooser.READ\_ONLY, or JCDateChooser.SPIN\_POPDOWN.

The CalendarComponent interface provides the mechanism for extracting parts of a JCDateChooser date. The methods are getDayComponent(), getMonthComponent(), and getYearComponent().

As noted in the section on properties, you set minimum and maximum dates by providing setMinimumDate() and setMaximumDate() with a java.util.Calendar object.

Set the currently selected date programmatically with setValue(), or determine what its value is with getValue(). The parameter is once again a java.util.Calendar object.

###### JCCalendar

While not subclassed from java.util.Calendar, JCCalendar is used in conjunction with it to provide for a classification of some dates as “special.” Special days are managed through these methods: addSpecialDate() and removeSpecialDate(), which take a SpecialDate object as a parameter. and isSpecialDate(). There is no restriction on how many dates may be deemed special.

The class contains a number of utility methods. One, called isLeapYear(), can be used to determine if any given year is a leap year. With dayOfWeek(), you can determine the day of the week given a year, month, and day. You can clone a Calendar object using copyCalendar().

Certain applications involving calendars require that certain days are treated specially. For example, some businesses that are open on the weekend close on Mondays. In such a case, it is useful to be able to lump all Mondays together and classify them as days when the store is closed. Perhaps the store’s founder always holds a sale on his birthday, the 29th of February. In this and similar cases, it’s useful to be able to denote anniversary days that occur on the same date every year. There are other days, such as \ Labor day, which is defined as the first Monday in September. JCCalendar contains inner classes DayOfWeek, MonthDayOfMonth, MonthWeekDayOfWeek to help you deal with these situations. These classes allow you to store various calendar objects of the types just mentioned. The first of these allows you to store a day, like Sunday, by declaring an instance variable

DayOfWeek sunday = new DayOfWeek(0);

From the example, you see that the seven days of the week are mapped using a zero- based index.

To store a date like July 4, use:

MonthDayOfMonth july4 = new MonthDayOfMonth(7, 4)

To store a date like Labor Day, use:

MonthWeekDayOfWeek laborDay = new MonthWeekDayOfWeek(9, 1, 1)

#### Examples

The illustrative code snippets shown here demonstrate how you can create special days and how you can set bounds on the permissible dates. Refer to the Date Chooser example, automatically installed into *com/klg/jclass/examples/elements/* when you install JClass Elements, for the complete example.

//The location of JCDateChooser import com.klg.jclass.util.calendar.\*;

//Create an instance of JCDateChooser within your class. dateChooser = new JCDateChooser();

...

//Create a "special day"

JCCalendar special\_dates = new JCCalendar();

// Make Sundays special days special\_dates.addSpecialDate(new JCCalendar.DayOfWeek(0));

...

dateChooser.setSpecialDates(special\_dates);

...

//Set bounds for the calendar Calendar max = Calendar.getInstance();

max.set(max.YEAR, 2050); dateChooser.setMaximumDate(max);

# 4

## JCPopupCalendar Component

[*Features of JCPopupCalendar*](#_bookmark134)[*Classes*](#_bookmark135)[*Constructors and Methods*](#_bookmark138)[*Listeners and Events*](#_bookmark140)[*Examples*](#_bookmark141)

#### Features of JCPopupCalendar

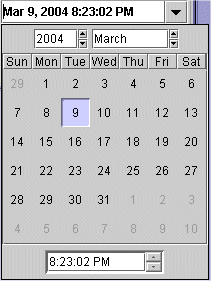
JCPopupCalendar is a component that allows you to edit the date and time using a drop- down calendar. In its editable form, the popup calendar displays a text field with a button next to it. Pushing on the button pops down a calendar from which a date and time can be selected. By default, the calendar has spinboxes for the year, month, and time along with a table which displays the days of the month. The day table updates each time the year and month are changed with the mouse clicks. The time spinbox allows editing of the hour, minute, second, and meridiem.

JCPopupCalendar is an extension of JComboBox. Instead of selecting an item from a drop- down list, the user selects a date/time value using a popup calendar editor.

JCPopupCalendar uses a JFormattedTextField that is configured to edit dates as its text editor. The text editor's value is kept in sync with the popup calendar editor's value, so changing one will automatically update the other. As with JComboBox, JCPopupCalendar is non-editable by default. In this case, the text field is replaced with a button which when selected activates the popup calendar editor. The popup calendar editor can still change the value in the non-editable case.

Note: This component can only be used with JDK 1.4 and above. Those using JDKs prior to JDK 1.4 can use JCPopupField which is a part of JClass Field.

##### 35



*Figure 4 A sample popup calendar.*

The default value for a JCPopupCalendar component is the current date and time.

#### Classes

The pertinent classes to JCPopupCalendar are:

|  |  |
| --- | --- |
| JCPopupCalendar | Creates the JCPopupCalendar component. |
| JCPopupCalendarEditor | Interface that the popup editor must implement. |
| JCPopupCalendarBeanInfo | Contains the JCPopupCalendar bean information. |
| DateTimeChooser | Creates the component that the popup calendar editor displays to the user.  This contains the JCDateChooser and TimeSpin components that are manipulated by the user to select a new date and/or time. |
| DateTimePopup | Creates the actual calendar popup editor. This class is responsible for implementing the  JCPopupCalendarEditor interface, containing the DateTimeChooser, and communicating with the actual popup object. |
| JCDateChooser | Allows the date to be edited. This component can be configured to use various formats. For more information, see [Features of JCDateChooser, in](#_bookmark87) [Chapter](#_bookmark87) 3. |

|  |  |
| --- | --- |
| TimeSpin | Allows the time to be edited with a spinner. |
| JCPopupListener | Responds to events that happen in the popup. |
| JCPopupEvent | Gets passed to JCPopupListeners when the value is committed from the popup calendar to JCPopupCalendar. |

#### Properties

|  |  |  |
| --- | --- | --- |
| **Method** | **Type** | **Description** |
| calendarType | int | Calendar type.  This must be one of the following:  JCPopupCalendar.DATE\_TIME (default): indicates that both the date and time can be edited.  JCPopupCalendar.DATE: indicates that only the date can be edited. |
| hidePopupOnDayTableClick | boolean | If the value is TRUE, the popup calendar editor will pop down when a day is selected.  If the value is FALSE (default), the day can be changed without the editor popping down, and the calendar will pop down when it is double-clicked. |
| maximumDate | java.util.Date | Maximum date value.  If a date is provided, the popup calendar cannot be set later than that value.  Default is null, meaning that there is no maximum. |
| minimumDate | java.util.Date | Minimum date value.  If a date is provided, the popup calendar cannot be set earlier than that value.  Default is null, meaning that there is no minimum. |

|  |  |  |
| --- | --- | --- |
| popupEditor | JCPopupCalendarEditor | Current calendar popup editor. Default is DateTimePopup.  The DateTimeChooser is retrieved  through this property, allowing for other properties to be set (for example, chooserType).  **Note:** It is not recommended that you  build your own popupEditor, but that you customize the one that is provided. |
| showApplyButton | boolean | If the value is TRUE, an Apply button is available on the calendar which, when selected, will commit the current value and will pop down the editor. This may be useful if the hidePopupOnDayTableClick property is FALSE, but it is still desirable to provide another way to dismiss the popup, other than double clicking on the day table.  If the value is FALSE (default), the Apply button is not available. |
| showPopupOnUpDownArrow | boolean | If the value is TRUE (default), the calendar popup editor will pop up when the up or down arrow is selected in the text editor.  If the value is FALSE, the calendar popup editor will not pop up when the up or down arrow is selected in the text editor. |
| value | java.util.Date | Current value of the component. Defaults to the current date and time. |

#### Constructors and Methods

###### JCPopupCalendar Constructors

JCPopupCalendar’s constructor constructs a popup calendar, where the default date and time can be configured, as well as the locale and calendar type.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JCPopupCalendar() | Constructs a JCPopupCalendar.DATE\_TIME calendar type, with the current date, the current time, and the default locale selected. |

|  |  |
| --- | --- |
| public JCPopupCalendar (int calendarType) | Constructs a JCPopupCalendar of the given calendar type with the current date, the current time, and the default locale selected. |
| public JCPopupCalendar (Date d) | Constructs a JCPopupCalendar.DATE\_TIME calendar type, with the default locale and provided date and time. |
| public JCPopupCalendar (int calendarType, Date d) | Construct a JCPopupCalendar of the given type with the default locale and provided date and time. |
| public JCPopupCalendar (Date d, Locale l) | Constructs a JCPopupCalendar.DATE\_TIME calendar type, with a specified locale and provided date and time. |
| public JCPopupCalendar (int calendarType, Date d, Locale l) | Constructs a JCPopupCalendar of the calendar type, with a specified locale and provided date and time. |

#### Listeners and Events

The JCPopupListener listens for JCPopup events, which are generated when the calendar popup editor’s value is committed to JCPopupCalendar and the popup is popped down. JCPopupEvent has the following methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| getSource() | The source of the event which is the DateTimePopup object. |
| getNewValue() | The new value to be committed. |

#### Examples

Please refer to examples.elements.CalendarPopup.java to see a working popup calendar, or refer to examples.elements.CalendarDialog.java to see a how to use the DateTimeChooser component in a dialog editor.

The following code produces a screen with three possible popup calendars: one in English, one in French, and one in Spanish.

import com.klg.jclass.swing.JCPopupCalendar; import com.klg.jclass.util.swing.JCExitFrame; import com.klg.jclass.util.swing.JCAlignLayout; import com.klg.jclass.util.JCEnvironment; import javax.swing.\*;

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

import java.util.Locale; import java.util.Date;

public class CalendarPopup extends JPanel { protected JCPopupCalendar popup1, popup2, popup3; public CalendarPopup()

{

// Set the layout setLayout(new BorderLayout());

// Place all the popup fields in a panel JPanel p = new JPanel();

add(p, BorderLayout.CENTER);

JCAlignLayout mgr = new JCAlignLayout(2, 3, 3); p.setLayout(mgr);

p.setBorder(new TitledBorder("JClass Elements JCCalendarPopup"));

//

// Example of a Date/Time JCPopupCalendar in English

//

Locale locale = new Locale("en", "US");

popup1 = new JCPopupCalendar(JCPopupCalendar.DATE\_TIME, new Date(), locale);

popup1.setEditable(true);

Component c = popup1.getEditor().getEditorComponent(); if (c instanceof JTextField) { ((JTextField)c).setColumns(15);

}

p.add(new JLabel("Date Time Editor (English): ")); p.add(popup1);

mgr.setResizeWidth(popup1, true);

//

// Example of a Date JCPopupCalendar in French

//

locale = new Locale("fr", "FR");

popup2 = new JCPopupCalendar(JCPopupCalendar.DATE, new Date(), locale); popup2.setEditable(true);

p.add(new JLabel("Date Editor (French): ")); p.add(popup2);

mgr.setResizeWidth(popup2, true);

//

// Example of a non-editable Date/Time JCPopupCalendar in Spanish.

//

locale = new Locale("es", "ES");

popup3 = new JCPopupCalendar(JCPopupCalendar.DATE\_TIME, new Date(), locale);

popup3.setEditable(false);

p.add(new JLabel("Date Time Editor (Spanish): ")); p.add(popup3);

mgr.setResizeWidth(popup3, true);

}

public static void main(String[] args)

{

if (JCEnvironment.getJavaVersion() < 140) { System.err.println("\nThis example is incompatible " +

"with JDKs prior to 1.4.0

System.exit(1);

}

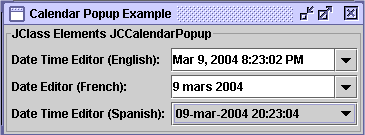
JCExitFrame frame = new JCExitFrame("JCPopupCalendar Examples"); CalendarPopup t = new CalendarPopup(); frame.getContentPane().add(t);

frame.pack();

frame.show();

}

}



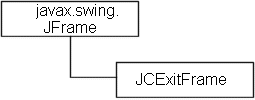
# 5

## Exit Frame

[*Features of JCExitFrame*](#_bookmark145)[*Properties*](#_bookmark147)[*Methods and Constructors*](#_bookmark150)[*Examples*](#_bookmark155)

#### Features of JCExitFrame

A subclass of JFrame that listens for window close events and exits the application when the event is received, or hides the window so that it can be made visible later on. There is a JFrame constant in JavaTM2 v1.3 called EXIT\_ON\_CLOSE that performs the same function.



It is useful for applications containing a single frame. If you used the utility frames available in JClass 3.x versions of *jclass.contrib*, it is useful to know that this replaces DemoFrame and ContribFrame.

#### Properties

A JCExitFrame has the same properties as a JFrame, and one additional one:

exitOnClose A Boolean property that determines whether the application should exit when the user closes the frame or when close() is called (default: true). If set to false, the frame is hidden; it can be made visible later.

Note: Compare this to using JFrame.EXIT\_ON\_CLOSE in JDK 1.3,

which performs the same function.

For a full listing of the properties, please see [Appendix A, Bean Properties Reference](#_bookmark797).

##### 43

#### Methods and Constructors

###### Methods

JCExitFrame subclasses from JFrame, making it a JFrame with a built-in mechanism for catching window-closing events. The following methods report or control which action is taken when a window-closing event is received.

|  |  |
| --- | --- |
| getExitOnClose() | Returns false if the window will be hidden rather than exiting when a window-closing event is received. |
| setExitOnClose() | A Boolean method that determines whether the application should exit when the user closes the frame or when close() is called (default: true). If set to false, the frame is hidden; it can be made visible later. |

###### Constructors

There are two constructors. The default constructor provides an untitled frame while the other accepts a parameter which is used to set the frame’s title.

|  |  |
| --- | --- |
| JCExitFrame() | Default constructor. |
| JCExitFrame(String title) | The parameter provides a title for the frame. |

#### Examples

Use a JCExitFrame as you would a JFrame, and manage window closing events using the

exitOnClose property.

import java.awt.Font;

import javax.swing.JScrollPane; import javax.swing.JTextArea; import com.klg.jclass.util.swing.\*;

public class ExitFrameExample {

static String message0 = "Many JClass examples and demos

use a JCExitFrame."; static String message1 = "\n\nKeep in mind that you can

hide a JCExitFrame \nrather than disposing of it entirely."; static String message = message0 + message1;

static JTextArea messageArea = new JTextArea(message);

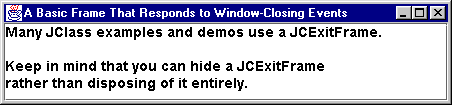
public static void main(String[] args){

String title = "A Basic Frame That Responds to Window-Closing Events"; JCExitFrame frame = new JCExitFrame(title);

messageArea.setFont(new Font("Times-Roman", Font.BOLD, 14)); frame.getContentPane().add(new JScrollPane(messageArea), "Center"); frame.setExitOnClose(false); // Hide the window instead of closing it. frame.setSize(450, 100);

frame.center(); frame.setVisible(true);

}



*Figure 5 A JCExitFrame containing a JTextArea.*

Chapter 5 Exit Frame **45**

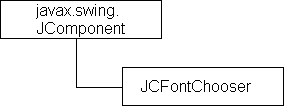
# 6

## Font Choosers

[*Features of JCFontChooser and its Subclasses*](#_bookmark159)[*Classes*](#_bookmark162)[*Properties*](#_bookmark169)[*Methods*](#_bookmark173)[*Examples*](#_bookmark176)

#### Features of JCFontChooser and its Subclasses

JCFontChooser is the abstract base class for JCFontChooserBar and JCFontChooserPane. It provides common data and methods for both components.



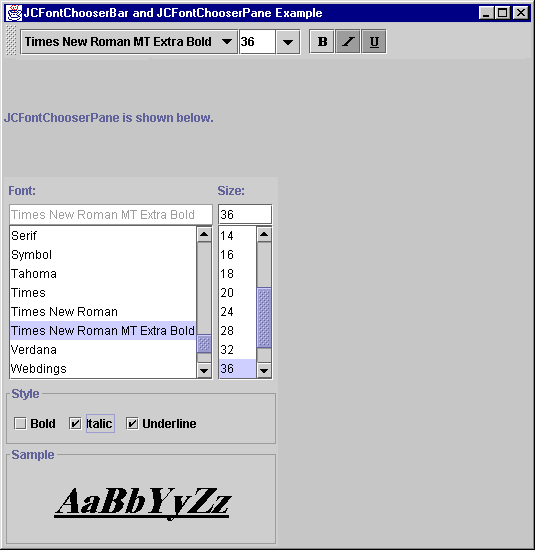
Constructors let you specify what the default fonts and sizes are, as well as letting you set whether underlining is on.

JCFontChooserPane — provides a pane of controls designed to allow a user to manipulate and select a font. It is suitable for use in a tab pane or a dialog window. JCFontChooserPane includes a preview area with sample text.

JCFontChooserBar provides a pane of controls designed to allow a user to manipulate and select a font. It is suitable for use in a JToolbar.

Like the standard Swing components, JCFontChooserBar provides for the optional use of a *Tool Tip*.

##### 47



*Figure 6 A JCFontChooserBar and a JCFontChooserPane.*

#### Classes

|  |  |
| --- | --- |
| JCFontEvent | Used to inform listeners that the font has been changed. Its constants are:  JCFontEvent.FONT\_NAME\_CHANGE, JCFontEvent.FONT\_SIZE\_CHANGE, JCFontEvent.BOLD\_STYLE\_CHANGE, JCFontEvent.ITALIC\_STYLE\_CHANGE, JCFontEvent.UNDERLINE\_STYLE\_CHANGE. |
| JCFontChooserBar | A GUI component suitable for a menu bar. |
| JCFontChooserPane | A GUI component suitable for a dialog or a tabbed pane. |
| JCFontListener | The listener interface. Methods are fontChanging and  fontChanged. |
| JCFontAdapter | A convenience class that provides empty implementations of the listener interface’s methods. |

#### Properties

###### Properties of JCFontChooserBar and JCFontChooserPane

|  |  |
| --- | --- |
| toolTipEnabled | A Boolean property that indicates whether Tool Tips are being used. The get method is called isToolTipEnabled. |
| selectedFont | The set method of this property has three different signatures: a single parameter Font font, a two parameter version, Font font, boolean underline, and a version for setting every font-related parameter, String name, int style, int size, boolean underline. |

For a full listing of the properties, please see [Properties of JCFontChooserBar](#_bookmark799) and [Properties of JCFontChooserPane](#_bookmark801) in [Appendix A](#_bookmark776).

#### Methods

Because the initial choice of font parameters is made in the constructor, and subsequent changes are made by interacting with the GUI, there are no public methods of interest in JCFontChooserBar or JCFontChooserPane. Only the listener methods need concern you.

You listen for font changes by implementing the JCFontListener interface. Its two methods are fontChanging(), and fontChanged(). Both methods take a JCFontEvent parameter. Use the first method to inspect and possibly veto the change in font, or in the underline state. Use the second to notify of these changes.

A JCFontEvent contains information about its source, the type of change that was made, old and new Font values, old and new underline values, and a Boolean fontChanging parameter that indicates whether this is a vetoable change or not.

Note: The “old” font and underline values are read-only.

#### Examples

In this example we’ll add both a JCFontChooserBar and a JCFontChooserPane to the same panel. Normally, you place a JCFontChooserPane in its own dialog, but adding it to a JPanel as is done here doesn’t change the way JCFontChooserPane’s properties are set. The code snippet shows how to instantiate both components and how to add a JCFontListener so you can respond to font-changed events. Since the listening object is JCFontExample, it needs to provide an implementation of fontChanged, the method that is declared in interface JCFontListener. Note that JCFontChooserBar is added to a

JToolbar object.

Chapter 6 Font Choosers **49**

public class JCFontExample extends JPanel implements JCFontListener {

...

bar = new JToolBar();

Font font3 = new Font("Serif", Font.PLAIN, 12); font3 = JCFontChooser.setUnderline(font3, true);

...

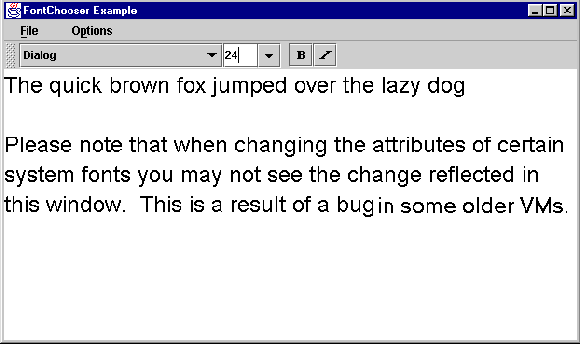
fontBar = new JCFontChooserBar(font3); bar.add(fontBar); fontBar.addJCFontListener(this);

...

fontPane = new JCFontChooserPane(font3); fontPane.addJCFontListener(this);

...

}



*Figure 7 A bug notice in a JCFontChooserPane.*

# 7

## HTML/Help Panes

[*Features of JCHTMLPane*](#_bookmark180)[*Features of JCHelpPane*](#_bookmark181)[*Classes*](#_bookmark183)[*Properties*](#_bookmark185)

[*Constructors and Methods*](#_bookmark187)[*Examples*](#_bookmark201)

#### Features of JCHTMLPane

JCHTMLPane is a subclass of Swing’s JEditorPane which has been hard-coded to use the HTML Editor kit. HTML display can be as simple as passing the HTML code to JCHTMLPane’s setText() method. Alternatively, you can pass the text as a parameter to the constructor. This class also implements a Hyperlink listener to implement link traversal and different cursor images (hand cursor and wait cursor).

JCHTMLPane is an extension of JEditorPane that lets you: Construct an HTML pane, given a URL.

Construct an HTML pane, given a pointer to HTML text. Change the icon for the cursor when it is over a link.

Follow the reference in a link.

Use an MDIMenuBar and MDIToolBar in addition to a JMenuBar.

Note that the HTML functionality in Swing’s JEditorPane is based on javax.swing.text.html.HTMLEditorKit, which supports most, but not all, HTML 3.2 tags. The APPLET tag is not supported (March, 2000), and care should be taken when using OBJECT, SCRIPT, FRAME, and dynamic HTML.

#### Features of JCHelpPane

JCHelpPane is an extension of JCHTMLPane in that it contains two JCHTMLPanes under a header pane. A typical use places an HTML page containing a title in the header pane, a table of contents page on the left, and a contents page on the right. You can use it to provide your users with a lightweight browser for a HTML-based help facility.

The lightweight browser becomes part of your application.

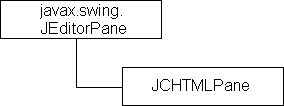
Once links have been followed, forward and back buttons allow users to retrace their steps.

##### 51

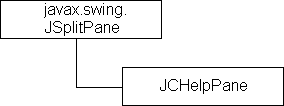
JCHelpPane checks to see if a URL for the title pane was specified. If it wasn't, the title pane is not shown.

#### Classes

JCHTMLPane provides all the functionality necessary for an HTML-based pane, while JCHelpPane implements a lightweight two- or three-paned help system. Both of these are JavaBeans.



*Figure 8 JCHTMLPane inherits from JEditorPane.*



*Figure 9 JCHelpPane inherits from JSplitPane.*

#### Properties

JCHTMLPane’s properties are the same as JEditorPane’s. The class behaves like a

JEditorPane with extra HTML awareness.

For a full listing of JCHTMLPane’s properties, see [Appendix A, Bean Properties Reference](#_bookmark804).

#### Constructors and Methods

##### Constructors

###### Constructors for JCHTMLPane

Along with the parameterless constructor for creating a blank pane, two others provide a handy way of instantiating a pane and providing it with HTML content in one operation.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JCHTMLPane() | Constructs a blank HTML pane. |
| JCHTMLPane(URL url) | Constructs an HTML pane with the specified URL. |
| JCHTMLPane(String text) | Constructs an HTML pane with the specified HTML text. |

###### Constructors for JCHelpPane

JCHelpPane’s constructors let you specify source pages using URLs or Strings. The latter may be advantageous if you generate some HTML-formatted text dynamically.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JCHelpPane() | Constructs a single blank “contents” pane. |
| JCHelpPane(URL contents, URL view) | Constructs, from the specified URLs, a help screen with a contents pane on the left and a view pane on the right. Note that Strings may be used in place of URLs. |
| JCHelpPane(URL contents, URL view, URL title) | Constructs, from the URLs, a help screen with three frames: a header frame that spans the top of the window, and two side-by-side frames underneath. Note that Strings may be used in place of URLs. |

##### Methods

###### JCHTMLPane

The method of note is setText(), which is inherited from JEditorPane. Use it to pass text with embedded HTML tabs to the JCHTMLPane. An alternative way to pass the text is to via the pane’s constructor, described above.

###### JCHelpPane

Although it is possible to construct a help browser using just the constructors for

JCHelpPane, it has a number of methods that may help your construction:

|  |  |
| --- | --- |
| getContentsPage() setContentsPage() | Gets or sets the contents page. That is, get or set the HTML pane on the left hand side. |
| getContentsPane() | Returns the HTML pane on the left hand side. |
| getTitlePage() setTitlePage() | Gets or sets the title page. That is, gets or sets the HTML pane at the top. |
| getViewPage() setViewPage() | Gets or sets the view page. That is, gets or sets the HTML pane on the right hand side. |
| getViewPane() | Returns the HTML pane on the right hand side. |
| isUseToolBar() setUseToolBar() | Gets or sets the value of useToolBar.  If the set method returns true, the component traverses up the tree to find its root pane container and adds a tool bar to it if one does not exist. If one exists, it adds the HTML navigation buttons to the existing toolbar. If two buttons exist in the tool bar named Back and Forward, then it will not add the buttons, but rather add listeners to those buttons. |

#### Examples

###### JCHTMLPane

The following incomplete code fragment shows how you can compose your HTML text dynamically, then pass it to an instance of JCHTMLPane. The result is shown in the accompanying figure.

String myHTMLText = "<HTML><HEAD><TITLE>JCHTMLPane Demo</TITLE></HEAD>"; myHTMLText += "<BODY><B>HTML (Bold) <P> <H1>JCHTMLPane

understands basic HTML tags,</H1>"; myHTMLText += "such as headings:";

myHTMLText += "<H2 COLOR=red>A second level heading.</H2>"; myHTMLText += "<H3 COLOR=blue><EM>And lists:</EM></H3><BR>"; myHTMLText += "<OL><LI>Life is like a box of choco-lates"; myHTMLText += "<LI>Judy, Judy, Judy";

myHTMLText += "<LI>Play it again, Sam</OL>"; myHTMLText += "<A HREF=\"http://www.quest.com\">

And links to other Web pages</A>"; myHTMLText += "<P>Tables too!<TABLE BORDER=10

BORDERCOLOR=BLACK BGCOLOR=WHITE>";

myHTMLText += "<tr><td>ROW ONE, First COLUMN cell</TD>

<TD>ROW ONE, Second COLUMN Cell</TD>

<TD>ROW ONE, Third COLUMN cell</TD></TR>";

myHTMLText += "<tr><td>ROW TWO, First COLUMN cell</TD>

<TD>ROW TWO, Second COLUMN Cell</TD>

<TD>ROW TWO, Third COLUMN cell</TD></TR>";

myHTMLText += "<tr><td>ROW THREE, First COLUMN cell</TD>

<TD>ROW THREE, Second COLUMN Cell</TD>

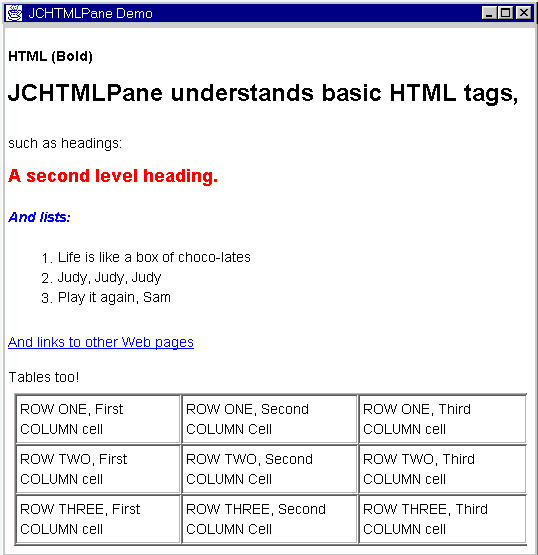
<TD>ROW THREE, Third COLUMN cell</TD></TR>"; myHTMLText += "</TABLE>";

myHTMLText += "</BODY></HTML>";

JCHTMLPane pane = new JCHTMLPane(myHTMLText); pane.setEditable(false);

pane.setVisible(true); frame.getContentPane().add(pane, BorderLayout.SOUTH);

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



*Figure 10 A JCHTMLPane whose contents are derived from HTML Strings in the class.*

###### JCHelpPane

This example demonstrates instantiating JCHelpPanes with both Strings and URLs. If any of the URLs can’t be found, the version of JCHelpPane that uses Strings is displayed.

import com.klg.jclass.util.swing.\*; import com.klg.jclass.util.value.\*; import javax.swing.\*;

import java.awt.\*;

/\*\*

\* This example demonstrates the use of a JCHelpPaneExample

\*/

public class HelpPaneExample {

// All the work is done in main()

public static void main(String args[]) { String contents = new String(

"The contents pane of the JCHelpPane if URL isn’t found."); String view = new String(

"The view pane of the JCHelpPane if URL isn’t found."); String title = new String("Header for the Help Pane."); JFrame frame = new JCExitFrame("Help Pane Example"); JCHelpPane app = new JCHelpPane(contents, view, title);

try {

java.net.URL contentsFromURL = new

java.net.URL("http://. /toc\_page.html");

java.net.URL viewFromURL = new

java.net.URL("http://. /readme.html");

java.net.URL titleFromURL = new

java.net.URL("http://. /jclasslogo.html");

app = new JCHelpPane(contentsFromURL, viewFromURL,

titleFromURL);

}

catch (java.net.MalformedURLException e) { System.out.println("Malformed URL");

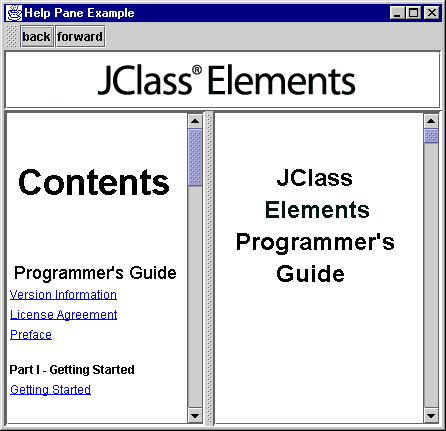
}

app.setPreferredSize(new Dimension(640, 400)); frame.getContentPane().add(app);

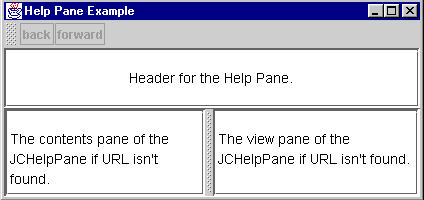
frame.pack(); frame.setSize(700, 450); frame.show();

}

}



*Figure 11 A JCHelpPane showing the HTML version of this manual.*



*Figure 12 In this example, the alternate JCHelpPane when the URL can’t be found.*

# 8

## Sortable Table

[*Features of JCMappingSort*](#_bookmark205)[*Features of JCSortableTable*](#_bookmark207)[*Classes and Interfaces*](#_bookmark210)[*Constructors and Methods*](#_bookmark220)[*Cell Renderers for JCSortableTable*](#_bookmark237)[*Examples*](#_bookmark239)

#### Features of JCMappingSort

Sorting can be accomplished by indexing the list of objects that are going to be ordered according to some comparison policy. It can be much more efficient to sort these indices instead of sorting the objects themselves. The idea is to form an array of indices. Initially, a[1] = 1, a[2] = 2, and so on, up to *n*, the length of the list. After sorting, the result might be a[1] = 9, a[2] = 3, ... a[n-1] = 1, ... a[n] = 7, where now the index in a[1] corresponds to the object that is the smallest element in the list according to the supplied comparison rule. The index in a[2] corresponds to the next smallest object, and so on. The list hasn’t changed, but the array supplies a mechanism for traversing the list according to some ordering principle.

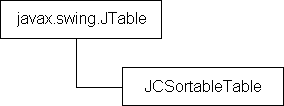
The foregoing paragraph shows you that if you want to use this type of mapping sort in your application, you’ll need to supply an array of indices and a comparator to use with your list. In some cases, a comparator is already available. A number of common objects implement the Comparable interface in Java 2. You can compare any of these types without needing to supply an explicit comparator.

JCMappingSort provides a sort() method, which takes an implementation of JCIntComparator and an array of indices as parameters, and modifies the passed-in array based on the compare() method defined by your implementation of JCIntComparator.

#### Features of JCSortableTable

JCSortableTable uses a comparator and a configureable list of column indices, making this class useful for establishing a sort policy that specifies what should be done when two elements in the primary column have the same value. Elements in the primary column that compare the same are arranged among themselves by sorting the secondary column. The process can be continued as necessary by including more columns in the list.

##### 59



Each column in a table may have an associated list of columns that are to be used as sort keys. Normally, the column itself is specified as the primary sort key.

You can set whether a table is re-sorted automatically when its data changes.

You can set or toggle the sorting order, permitting sorting from less to greater, or from greater to less.

Sort Dates, Objects that implement Comparable, and wrapped primitive types.

You can provide your own implementation of JCRowComparator to perform row comparisons in the sort algorithm. See the example at the end of this chapter for details.

#### Classes and Interfaces

###### JCMappingSort

|  |  |
| --- | --- |
| CollectionIntComparator | Implements JCIntComparator to compare two lists. |
| JCIntComparator | An interface that declares a compare method taking two indices as parameters. The compare method must be able to compare the Objects corresponding to the indices. |
| JCMappingSort | Contains a static sort() method that is passed a JCIntComparator and an array of indices. The array containing the indices is sorted rather than sorting the list objects to which they refer. |

You’ll find these classes and interfaces in com.klg.jclass.util.

###### JCSortableTable

|  |  |
| --- | --- |
| JCSortableTable | JCSortableTable is a subclass of JTable that internally wraps any TableModel it is given with a JCRowSortTableModel and provides a Comparator that has a configureable list of the column indexes that it uses for sorting. Clicking on a column header invokes the sorting behavior tied to that column, clicking again reverses the direction of the sort. |
| JCRowComparator | This interface is to be used with JCRowSortModel. It sorts rows using a specified ordered list of columns as the sort keys. By default, it sorts on the first column. |
| JCRowSortModel | An interface that defines methods for sorting rows by specifying which columns are to be used as keys. |

**Using your own comparator with** **JCSortableTable**

If you wish to provide your own comparator for a JCSortableTable, follow these steps:

* + 1. Create a javax.swing.table.TableModel.
    2. Create a com.klg.jclass.util.swing.DefaultRowSortTableModel, giving it the TableModel.
    3. Set your comparator to this instance of a DefaultRowSortTableModel.
    4. Set the DefaultRowSortTableModel on your JCSortableTable.

Note that the data model you set in step 2 should be a JCRowSortTableModel. If it is not,

JCSortableTable will wrap the data model you provide with a JCRowSortTableModel.

#### Constructors and Methods

###### Constructors for JCSortableTable

|  |  |
| --- | --- |
| JCSortableTable() | JCSortableTable is a subclass of JTable that internally wraps any TableModel it is given with a JCRowSortTableModel and provides a Comparator that has a configureable list of the column indexes that it uses for sorting. |
| JCSortableTable( int numRows, int numColumns) | Constructor that specifies the number of rows and columns in the table. |

|  |  |
| --- | --- |
| JCSortableTable( Object[][] rowData, Object[] columnNames) | The constructor for a data source composed of an array of Objects. |
| JCSortableTable( TableModel dm) | Constructor that accepts a TableModel. |
| JCSortableTable( TableModel dm, ColumnModel cm) | Constructor that accepts both a ColumnModel and a  TableModel. |
| JCSortableTable( TableModel dm, ColumnModel cm, ListSelectionModel sm) | Constructor that accepts a ColumnModel, a TableModel, and a ListSelectionModel. |
| JCSortableTable( Vector rowData, Vector columnNames) | The constructor for a Vector data source. |

The core of the sorting mechanism is based on providing the sort() method with a list of indices specifying an ordered list of columns on which the sort is to be based:

public static void sort(JCIntComparator comparator, int indices[])

public static void sort(JCIntComparator comparator, int indices[],

int start, int end)

Both methods require a JCIntComparator and an array of indices. The second method includes two additional parameters that are useful in many sorting algorithms.

###### Methods

In addition to the host of methods it inherits from JTable, JCSortableTable adds many of its own:

|  |  |
| --- | --- |
| createDefault ColumnsFrom Model() | Overridden from the superclass to allow auto-creation of our own column model. |
| getAutoSort() | Returns whether the data is automatically sorted when it changes according the current comparator. |
| getCellEditor() | Takes parameter int row, int column to get the cell editor for that row and column. |
| getCellRenderer() | Takes parameter int row, int column to get the cell editor for that row and column. |
| getKeyColumns() | Takes parameter int column to return the key columns used to sort the table model when clicking on the specified column. |

|  |  |
| --- | --- |
| getUnsortedRow() | Takes parameter int sortedRow to return the unsorted row index of specified sorted row. |
| setAutoSort() | Takes parameter boolean autoSort to specify whether the data should be automatically sorted when it changes. |
| setKeyColumns() | Takes parameters int column, int[] keyColumns to set the key columns used to sort the table model when clicking on specified column. |
| setModel() | Takes parameter javax.swing.table.TableModel newModel to set the data model for this table to newModel and registers with for listener notifications from the new data model. |
| setTableHeader() | Takes parameter javax.swing.table.JTableHeader newHeader to overwrite the default implementation and add a MouseListener to the new table header. |
| sort() | Takes parameter int column to sort rows using the quicksort algorithm. |
| tableChanged(e) | Uses parameter javax.swing.event.TableModelEvent e to pass information about the event. Overrides super class method to check for a change in sorting. |
| unsort() | Restores the unsorted order. |

#### Cell Renderers for JCSortableTable

Normally, you do not need to be concerned with the details of how table cells are rendered because renderers for most common cases have already been supplied. On the other hand, you may wish to use a custom renderer of your own design. While it is possible to use setDefaultRenderer() to set a cell renderer for a JTable, the method is not available for use with JCSortableTable. Instead, JClass uses its own powerful cell editor/renderer mechanism. This allows all JClass products to manage collections of JCCellRenderer types uniformly instead of having to manage the renderer types separately. To set your own cell renderer, use JClass Cell’s EditorRendererRegistry, and implement one of the renderer interfaces. Please see the *com.klg.jclass.cell API* for details.

#### Examples

###### JCMappingSort example

JCMappingSort cannot be instantiated by calling its constructor. Instead, it has two static methods of the form:

public static void sort(JCIntComparator compartor, int indices[]); public static void sort(JCIntComparator compartor, int indices[],

int start, int end);

The purpose of these two methods is to sort a mapping of indices instead of an array of objects. This is particularly useful when dealing with a Collection, or some form of data model where you reference a data element with an index. Your implementation of the JCIntComparator interface provides the implementation details for the objects you are sorting.

JCIntComparator should look like this:

public interface JCIntComparator {

public int compare(int index1, int index2);

}

The CollectionIntComparator is a specific implementation of JCIntComparator that can compare Collections. Sample code looks like this:

public class CollectionComparator implements JCIntComparator {

protected Collection collection; protected Comparator comparator;

public CollectionComparator(Collection collection, Comparator

comparator) { this.collection = collection; this.comparator = comparator;

}

public CollectionComparator(Collection collection) { this(collection, null);

}

public int compare(int i1, int i2) { Object a1 = collection.get(i1); Object a2 = collection.get(i2);

if (comparator != null) {

// use comparator if provided return comparator.compare(a1, a2);

}

else if (a1 instanceof Comparable) {

// items are comparable so get them to compare themselves return ((Comparable) a1).compare(a2);

}

else {

// We have no comparator and the objects are not comparable

throw new IllegalArgumentException("Objects are not Comparable; please provide a Comparator with the constructor:

CollectionComparator(Collection collection, Comparator

comparator)");

}

}

}

###### JCSortableTable examples

One use of JCSortableTable is shown in SortTable.java, which is located in *JCLASS\_HOME/examples/elements/*. The following code is based on that example. It demonstrates sorting on columnar data containing Strings, and two types of primitives: Boolean values and integers. The example provides its own implementation of JCRowComparator to perform a comparison between two rows in the table.

import java.awt.BorderLayout; import java.awt.Container; import java.awt.Dimension;

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

import java.util.GregorianCalendar; import java.util.Comparator; import java.text.\*;

import javax.swing.JButton; import javax.swing.JPanel; import javax.swing.JScrollPane; import javax.swing.UIManager;

import javax.swing.table.AbstractTableModel; import javax.swing.BoxLayout;

import javax.swing.table.DefaultTableCellRenderer;

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

import com.klg.jclass.util.swing.DefaultRowSortTableModel; import com.klg.jclass.util.swing.DefaultRowComparator; import com.klg.jclass.util.swing.JCRowSortModel;

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

/\*\*

* Sorting is allowed on these columns:
* "First Name", "Last Name", "Position", "Favorite Number", and
* "Vegetarian"

\*/

public class SortDateJCSortableTable extends JPanel implements ActionListener {

protected final static String[] names =

{"First Name", "Last Name", "Position",

"Favorite Number", "Vegetarian", "Calendar","GregorianCalendar"};

protected final static Object[][] data = {

{"Diana", "Zukerman", "Research Officer", new Integer(1), new Boolean(false),"",""},

{"Adam", "Petersen", "Consultant",

new Integer(2), new Boolean(false),"",""},

{"Mary", "Binfield", "Research Associate", new Integer(5), new Boolean(false),"",""},

{"Michael", "Rizzo", "Research Fellow", new Integer(2), new Boolean(true),"",""},

{"Ahmad", "Baldi", "Consultant",

new Integer(3), new Boolean(false),"",""},

{"Ian", "Clemente", "Research Fellow",

new Integer(7), new Boolean(false),"",""},

{"David", "Rubinstein", "Consultant",

new Integer(4), new Boolean(false),"",""},

};

protected JButton buttonUnsort = null; protected JCSortableTable sortableTable = null;

/\*\* Indicates that the first object is less than the second object.

\*/

public static final int LESS\_THAN = -1;

/\*\* Indicates that the first object is equal to the second object. \*/ public static final int EQUAL = 0;

/\*\* Indicates that the first object is greater than the second object. \*/ public static final int GREATER\_THAN = 1;

public SortDateJCSortableTable() {

// Set a simple BoxLayout manager

setLayout(new BoxLayout(this,BoxLayout.X\_AXIS));

//set up the calender values to be tested for (int r =0 ; r < data[0].length ; r++){

Calendar c = Calendar.getInstance(); c.set(1998+r, r, 1);

GregorianCalendar gc = (GregorianCalendar)c; data[r][5] = c;

data[r][6] = gc;

}

// Create and add the table sortableTable = createTable(); add(new JScrollPane(sortableTable));

// Create and add an Unsort button for the table buttonUnsort = new JButton("Unsort"); buttonUnsort.addActionListener(this);

add(buttonUnsort);

}

public static JCSortableTable createTable() { EditableTableModel model = new EditableTableModel();

JCSortableTable table = new JCSortableTable();

// JCSortTable will do this anyway,

// but this way we have a member handle to it. DefaultRowSortTableModel mRSmodel =

new DefaultRowSortTableModel(model); mRSmodel.setComparator(new MyComparator());

//set model and cast it down to the "DefaultRowSortTableModel" table.setModel(mRSmodel);

// We use the last name if the first name is the same. int sort0[] = {0, 1};

table.setKeyColumns(0, sort0);

// We use the first name if the last name is the same. int sort1[] = {1, 0};

table.setKeyColumns(1, sort1);

// We use person's name if the department is the same. int sort2[] = {2, 0, 1};

table.setKeyColumns(2, sort2);

//set the non primitive renderers, no editor defined for this example

table.getColumn("GregorianCalendar").setCellRenderer( new CGFCalendarCellRenderer());

table.getColumn("Calendar").setCellRenderer( new CGFCalendarCellRenderer());

return table;

}

public static void main(String args[]) { JCExitFrame frame = new JCExitFrame(

"SortDateJCSortableTable Example"); SortDateJCSortableTable app =

new SortDateJCSortableTable();

if (args.length > 0) {

if (args[0].equals("windows")) { try { UIManager.setLookAndFeel(

"com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

} catch (Exception e) {}

}

}

frame.getContentPane().add(app); frame.setBounds(50, 50, 650, 350); frame.show();

}

//===== ActionListener interface method ======== public void actionPerformed(ActionEvent e) {

if (e.getSource() instanceof JButton) { sortableTable.unsort();

}

}

public static class CGFCalendarCellRenderer extends DefaultTableCellRenderer

{

protected java.text.DateFormat date\_formatter = java.text.DateFormat.getDateInstance();

public void setValue(Object o)

{

String str = null;

if (o instanceof String) { str = (String)o;

}

else if (o instanceof Calendar) {

str = date\_formatter.format(((Calendar)o).getTime());

}

else {

str = o.toString();

}

super.setValue(o == null ? "" : str);

}

}

private static class EditableTableModel extends AbstractTableModel {

EditableTableModel() { super();

}

public int getColumnCount() { return names.length;

}

public int getRowCount() { return data.length;

}

public Object getValueAt(int row, int col) { return data[row][col];

}

public String getColumnName(int column) { return names[column];

}

public Class getColumnClass(int col) {

return getValueAt(0,col).getClass();

}

// Disallow edits on dates, and on favorite number public boolean isCellEditable(int row, int col) { return col != 3 && col != 5 && col != 6;

}

public void setValueAt(Object aValue, int row, int column) { data[row][column] = aValue;

}

} // EditableTableModel

static class MyComparator extends DefaultRowComparator { public MyComparator(){

super(JCRowSortModel.FORWARD);

}

public int compare(JCComparableRow row1, JCComparableRow row2) { int[] kc = super.getKeyColumns();

for (int i = 0; i < kc.length; i++) { Object o1 = row1.getValueAt(kc[i]); Object o2 = row2.getValueAt(kc[i]);

if(o1 instanceof Calendar && o2 instanceof Calendar) { Calendar c1 = (Calendar)o1;

Calendar c2 = (Calendar)o2;

if(c1.equals(c2)) { return EQUAL;

}

else if(c1.before(c2)) { return LESS\_THAN;

}

else {

return GREATER\_THAN;

}

}

}

return super.compare(row1, row2);

}

}//MyComparator

} // SortDateJCSortableTable

# 9

## Multiple Document Frame

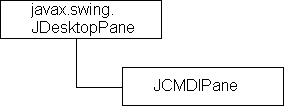
[*Features of JCMDIPane and JCMDIFrame*](#_bookmark246)[*Properties*](#_bookmark252)[*Methods*](#_bookmark254)[*Examples*](#_bookmark290)

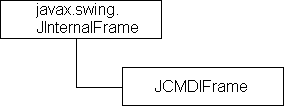
#### Features of JCMDIPane and JCMDIFrame

The *Multiple Document Interface* (MDI) model is a common way of organizing and presenting information in windows-style environments. The MDI model makes it possible to arrange multiple child windows inside a parent window, such as multiple files in a text editor, or multiple charts in one frame. In effect, the parent window becomes the desktop within which the children operate. Before Swing, there was no way of building MDI applications using the Abstract Windowing Toolkit (AWT).

If you were limited to using raw Swing components, you would likely build your primary GUI application within a JFrame. The container used to hold a multiple-document interface is a JDesktopPane, which you would put into the content pane of your JFrame. Finally, you would add JInternalFrames as needed for your document windows.

The JClass Elements components JCMDIPane and JCMDIFrame augment the functionality of JDesktopPane and JInternalFrame respectively. Simply replace JDesktopPane with JCMDIPane, and JInternalFrame with JCMDIFrame, and your job is almost complete! The only other thing you need to do is to use the setMDIMenuBar() method to set individual menu bars on each of your internal frames. These menu bars will replace the default menu bar that you set on JCMDIPane.





##### 71

JClass Elements’s multiple document JCMDIPane interface component extends Swing’s

JDesktopPane view to provide the following standard MDI features:

True maximization. Instead of keeping two menu bars when an internal pane is maximized, JCMDIPane optimizes screen real estate by placing menus on the desktop’s menu bar. All necessary functionality is preserved.

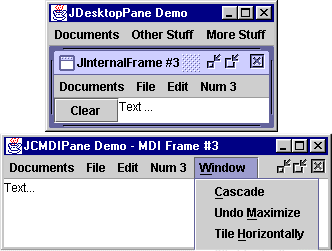
Automatically adds a localized Windows menu containing two sections.

The upper section of the Windows menu allows you to select from one of three window tiling algorithms: Cascade, Tile Horizontally, or Tile Vertically.

The upper section of the Windows menu also allows you to *Minimize/Maximize* the selected frame, or to (re)*Arrange Icons* of the minimized frames.

The lower section of the Windows menu provides a list of the titles of the internal frames, giving the user the ability to switch focus to any internal frame by selecting its name from the menu.

Adds unmaximize/close icons to the far right of the menu bar when one of the frames is maximized.



*Figure 13 The differences between a JCMDIPane (lower image) and a JInternalFrame (upper image).*

###### Public classes:

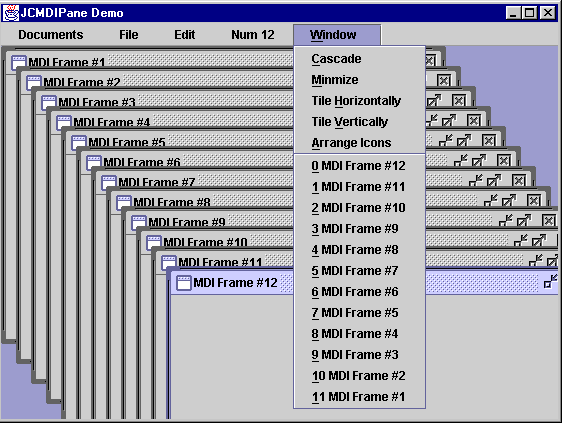
JCMDIPane — subclass of JDesktopPane JCMDIFrame — subclass of InternalFrame

JCMDIPane is API compatible with JDesktopPane, but the behavior differs in that it automatically generates a Windows menu on the first toolbar it finds in its ancestral hierarchy. This Windows menu has arrangement options Cascade, Tile Horizontally, Tile Vertically, and Arrange Icons, and a selectable list of all the existing internal frames. When frames are maximized the first child of an internal frame’s content pane is

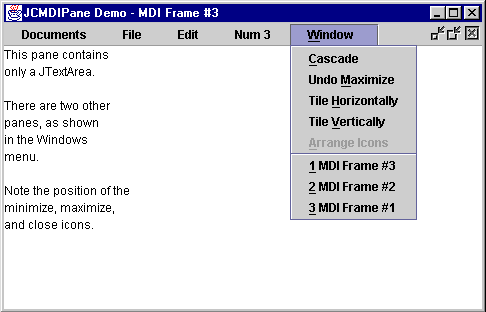
reparented to a panel that is mapped on top of all the frames so that the maximized frame makes maximal use of the existing window real estate.

Default dragging and resizing behavior is done speedily by drawing wire frames.

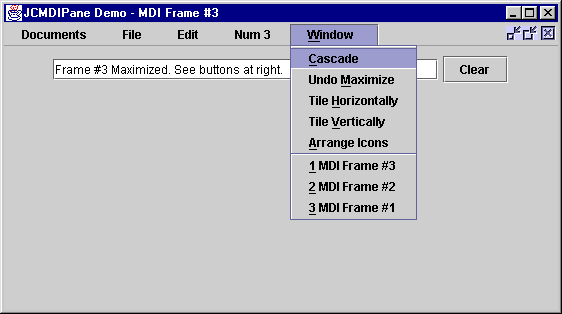
JCMDIFrame, when calling getContentPane, returns an additional child that is the single child of the true content pane. This is done for easy reparenting purposes as well as for support routines that aid in the manipulation of this child.



*Figure 14 A JFrame containing a JCMDIPane and multiple JCMDIFrames.*



*Figure 15 A maximized internal frame—the iconify and close buttons have moved to the menu bar.*



*Figure 16 One frame is maximized. The Iconify, Maximize, and Close buttons appear on the menu bar.*

#### Properties

For a full listing of these components’ properties, see [Properties of JCMDIFrame](#_bookmark782) and [Properties of JCMDIPane](#_bookmark783), in [Appendix A](#_bookmark776).

JCMDIFrame has the same properties as JInternalFrame.

Along with the properties inherited from JDesktopPane, there are two additional properties in JCMDIPane: frameManipulationStyle and considerIconsWhenTiling. Setting frameManipulationStyle allows you to control how a frame is painted when it is dragged within the desktop. The default style, JCMDIPane.DEFAULT, causes JCMDIPane to repaint the entire frame when dragging. The second style, called wireframe, causes JCMDIPane to repaint a rectangle that matches the size of the frame. The wireframe dragging style is used when frameManipulationStyle is set to JCMDIPane.WIREFRAME. The considerIconsWhenTiling property controls the way that windows are tiled (false means that windows will be tiled using the entire desktop area; true means that windows will not be tiled over any icons that appear on the desktop).

#### Methods

###### Methods of JCMDIFrame

|  |  |
| --- | --- |
| getMDIMenuBar() setMDIMenuBar() | Gets or sets the menu bar associated with this frame. If the parent of this frame is a JCMDIPane, then this menu bar will become the containing frame's menu bar when this frame becomes active. setMDIMenuBar() takes a JMenuBar as a parameter. |
| getMDIToolBar() setMDIToolBar() | Gets or sets the toolbar associated with this frame. If the parent of this frame is a JCMDIPane, then this toolbar will become the containing frame's toolbar when this frame becomes active. setMDIToolBar() takes a JToolBar as a parameter. |
| getContentPane() | Overrides getContentPane() to provide a container one level removed so that the frame can be maximized by reparenting its children to a different parent. |

**Methods of** **JCMDIPane**

|  |  |
| --- | --- |
| getAllNonIconifiedFrames() getAllIconifiedFrames() | Returns all non-iconified/iconified JCMDIFrames currently displayed in the desktop. |
| getDragMode() setDragMode() | Sets the dragging style of the frames on the desktop. Because JCMDIPane uses its own Desktop Manager, it does not use the dragging implementation of JDesktopPane; instead, it uses the dragging implementation of frameManipulationStyle.  However, setting this property actually sets the frameManipulationStyle to the equivalent style. Valid styles are:  OUTLINE\_DRAG\_MODE — corresponds to WIREFRAME LIVE\_DRAG\_MODE — corresponds to DEFAULT |
| getFrameManipulationStyle() setFrameManipulationStyle() | Sets the frame manipulation style. Valid styles are: WIREFRAME — drags and resizes as a wire frame, DEFAULT — default style specified by the PLAF you are using. The default style causes JCMDIPane to paint the entire frame when dragging it. |
| setInitialLayout() | Allows the layout of the MDIFrame windows to be set before the MDIPane window has been displayed. This has no effect after the MDIPane has been displayed for the first time. |
| isMaximized() setMaximized() | Methods to manage the maximized pane. |

|  |  |
| --- | --- |
| getMDIMenuBar() setMDIMenuBar() | The parameterless version of getMDIMenuBar() returns the menu bar used if we have no internal frames, whereas a JInternalFrame parameter is used to return the menu bar used for the specified frame. If the frame is a JCMDIFrame with a non-null MDIMenuBar, then this is returned. Otherwise, the MDIMenuBar for the this pane is returned.  setMDIMenuBar() with a JMenuBar parameter sets the toolbar to use if there are no internal frames. |
| getMDIToolBar() setMDIToolBar() | Returns the toolbar used for the specified frame. If the frame is a JCMDIFrame with a non-null MDIToolBar, then this is returned; otherwise, the MDIToolBar for the this pane is returned. getMDIToolBar() takes a JInternalFrame as a parameter.  Sets the toolbar to use if there are no internal frames.  setMDIToolBar() takes a JToolBar as a parameter. |
| getNonSelectedIcon() setNonSelectedIcon() | Gets or sets the non-selected icon, which appears before the non-selected items in the menu. The default is an empty icon that acts as a placeholder so menu items will be aligned properly. Setting both these icons to null restores the previous behavior. |
| getSelectedIcon() setSelectedIcon() | Gets or sets the icon which is to appear beside the selected window item in the Windows menu. The default icon is a check mark. |
| getPreferredSize() | If this pane has an ancestor that's a scroll pane or it has no children, then it returns the default preferred size. If it has children and no scroll pane for an ancestor, then it returns a size big enough to show all its children in their current locations. |
| getTopFrame() | Returns the topmost frame. |
| setInitialLayout() | Allows the layout of the MDIFrame windows to be set before the MDIPane window has been displayed. This has no effect after the MDIPane has been displayed for the first time. If not called, the initial layout is unpredictable.  Pass one of these constants to the method: JCMDIPane.TILE\_HORIZONTAL, JCMDIPane.TILE\_VERTICAL, JCMDIPane.CASCADE. |

|  |  |
| --- | --- |
| activateFrame() arrangeIcons() cascadeWindows() | Makes this frame the active frame.  Arranges “iconified” panes along the bottom. Arranges non-iconified panes in cascade form. |
| closeFrame() | Closes or deactivates a frame. |
| deactivateFrame() |  |
|  | Maximizes a pane, filling the host frame. |
| maximize() |  |
|  | Tiles the frames in the specified direction |
| tileWindowsHorizontally() |  |
| tileWindowsVertically() | Returns the frame to its former size. |
| unmaximize() |  |
|  | Note: These methods were protected in version 4.0  and have been made public in version 4.0.1. |

There are a number of protected methods available to application programmers who wish to subclass a JCMDIPane. Consult the API for a list of these methods.

#### Examples

This code snippet highlights the few things that need to be done to convert your MDI application based on JInternalFrame into one based on JCMDIFrame.

import com.klg.jclass.swing.JCMDIPane; import com.klg.jclass.swing.JCMDIFrame;

/\*\*

* The class extends JClass Elements’ JCExitFrame so you don’t have to
* write repetitive window closing code.

\*/

public class MDIInternalFrameDemo extends JCExitFrame implements

ActionListener {

/\*\*

* The internal frames reside inside a JCMDIPane

\*/

public MDIInternalFrameDemo() { super("JCMDIPane Demo");

}

/\*\*

...

desktop = new JCMDIPane(); // a custom layered pane createFrame(); // Create first window

...

* Each frame can have its own menu bar, whose elements are
* defined by you. A "Window" menu is added automatically.

\*/

protected void createFrame() {

JCMDIFrame iframe = new JCMDIFrame(

"MDI Frame #" + (++MDIFrameCount), true, //resizable

true, //closable true, //maximizable true);//iconifiable

...

/\*\*

* Use this method to set the menu bar on the frame, even though
* it appears on the desktop rather than on the individual frame.

\*/ iframe.setMDIMenuBar(mbar);

...

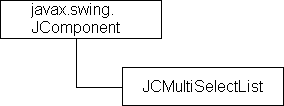
# 10

## Multi-Select List

[*Features of JCMultiSelectList*](#_bookmark294)[*Properties*](#_bookmark295)[*Constructors and Methods*](#_bookmark300)[*Examples*](#_bookmark314)

#### Features of JCMultiSelectList

JCMultiSelectList matches the API for JList except that two lists instead of one appear in the component’s GUI. There are four buttons between the two lists that move items back and forth. The left-hand list contains non-selected items and the right-hand list contains the selected items. In the context of a JCMultiSelectList, if an item is marked as selected, it means more than simply being highlighted. Besides providing a visual division of list items into the two columns, selected and non-selected, there are numerous methods for dealing with the values and indices of a set of selected values.



JCMultiSelectList provides a visual component that clearly distinguishes items chosen from a given list by removing them from the original list and placing them in another container. See the next figure for details.

You can create a JCMultiSelectList using one of its five constructors, four of which correspond to the constructors of a JList. The remaining constructor has an empty ListModel, but has a parameter for setting the horizontal gap between the two lists.

As with a JList, you can specify content using the ListModel interface, or you can supply content using Objects or Vectors.

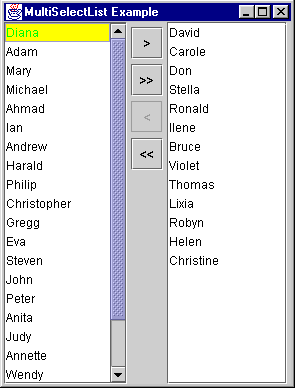
You are able to modify content in various ways depending on which objects you used to populate the main list.

End users may perform single or multiple, contiguous or non-contiguous selections of list items.

The ListSelectionModel generates a ListSelectionEvent to allow you to process user interactions.

##### 79

Four buttons control the movement of items in one list to the other. They are shown in the next figure. The top button moves selected items from the left-hand list to the right- hand list. The second from the top moves all items out of the left-hand list to the right- hand list. The bottom two buttons perform the analogous operation, but in the other direction.



*Figure 17 GUI for JCMultiSelectList.*

You set a ListModel on the component, or you can use the default model that is provided. In the latter case, you simply add Objects to the existing component. The component uses getSelected() to determine which items should appear in the right-hand list. Only the non-selected items show on the left.

#### Properties

A selection of JCMultiSelectList’s properties are shown in the following table. Please see [Properties of JCMultiSelectList](#_bookmark779) and [Bean Properties Reference](#_bookmark773) in [Appendix A](#_bookmark776) for a complete list.

|  |  |
| --- | --- |
| **Property** | **Description** |
| model | Gets or sets the model associated with the list. |
| prototypeCellValue | Sets the prototypical cell value, a cell used for the calculation of cell widths, rather than forcing the calculation to inspect every item in the list. |
| toolTipText | Gets or sets the text that appears in the tool tip. |

#### Constructors and Methods

###### Constructors

There are constructors for ListModel, array, and Vector types of data models.

###### Methods

JCMultiSelectList subclasses from JComponent, giving it a host of inherited methods. It overrides some, like addListSelectionListener, to provide specific functionality. The table shows a few frequently used methods. Please refer to the API for a full list.

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| addListSelectionListener() | Adds a listener to the ListSelectionListener’s list. Its parameters are a javax.swing.event and a ListSelectionListener. |
| addSelectionInterval() | Adds the specified interval to the current selection. It takes two int parameters that specify the beginning and ending positions of the interval. |
| clearSelection() | Clears the selection. |
| deselectAll() | Moves all items from the right list to the left list. |
| deselectItem() | Moves the items selected in the right list to the left list. |
| fireSelectionValueChanged() | Forwards the given notification event to all registered listeners. It takes two int parameters that specify the begin and end positions of the interval, and a third boolean parameter that specifies whether this is one of a rapidly occurring series of events. Parameters are int firstIndex,  int lastIndex, boolean isAdjusting  See [javax.swing.event.ListSelectionEvent](http://www.javasoft.com/products/jdk/1.2/docs/api/javax/swing/event/ListSelectionEvent.html). |
| getAnchorSelectionIndex() | Returns the first index argument from the most recent interval selection. |
| getSelectedIndex() | Returns the index of the first selected cell. |
| getSelectedIndices() | Returns the array of indices of selected items. |
| getSelectedValues() | Returns an array of the selected cell values. |
| setModel() | Sets the list’s data model. Its single parameter is a  [javax.swing.ListModel](http://www.javasoft.com/products/jdk/1.2/docs/api/javax/swing/ListModel.html). |

Chapter 10 Multi-Select List **81**

#### Examples

See examples.elements.MultiSelectList for a full listing of this example.

One of JCMultiSelectList’s constructors takes an array of list items as its parameter. Call this array data and define it as follows:

static String[] data = {"Tom", "Dick", "Harry"};

Create a JCMultiSelectList and give it the data:

JCMultiSelectList list = new JCMultiSelectList(data);

You process events generated by the list by implementing the valueChanged() method of the ListSelectionListener interface.

# 11

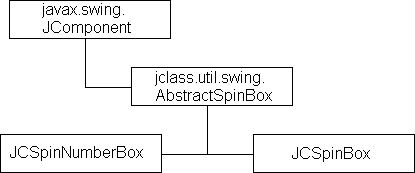
## Spin Boxes

[*Features of JCSpinBox and JCSpinNumberBox*](#_bookmark319)[*Classes and Interfaces*](#_bookmark320)[*Properties*](#_bookmark328)

[*Constructors and Methods*](#_bookmark347)[*Examples*](#_bookmark358)

#### Features of JCSpinBox and JCSpinNumberBox

Swing provides checkboxes and radio buttons, but no spin boxes. The JClass spin boxes fill the need for components that let the user select a number or an Object by clicking on up or down arrows. You can use a JCSpinBox to spin through a list of Objects (so long as the required editors and renderers exist, or have provided them), or use a JCSpinNumberBox, which can display any numeric object.



JCSpinBox looks very much like a JComboBox except that it has no dropdown. It takes a list of objects and presents these values in a spin box. You use the up and down arrows to cycle through the list.

Use JCSpinNumberBox for incrementing and decrementing objects of type java.lang.Number. You can select numbers of type Byte, Short, Integer, Long, or Float, and you can set maximum and minimum values for the spin operation.

Both components follow Swing’s MVC paradigm. A JCSpinBoxModel interface is used to manage the spin box’s data.

JCSpinBox has four constructors for the various ways in which you can supply data; that is, as Objects, Vectors, or via a JCSpinBoxModel. A fourth parameterless constructor is available. It uses an empty DefaultSpinBoxModel as a placeholder for data that will be provided later.

Contents of a spin box may be modified via a JCSpinBoxEditor interface.

##### 83

A KeySelectionManager interface defines a method for associating a keystroke to an item in the spin box.

JCSpinBoxModel methods are inherited from javax.swing.ListModel and javax.swing.ComboBoxModel. These are addListDataListener, getElementAt, getSize, removeListDataListener, getSelectedItem, and setSelectedItem.

The listener is the addValueListener, and the event is JCValueEvent.

#### Classes and Interfaces

###### Interfaces

|  |  |
| --- | --- |
| JCSpinBoxEditor | The editor component used for JCSpinBox components. |
| JCSpinBoxModel | A data model for JCSpinBox modeled after  javax.swing.ComboBoxModel.  JCSpinBoxModel is a ListDataModel with a selected item. This selected item is in the model since it is not always in the item list. It inherits its methods from [javax.swing.ComboBoxModel](http://www.javasoft.com/products/jdk/1.2/docs/api/javax/swing/ComboBoxModel.html) and [javax.swing.ListModel](http://www.javasoft.com/products/jdk/1.2/docs/api/javax/swing/ListModel.html). |
| JCSpinBox MutableModel | Extends JCSpinBoxModel to define models that are changeable. It declares methods for adding, inserting, and removing elements. |

**Helper** **Classes**

|  |  |
| --- | --- |
| AbstractSpinBox | The super class for JCSpinBox and JCSpinNumberBox. The class is abstract because it does not define spinUp(), spinDown(), and checkArrowButtons(), but it does provide the common functionality for JCSpinBox and JCSpinNumberBox. |
| JCValueEvent | The event object has methods getSource, getOldValue, and getNewValue, allowing you to find out which spin box posted the event, and its old and new values. |

#### Properties

**JCSpinBox properties**

These properties contain all of the functionality of JCSpinBox. In keeping with Swing’s MVC design paradigm, the JCSpinBoxModel interface contains a data model for JCSpinBox modeled after javax.swing.ComboBoxModel. JCSpinBoxModel is a

ListDataModel with a selected item. This selected item is in the model since it is not always in the item list.

|  |  |
| --- | --- |
| **Property Name** | **Description** |
| actionCommand | Sets or returns the action command that is included in the event sent to action listeners. |
| continuousScroll | Determines how selection is handled when the mouse button is held down on a spin arrow button. If continuousScroll is true, the component scrolls continuously through the items in the scroll box until the mouse button is released. If continuousScroll is false, a separate mouse click is required to select the next item in the scroll box. |
| getItemAt | Returns the list item at the specified index. |
| getItemCount | Returns the number of items in the list. |
| model | Sets or returns the data model currently used by the JCSpinBox. |
| renderer | Sets or returns the renderer used to display the selected item in the JCSpinBox field. |
| selectedIndex | Returns the index of the currently selected item in the list, or selects the item at the position marked by the index. |
| selectedItem | Returns the currently selected item, or sets the selected item in the JCSpinBox by specifying the object in the list. |
| isEditable | Returns true if the JCSpinBox is editable. |

###### JCSpinNumberBox properties

These properties let you specify the operation, that is, whether the numbers in the spin box are whole numbers or floating point numbers. Additionally, you can set the spin increment and bounds.

For a complete list of properties, please see [Properties of JCSpinBox](#_bookmark807) and [Properties of](#_bookmark809) [JCSpinNumberBox](#_bookmark809) in [Appendix A](#_bookmark776).

|  |  |
| --- | --- |
| **Property Name** | **Description** |
| continuousScroll | Determines how selection is handled when the mouse button is held down on a spin arrow button. If continuousScroll is true, the component scrolls continuously through the items in the scroll box until the mouse button is released. If continuousScroll is false, a separate mouse click is required to select the next item in the scroll box. |

|  |  |
| --- | --- |
| **Property Name** | **Description** |
| maximumValue | Returns or sets the maximum value. The default is  Long.MAX\_VALUE |
| minimumValue | Returns or sets the minimum value. The default is  Long.MIN\_VALUE. |
| numberFormat | The NumberFormat object used by the spinner to parse and format numbers |
| operation | Takes a JCSpinNumberBox.INTEGER and sets the operation. |
| spinStep | The spin increment. The default is 1. |
| value | The current value of the spinner. |
| valueRange | Convenience method to set maximum and minimum values together. Defaults are Long.MIN\_VALUE,  Long.MAX\_VALUE. |

#### Constructors and Methods

###### Constructors

|  |  |
| --- | --- |
| JCSpinNumberBox() | Use this component when you want to let your users increment or decrement a object of type java.lang.Number.  Long.MIN\_VALUE, Long.MAX\_VALUE, and floating point numbers outside the range Double.MIN\_VALUE cause an exception. Use setOperation(JCSpinNumberBox.FLOATING\_POINT) when you want to use floating point numbers. The use of setOperation(JCSpinNumberBox.INTEGER) is optional, since this is the default case. |
| JCSpinBox() | Use this component when you want a spin box containing an Object. For some non-standard objects, you may need to create your own editor and renderer. |

**JCSpinBox methods**

These methods manage a list of items by providing methods for adding and removing items from the list of objects, and for adding listeners for these changes. See the API for the complete list of JCSpinBox methods.

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| addActionListener() | Adds an ActionListener. |
| removeActionListener() | Removes an ActionListener. |
| removeAllItems() | Removes all items from the item list. |
| addItem() | Adds an item to the item list. |
| removeItem() | Removes an item from the item list. |
| removeItemAt() | Removes the item at anIndex. To use this method, the JCSpinBox data model must implement JCSpinBoxMutableModel. |
| addItemListener() | Adds a java.awt.event.ItemListener. Its parameter is the class that will receive the event when the selected item changes. |
| removeItemListener() | Removes a java.awt.event.ItemListener. |

#### Examples

To use a JCSpinNumberBox, simply instantiate it and set its parameters according to your needs, for example:

JCSpinNumberBox float\_spin = new JCSpinNumberBox(); float\_spin.setName("FloatingPointSpinBox"); float\_spin.setValue(new Integer(0));

float\_spin.setValueRange(new JCSpinNumberBox.Range(new Integer(0),

new Integer(12)));

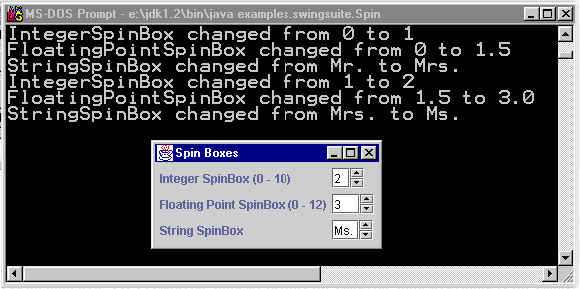
float\_spin.setSpinStep(new Double(1.5)); float\_spin.setOperation(float\_spin.FLOATING\_POINT);

You don’t need to use the setOperation method when you create an INTEGER version of a JCSpinNumberBox since that is the default type.

Similarly, you can create a JCSpinBox:

JCSpinBox string\_spin = new JCSpinBox(titles); string\_spin.setName("StringSpinBox"); string\_spin.setSelectedIndex(0); string\_spin.addValueListener(listener);

The figure shows that each time a mouse click changes a spin box’s value, the generated event can report on both the old and the new value. The output in [Figure 18](#_bookmark359) results from clicking each spin box in succession twice.



*Figure 18 Capturing spin box events.*

###### Listening for Spin Box Events

JCNumberSpinBox uses the JCValueModel interface to set and get its values, and to define its listeners. To respond to spin events, do something like this:

* + 1. Create a listener as part of the setup for the component, for example:

JCValueListener listener = new ValueListener();

* + 1. Then add a listener to the spin box:

float\_spin.addValueListener(listener);

* + 1. Implement the listener class and define a valueChanged method:

class ValueListener implements JCValueListener { public void valueChanging(JCValueEvent e) {

}

public void valueChanged(JCValueEvent e) { System.out.println(((Component) e.getSource()).getName() +

" changed from " +

e. getOldValue() + " to " + e.getNewValue());

}

} // end of ValueListener

# 12

## Splash Screen

[*Features of JCSplashScreen*](#_bookmark364)[*Classes and Interfaces*](#_bookmark366)[*Methods and Constructors*](#_bookmark368)[*Examples*](#_bookmark372)

#### Features of JCSplashScreen

A splash screen is an image that appears while an application is loading. It serves both as an indication that the program is being loaded from disk and as a place to put notices, such as copyrights, version or release numbers, and the like.

JCSplashScreen does the following:

Creates a splash screen given an Icon or the location of the image. The image location is the package path of the image and must be in the classpath. Any Icon, such as a GIF, JPEG, or other supported image may be used, so long as the time it takes to load is acceptable.

Once instantiated, a JCSplashScreen appears only once. Hiding it causes it to be disposed.

#### Classes and Interfaces

The stand-alone class com.klg.jclass.swing.JCSplashScreen subclasses from java.lang.Object, providing an independent mechanism for displaying an image in a window in the middle of the screen.

No interfaces are used in JCSplashScreen.

##### 89

#### Methods and Constructors

###### Constructors

|  |  |
| --- | --- |
| JCSplashScreen() | JCSplashScreen has two constructors for instantiating a splash screen, one taking a String that specifies the location of the image, and the other taking an Icon. It throws an illegalArgumentException if the image String or image icon is invalid. |

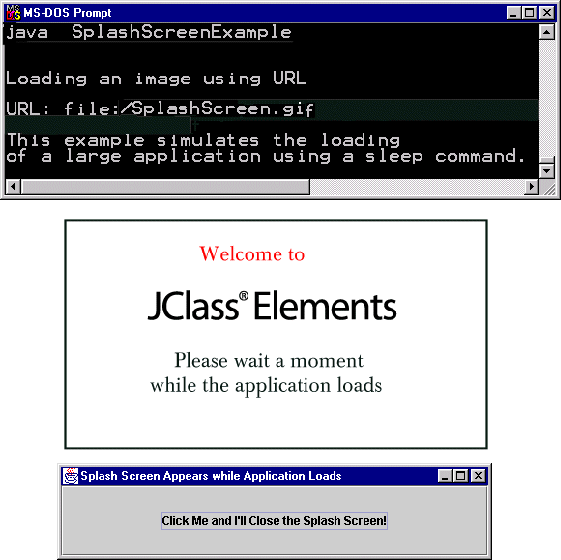
**Methods**

|  |  |
| --- | --- |
| setVisible() | A Boolean method that shows or hides the splash screen. Once the splash screen is hidden, the splash screen window will be disposed. This means the splash screen cannot become visible again. |

#### Examples

If you compile and run the code for this example, which is given below, you’ll see a message printed on the console informing you that the application has started. The image for the splash screen is loaded and is made visible, then the program enters a wait state until a sleep command times out. An application that takes a long time to load would exhibit similar behavior. The user knows that loading is in progress because the splash screen is visible. It contains whatever graphic information you think is appropriate.

The example uses a JCExitFrame to hold a button that controls the disposal of the splash screen. All that is required for its disposal is the command setVisible(false), but once it is given, the splash screen is gone for good. You would issue this command after receiving notification that the application is ready to run.



*Figure 19 The visible elements in SplashScreenExample.*

import com.klg.jclass.swing.\*; import com.klg.jclass.util.swing.\*; import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionListener; import java.awt.event.ActionEvent; import java.net.\*;

public class SplashScreenExample extends JFrame implements

ActionListener {

static JCSplashScreen ss; static Icon image;

SplashScreenExample() { URL url =

getClass().getResource("/images/SplashScreen.gif");

// convert URL to Image icon image = new ImageIcon(url); System.out.println(

"\n\nLoading an image using URL\n\nURL: " + url); System.out.println("This example simulates the loading");

Chapter 12 Splash Screen **91**

System.out.println(

"of a large application using a sleep command.");

// initialize(image);

ss = new JCSplashScreen(image);

}

public void actionPerformed(ActionEvent e) { if (e.getSource() instanceof JButton) {

ss.setVisible(false);

}

}

public static void main(String[] args){ SplashScreenExample sse = new SplashScreenExample();

String title = "Splash Screen Appears while Application Loads"; JCExitFrame frame;

frame = new JCExitFrame(title); ss = new JCSplashScreen(image); ss.setVisible(true);

try {

Thread.currentThread().sleep(10000);

} catch (Exception e) {}

Container cp = frame.getContentPane(); JButton btn = new JButton(

"Click Me and I'll Close the Splash Screen!"); btn.addActionListener(sse);

cp.add(btn); frame.setSize(450, 100); frame.setVisible(true);

frame.setExitOnClose(true); // Close the window so the

application can exit.

}

}

# 13

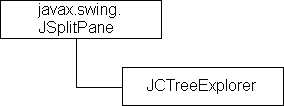
## Tree/Table Components

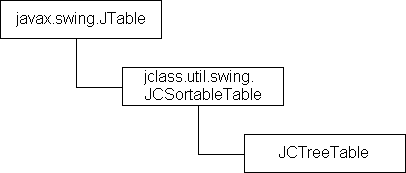
[*Features of JCTreeExplorer and JCTreeTable*](#_bookmark377)[*Classes and Interfaces*](#_bookmark378)[*Properties*](#_bookmark391)[*Examples*](#_bookmark461)

#### Features of JCTreeExplorer and JCTreeTable

Swing’s JTree and JTable are the two components that do more than merely display data; they attempt to manage the data as well. This becomes important when you need to organize large amounts of data and provide a view that displays a portion of it along with an indication of its relationship to the rest. Information that has a hierarchical structure, like a file system, can be displayed as tree data, while other types of data nicely fit a tabular format. There are a large number of data structures that combine tree-like and table-like properties. A file system has a hierarchical organization that begs to be represented as a tree, yet the individual directories and files have properties, such as name, size, type, and date modified, that fit nicely in a row-column organization.

Obviously there is a need for a component that lets you combine the look and functionality of both a tree and a table.





JCTreeExplorer and JCTreeTable fill the need for components that have the dual characteristics of a tree and a table, and provides these functions:

Allows you to view the object as a tree, with an accompanying table. Any tree node may have tabular data associated with it.

##### 93

Contains a flexible “painter” object that accommodates a Swing cell renderer or a JClass cell renderer.

Permits the construction of arbitrary data sources as treetables through its JCTreeTableModel interface. Any class can be used to supply data to the treetable, provided that it implements the JCTreeTableModel interface.

The two components have advanced column sorting functionality. Each column can have a different ordered set of columns that are to be used as secondary sort keys. When a user clicks on a column header to sort that column, any identical cells are arranged based on the sort order of the secondary key, or keys.

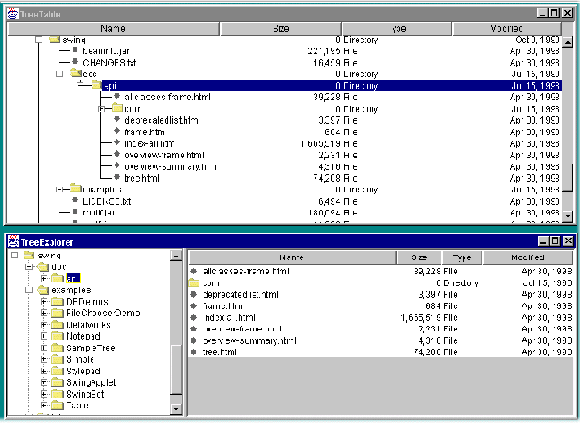
Cells may be edited by implementing JCCellEditor.

Folder icons can be customized by replacing the editor/renderer, or by setting a

JCIconRenderer.

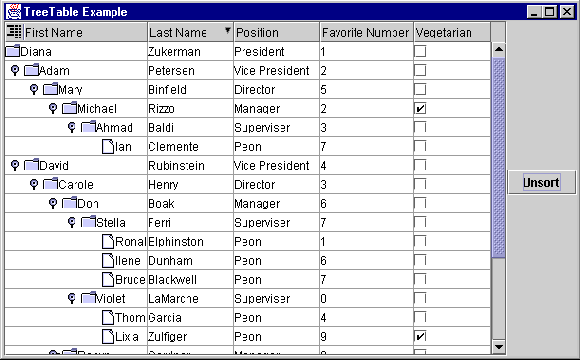
Since JCTreeExplorer and JCTreeTable are enhancements of Swing’s JTable and JTree, it’s a good idea to be familiar with those components to ease the learning curve. Need a primer on Swing’s table and tree components? See the tutorial on [How to Use Tables](http://www.javasoft.com/docs/books/tutorial/uiswing/components/table.html) and [How to Use Trees](http://www.javasoft.com/docs/books/tutorial/uiswing/components/tree.html) at Sun’s *javasoft* Web site.

JCTreeExplorer presents a table for the currently selected node, while JCTreeTable shows table rows for all visible nodes. See the following figure for the different visual characteristics between the two components.

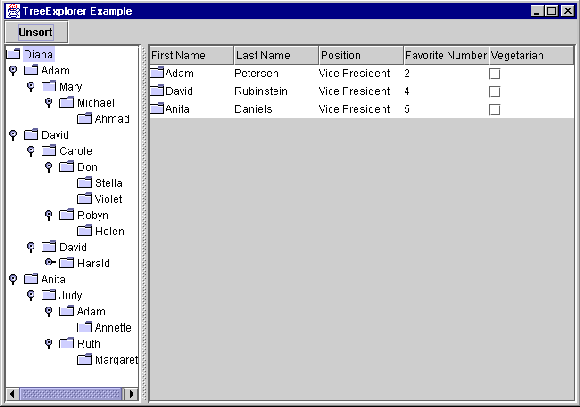


*Figure 20 Views of a JCTreeTable and a JCTreeExplorer.*

JCTreeTable and JCTreeExplorer support Swing’s pluggable look-and-feel. The previous figure shows the Windows look and feel, while the following two figures show the default Metal look and feel.



*Figure 21 JCTreeTable component, metal look and feel.*



*Figure 22 JCTreeExplorer component, metal look and feel.*

#### Classes and Interfaces

###### Interfaces for JCTreeExplorer and JCTreeTable

|  |  |
| --- | --- |
| **Interface** | **Description** |
| JCTreeIconRenderer | In com.klg.jclass.util.treetable, this interface represents a class that renders a tree icon. Not a renderer in the strict definition of the word, it provides the icon to be rendered. It is not necessary to make this interface do the rendering since Icons know how to render themselves.  A default implementation of this class simply returns the  *plaf* icon it is passed.  The purpose of this mechanism is to allow a user to override the icons being drawn in a tree without   1. having to figure out the default *plaf* for the icons that they do not wish to override, and 2. overriding simple data-type editors.   Its single method is getNodeIcon. |
| JCTreeTableModel | Model to use that combines the TreeModel and TableModel interfaces. This model allows data to be viewed as a multicolumn tree in a left-hand pane, and a table in a right-hand pane.  Note that specific implementations need to implement both JCTreeTableModel and TableModel for them to work properly. |

Here is the definition of the JCTreeTableModel interface:

public interface JCTreeTableModel {

// Returns the value of the specific node and column public Object getValueAt(Object node, int column);

// Returns whether a particular cell is editable,

// given the node and column

public boolean isCellEditable(Object node, int column);

// Sets the value at a particular node and column

public void setValueAt(Object value, Object node, int column);

// The following methods map exactly onto

// javax.swing.table.TableModel

public void addTableModelListener(TableModelListener l); public Class getColumnClass(int column);

public int getColumnCount();

public String getColumnName(int column); public int getRowCount();

public Object getValueAt(int row, int column); public boolean isCellEditable(int row, int column);

public void removeTableModelListener(TableModelListener l); public void setValueAt(Object value, int row, int column);

// The following methods map exactly onto javax.swing.tree.TreeModel public void addTreeModelListener(TreeModelListener l);

public Object getChild(Object parent, int index); public int getChildCount(Object parent);

public int getIndexOfChild(Object parent, Object child); public Object getRoot();

public boolean isLeaf(Object node);

public void removeTreeModelListener(TreeModelListener l); public void valueForPathChanged(TreePath path, Object newValue);

}

###### Classes

|  |  |
| --- | --- |
| **Class** | **Description** |
| DefaultTreeTable SelectionModel | Extends javax.swing.tree.DefaultTreeSelectionModel and implements JCTreeTableSelectionModel. Like JCMultiSelectList, a treetable offers different selection modes. A treetable has an associated DefaultTreeTableSelectionModel when it is created, but you can define your own selection model, so long as it is a subclass of DefaultTreeSelectionModel and implements JCTreeTableSelectionModel. |
| TreeTableSupport | Abstract class in com.klg.jclass.util.treetable that provides an implementation that handles a TreeModel/TableModel combination for use in a Table component.  Its functionality includes tracking expanded node counts, mapping and posting expansions, and selection events. It also provides a node “painter” object that can be wrapped into a Swing CellRenderer or a JClass CellRenderer. |
| JCSortableTable | A subclass of JTable that internally wraps any TableModel it is given with a JCRowSortTableModel and provides a Comparator that has an adjustable list of the column indexes that it uses for sorting. Clicking on a column header invokes the sorting behavior tied to that column; clicking again reverses the sort. |

|  |  |
| --- | --- |
| **Class** | **Description** |
| TreeWithSortable Children | This class implements JCTreeTableModel and  JCRowSortModel.  It constructs a JCTreeTableModel that wraps a given instance of a JCTreeTableModel and provides a sorted mapping of the children for any given leaf node. The sort order is defined by the configurable Comparator property. |
| com.klg.jclass. util.treetable BranchTree | An implementation of TreeModel that wraps a tree model so that it only exposes branches; that is, non-leaf nodes. This is useful for explorer-type views where the “tree view” portion only displays the branch nodes. |
| DefaultTreeIcon Renderer | An implementation of JCTreeIconRenderer, its getNodeIcon method returns the icon to render at the right of the specified value. This simple implementation returns the plaf icon passed to it. |
| JCTreeExplorer | A subclass of JSplitPane that provides a tree view on the left-hand side of the split pane, and a table view on the right.  Constructor: JCTreeExplorer(JCTreeTableModel) |
| JCTreeTable | A subclass of JTable that handles listeners, rendering, editing, and painting of a component that combines tree-like and table-like properties.  Constructor: JCTreeTable(JCTreeTableModel) |
| com.klg.jclass. util.treetable NodeChildrenTable | Maps the children of a particular node in a  JCTreeTableModel into a standard Swing TableModel. |

**Providing your own sorting** **mechanism**

If you need to provide your own sorting algorithm, one way is to subclass

JCRowComparator and pass a comparator of the new type to TreeWithSortableChildren.

#### Properties

For a complete list of properties, please see [Properties of JCTreeExplorer](#_bookmark785) and [Properties](#_bookmark787) [of JCTreeTable](#_bookmark787) in [Appendix A](#_bookmark776).

###### Properties of JCTreeExplorer

|  |  |
| --- | --- |
| getTree() | Returns the JTree component used. |
| getTable() | Returns the JTable component used. |
| setKeyColumns() | Sets which columns are to be used as primary and secondary sort keys. It takes a column number (0, 1, 2 ...) as its first parameter, and an array of column numbers as its second parameter.  Example: setKeyColumns(0, {1, 0}) specifies that when the user clicks on the header in the first column, sorting takes place based on the second column, and identical entries in the second column are sorted based on the ordering implied by the first column. This is a useful sort key for directories, where the first column is the file or directory name, and the second column contains the object’s size. Because directories have size zero, they are sorted at the top and then arranged alphabetically. |

**Properties of** **JCTreeTable**

|  |  |
| --- | --- |
| treeTableModel | The treeTableModel is the interface for the data. |
| treeIconRenderer | Gets or sets the icon renderer. If this property is set to null, no icon will be shown. |
| rootVisible | You can show the root node or not, depending on the setting of this Boolean property. The accessor method is called isRootVisible. |
| showsRootHandles | Determines whether the node handles are to be displayed. |

#### Methods

**JCTreeExplorer Methods**

|  |  |
| --- | --- |
| getSelectionPath() | Returns the javax.swing.tree.TreePath of the first selected row inside the table view. |
| getSelectionPaths() | Returns the javax.swing.tree.TreePath of the first selected row inside the table view. |
| getTable() | Returns the JTable component used. |
| getTree() | Returns the JTree component used |

|  |  |
| --- | --- |
| getTreeIconRenderer() | Returns the icon renderer being used. |
| setTreeIconRenderer() | Sets the icon renderer. |
| setUI() | Sets the javax.swing.plaf.TableUI UI. |

**JCTreeTable Methods**

|  |  |
| --- | --- |
| addSelectionPath() | Adds the node identified by the specified TreePath to the current selection. |
| addSelectionPaths() | Adds each path in the array of paths to the current selection. |
| addTableHeader MouseListener() | Adds a MouseListener to the table header. |
| addTreeExpansion Listener() | Adds a listener for TreeExpansion events. |
| addTreeWillExpand Listener() | Adds a listener for TreeWillExpand events. |
| collapsePath() | Ensures that the node identified by the specified path is collapsed and viewable. |
| collapseRow() | Ensures that the node in the specified row is collapsed. |
| createSortable TableColumn() | Creates a TableColumn. |
| expandPath() | Ensures that the node identified by the specified path is expanded and viewable. |
| expandRow() | Ensures that the node in the specified row is expanded and viewable. |
| getCellEditor() | Overridden to return the appropriate data render for the first column if the treetable is in tree display mode. |
| getCellRenderer() | Overridden to return the appropriate data render for the first column if the treetable is in tree display mode. |
| getClosestPath  ForLocation() | Returns the row to the node that is closest to X,Y. |
| getEditingPath() | Returns the path to the element that is currently being edited. |
| getExpandedDescendants() | Returns an Enumeration of the descendants of path that are currently expanded. |

|  |  |
| --- | --- |
| getPathForLocation() | Returns the path for the node at the specified location. |
| getPathForRow() | Returns the path that is displayed at the table row specified. |
| getRowForLocation() | Returns the row for the specified location. |
| getRowForPath() | Returns the row that displays the node identified by the specified path. |
| getRowsForPaths() | Returns the rows for the visible specified paths. |
| getScrollsOnExpand() | Returns true if the tree scrolls to show previously hidden children. |
| getSelectedPath() | Returns the TreePath of the first selected row. |
| getSelectionPath() | Returns the path to the first selected node. |
| getSelectionPaths() | Returns the paths of all selected values. |
| getShowNodeLines() | Returns the state of ShowNodeLines. If true, the connecting lines that are drawn between nodes in an explorer view are shown. |
| getShowsRootHandles() | Returns true if handles for the root nodes are displayed. |
| getTreeIconRenderer() | Returns the icon renderer being used. |
| getTreeSelectionModel() | Returns the model for selections. |
| getTreeTableModel() | Returns the JCTreeTableModel that is providing the data. |
| getView() | Returns the current view. |
| isPathSelected() | Returns true if the item identified by the path is currently selected. |
| isRootVisible() | Returns true if the root node of the tree is displayed. |
| isSortable() | Is the treetable sortable? Returns the value of the  sortable property. |
| makeVisible() | Ensures that the node identified by a path is currently viewable. |
| removeTreeExpansion Listener() | Removes a listener for TreeExpansion events. |

|  |  |
| --- | --- |
| removeTreeWillExpand Listener() | Removes a listener for TreeWillExpand events. |
| setRootVisible() | Determines whether or not the root node from the  TreeModel is visible. |
| setScrollsOnExpand() | Determines the behavior of a node when it is expanded. If true, the viewport will scroll to show as many descendants as possible when the node is expanded; if false, the viewport will not scroll. |
| setSelectionPath() | Selects the node identified by the specified path. |
| setSelectionPaths() | Selects the specified paths. |
| setShowNodeLines() | Allows you override the *plaf*-specified behavior for drawing lines. |
| setShowsRootHandles() | Determines whether the node handles are to be displayed. |
| setSortable() | Sets whether the treetable is sortable. |
| setSwitchPolicy() | Sets the switchPolicy variable that determines whether or not to allow view switching Options are: JCTreeTable.SWITCH\_BUTTON\_DONT\_SHOW, JCTreeTable.SWITCH\_VIEW\_NEVER, JCTreeTable.SWITCH\_VIEW\_TO\_TABLE\_ON\_SORT, JCTreeTable.SWITCH\_VIEW\_ON\_ICON\_ONLY. |
| setTreeIconRenderer() | Sets the icon renderer. |
| setTreeTableModel() | Sets the TreeModel that will provide the data. |
| setTreeTable SelectionModel() | Sets the tree's selection model. |
| setUI() | Sets the javax.swing.plaf.TableUI UI. |
| setView() | Sets whether the view is for a tree or a table. |
| updateUI() | Updates the UI. |

#### Examples

**Implementing a custom node icon for a JCTreeTable**

Your application may require that you supply your own custom node icon for the tree view. Create your own implementation of JCTreeIconRenderer, and write a method similar to the one whose signature is shown here:

public Icon getNodeIcon(TreeModel treemodel,

Object node, Object value,

Class object\_class, boolean is\_leaf, boolean is\_expanded, Icon plaf\_icon)

You may want to have two icons, one for nodes with children and one for leaf elements. In that case, use the boolean parameter is\_leaf to choose which icon will be used.

The method should return the Icon you want to use. Pass your implementation of JCTreeIconRenderer to your instance of a JCTreeTable using setTreeIconRenderer(JCTreeIconRenderer renderer).

Please see *TreeExplorer.java* and *TreeTable.java* in the *JCLASS\_HOME\examples\elements\*

directory for some examples of using both JCTreeExplorer and JCTreeTable.

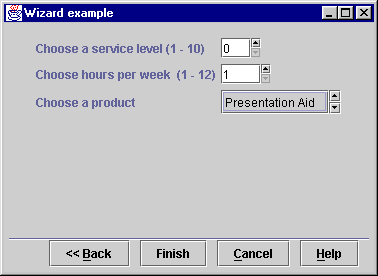
# 14

## Wizard Creator

[*Features of JCWizard and JCSplitWizard*](#_bookmark465)[*Classes*](#_bookmark466)[*Constructors and Methods*](#_bookmark472)[*Events*](#_bookmark489)[*Examples*](#_bookmark500)

#### 14.1 Features of JCWizard and JCSplitWizard

JCWizard lets you create and manage a Wizard-style group of dialogs by supplying informative events and special page components with standard buttons. You add a JCWizardListener to your JCWizardPages to invoke the actions that each page needs to perform.



*Figure 23 A sample Wizard page.*

JCWizard supplies these features:

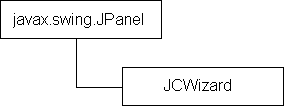
Standard Next, Back, Finish, Cancel and Help buttons that are characteristic of Wizard dialogs.

You provide instructions for the end-user on each page, and define the actions corresponding to the choices made by the end-user.

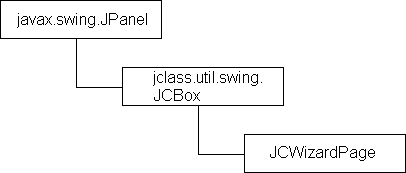
The Wizard’s pages are instances of JCWizardPage. As is usual with Swing components, you do not add children to JCWizardPage. Instead, you call its getContentPane() method and add items to it.

##### 105

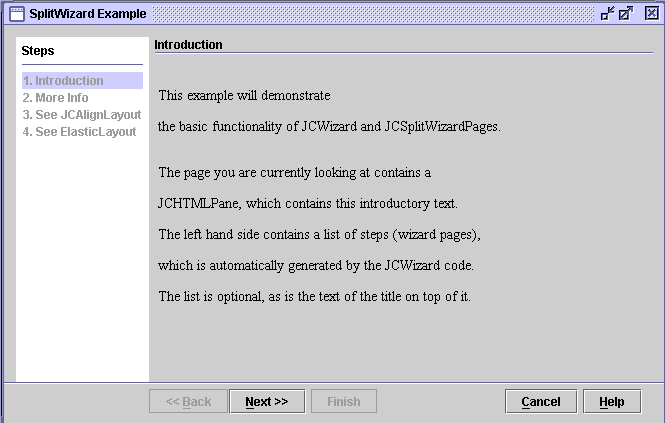
The JCWizard component is a container that manages JCWizardPages. The pages are added to it in a way that only one of them shows at a time, but navigation buttons let the end user move back and forth through the deck. The component posts a JCWizardEvent as changes occur.



The JCWizardPage provides a getContentPane() method to return the panel to which you add content. It automatically builds content to manage the Next, Previous, Finish, Cancel, Help buttons at the bottom right of the page.



JCSplitWizard, on the other hand, creates a split-Wizard layout, which allows for one page to be created with multiple panels, rather than multiple pages with one panel.



*Figure 24 A sample split-Wizard page.*

The button features provided for a standard Wizard are also available for the split-Wizard, though there is only one set that will apply to the entire Wizard. (The Back button is automatically unavailable on the first page, and the Next button is automatically unavailable on the last page.) Pages that are added are displayed in the right pane, while the left pane is used for the progress list, if one has been created.

#### Classes

The classes in the JCWizard group are:

|  |  |
| --- | --- |
| JCWizard | Creates and manages a Wizard-style set of dialogs. To create your own Wizard, design JCWizardPages and add Wizard listeners. |
| JCWizardPage | A JCBox that knows about Wizard-style actions. |
| JCWizardEvent | The event object that carries information about changes to  JCWizard pages. |
| JCWizardListener | The listener for JCWizard events. |

The classes in the JCSplitWizard group are:

|  |  |
| --- | --- |
| JCSplitWizard | Creates a wizard with two panes, and a bottom button panel. The panes can contain any component, or left pane can optionally contain an automatically generated list of Steps. |

#### Constructors and Methods

###### JCWizard Constructors

JCWizardPage’s constructor lets you specify the buttons you want on a page by combining

JCWizardPage constants, as follows:

page = new JCWizardPage(JCWizardPage.NEXT |

JCWizardPage.PREVIOUS | JCWizardPage.FINISH | JCWizardPage.CANCEL | JCWizardPage.HELP);

###### JCSplitWizard Constructors

JCSplitWizard has two constructors. There is a constructor that can be used to create a standard two-pane Wizard, along with buttons and an empty string as a title.

The second constructor is equipped with two arguments:

The int argument that specifies which buttons should be included; The String title that specifies the title of the progress list.

###### JCWizard Methods

JCWizard inherits both properties and methods from JPanel. Listed here are the methods that JCWizard itself defines to provide the needed functionality for managing Wizard pages.

|  |  |
| --- | --- |
| **Method** | **Description** |
| add() | Adds a page to the Wizard. |
| addWizardListener() | Adds a new JCWizardListener to the list. |
| cancel() | Invokes the registered “cancel” action. |
| finish() | Invokes the registered “finish” action. |
| first() | Moves to the first page in the Wizard. |
| help() | Invokes the registered “help” action. |
| last() | Moves to the last page in the Wizard. |
| next() | Advances to the next page in the Wizard. |
| previous() | Moves to the previous page in the Wizard. |
| show() | The method takes a parameter (String name), and moves to the Wizard page with the specified name. |

###### JCSplitWizard Methods

Listed here are the methods that JCSplitWizard itself defines to provide the needed functionality for managing split-Wizard pages.

|  |  |
| --- | --- |
| **Method** | **Description** |
| addPage() | Adds a page to the right panel of the Wizard. |
| getLeftPanel() | Accesses the left panel, which can contain an automatically generated list of steps based on titles, an image, help text, or any other desired components. |
| cancel() | Invokes the registered “cancel” action. |
| finish() | Invokes the registered “finish” action. |
| first() | Moves to the first page in the Wizard. |

|  |  |
| --- | --- |
| help() | Invokes the registered “help” action. |
| last() | Moves to the last page in the Wizard. |
| next() | Advances to the next page in the Wizard. |
| previous() | Moves to the previous page in the Wizard. |

#### Events

A JCWizard or JCSplitWizard listens for JCWizardEvents. A JCWizardEvent contains information on the Object that triggered the event, the Component’s current page and new page, two Booleans, whether the event occurred on the last page, and whether the event should be allowed to finish processing.

Interface JCWizardListener methods are:

|  |  |
| --- | --- |
| nextBegin | Invoked *before* advancing to the next page. Calling e.setAllowChange(false) will prevent the advance to the next page. Check e.isLastPage() to see if you are on the last page. |
| nextComplete | Invoked after advancing to the next page. |
| previousBegin | Invoked *before* returning to the previous page. Calling e.setAllowChange(false) will prevent the return to the previous page. |
| previousComplete | Invoked after advancing to the previous page. |
| finished | Invoked if a “finish” action is triggered. |
| canceled | Invoked if the “cancel” action is triggered. |
| help | Invoked if the “help” action is triggered |

#### Examples

Please refer to examples.elements.Wizard.java to see how to construct regular Wizard pages. Briefly, these are the steps:

1. Add an instance of a JCWizard to a JPanel or similar component.
2. Create a JCWizard page.
3. Specify the buttons that should appear on each page.
4. Name the page.
5. The content for each page will likely be a JPanel. Add it to the Wizard page’s content pane.
6. Add the page to the JCWizard.
7. Continue adding pages as necessary.

Please refer to examples.elements.SplitWizard.java to see how to construct split- Wizard pages. Briefly, these are the steps:

1. Create an instance of a JCSplitWizard, adding the desired buttons and determining whether or not a progress list should be generated.

To create a progress list, pass the title of the list as the second argument in the constructor.

If no progress list is desired, simply leave the title null, or use the no-argument constructor.

1. Create content for the right-hand wizard pages. These can be any instance of

JComponent.

1. Add the pages to the wizard using wizard.addPage(JComponent, page title), where page title is the title of the page.
2. Add the wizard to a container; add listeners, if desired.
3. If a progress list has not been specified, call getLeftPage() to add content to the left pane.

# 15

## Layout Managers

[*Features of the Layout Managers in JClass Elements*](#_bookmark504)[*Interfaces*](#_bookmark516)[*Properties*](#_bookmark518)

[*Constructors and Methods*](#_bookmark527)[*Examples*](#_bookmark540)

#### 15.1 Features of the Layout Managers in JClass Elements

This chapter describes JClass Elements’ layout managers and the components that are closely associated with them. The layout managers are JCAlignLayout, JCColumnLayout, JCElasticLayout, JCGridLayout, and JCRowLayout. JCBorder, JCBox, JCBrace, and JCSpring are the associated components.

##### 15.1.1 Layout Manager Classes

###### JCAlignLayout

JCAlignLayout is a layout manager that provides a simple way to lay out a vertically arranged group of control components, each with an associated label (or other component) placed to its left.

###### JCColumnLayout

JCColumnLayout is a simple subclass of JCElasticLayout that allows layout in a single column.

###### JCElasticLayout

JCElasticLayout is a layout manager that supports JCElastic components either horizontally or vertically. A component is considered *elastic* if it either implements the JCElastic interface or it has a constraint object that implements the JCElastic interface. Layout is performed in either a single row or column (depending on its orientation when created). The preferred size is calculated in the direction of orientation. If the container is bigger than the preferred size of all the components then the extra space is divided up between the components that are “elastic” in the direction of the orientation. The extra space is allocated to each of the components with respect to their “elasticity”. If all the elastic components have the same elasticity (in the direction of the orientation) then they are equally stretched. If there is an uneven number of pixels to apportion, then the first *n* units of elasticity are allocated the extra pixels, where *n* is the remainder when the total elasticity is divided by the number representing the extra pixels (*n = total\_elasticity mod extra\_pixels)*.

##### 111

###### JCGridLayout

JCGridLayout is an improved subclass of GridLayout. It lays out a grid of rows and columns based on the attributes of the individual rows and columns. Whereas GridLayout uses the widest and tallest child to size each cell, JCGridLayout uses the widest element in a column to set the width of that column, and the tallest element in a row to set the height of that row.

###### JCRowLayout

JCRowLayout is a simple subclass of JCElasticLayout that allows layout in a single row.

##### 15.1.2 Associated Component Classes

###### JCBorder

JCBorder can be used with any layout manager. With it you can place a border anywhere, not just around a component. You draw a border by overriding the component’s paint method and calling JCBorder.draw(). Its parameters allow you to specify the Graphics object it will be passed, along with its border style, border size in pixels, placement of the top left corner relative to its parent, its width and height, and the shadow colors for its sides. Please refer to the API for a full description of the two variations of the parameter list for this method.

Border styles may be any one of the following:

|  |  |
| --- | --- |
| JCBorder.ETCHED\_IN | Double line, border appears inset. |
| JCBorder.ETCHED\_OUT | Double line, border appears raised. |
| JCBorder.FRAME\_IN | 1-pixel shadow-in at edge, border appears framed. |
| JCBorder.FRAME\_OUT | 1-pixel shadow-out at edge, border appears framed. |
| JCBorder.IN | Border appears inset. |
| JCBorder.OUT | Border appears raised. |
| JCBorder.CONTROL\_IN | MS-Windows control shadows. |
| JCBorder.CONTROL\_OUT | MS-Windows control shadows. |
| JCBorder.PLAIN | Border drawn in foreground color. |
| JCBorder.NONE | No border drawn. |

###### JCBox

JCBox is a Swing container that uses the JCElasticLayout to lay out components in a single row or column. Use the orientation property within an IDE to control the orientation of the box. The JCSpring and JCBrace components are useful Beans to use in conjunction with this container.

###### JCBrace

An implementation of a component that participates in a layout even though it has no view. It is called a *brace* because its main function is to reserve space as a way of controlling the layout of the visible components. A brace usually has equal minimum and preferred sizes, and an unlimited maximum size.

###### JCSpring

This is a stretchable concrete implementation of the JCElasticLayout interface, which specifies components as stretchable for the JCElasticLayout manager and its subclasses. A JCSpring has independently settable elasticity parameters for both the horizontal and vertical directions.

#### Interfaces

JCElasticLayout — The interface that informs enabled layout managers that a particular component should be stretched to its maximum before stretching any non-elastic components.

#### Properties

###### JCBox

|  |  |
| --- | --- |
| alignment | One of SwingConstants.LEFT, SwingConstants.CENTER, or  SwingConstants.RIGHT, specifying the alignment of the layout. |
| orientation | One of JCElasticLayout.HORIZONTAL or JCElasticLayout.VERTICAL, specifying how the container is to lay out its components. |

**JCBrace**

|  |  |
| --- | --- |
| orientation | JCElasticLayout.HORIZONTAL or JCElasticLayout.VERTICAL, specifying whether it is a horizontal or a vertical brace. |
| length | This is the value of both the minimum size and the preferred size. Whether the length refers to a horizontal or a vertical dimension depends on the orientation. |

**JCSpring**

|  |  |
| --- | --- |
| horizontalElasticity verticalElasticity | These are properties with integer values specifying the relative elasticities of the components to which they refer. |

#### Constructors and Methods

* + 1. **Layout Managers**

**JCAlignLayout**

|  |  |
| --- | --- |
| getLabelVertical Alignment() | Returns the vertical position of a label relative to its control. |
| setResizeHeight() | Sets whether the control should be resized vertically to the height of the largest component in its row (default: false). This value is ignored for labels (the components in odd columns). |
| setResizeWidth() | Sets whether the control should be resized horizontally to its parent's right edge if it is in the last column (default: false). |
| setLabelVertical Alignment() | Sets the vertical position of a label relative to its control. Choices are TOP, MIDDLE (default), or BOTTOM. |

**JCColumnLayout**

A simple subclass of JCElasticLayout that arranges layout in a single column.

|  |  |
| --- | --- |
| JCColumnLayout() | Creates a column layout that aligns components on the left. |
| JCColumnLayout( int alignment) | Creates a column layout that aligns components to the specified alignment: SwingConstants.LEFT, SwingConstants.CENTER, or SwingConstants.RIGHT |

###### JCElasticLayout

Use its constructors to provide the layout you want.

|  |  |
| --- | --- |
| JCElasticLayout (int orientation) | Creates a row layout that by default aligns components to the left of the row or column. |

|  |  |
| --- | --- |
| JCElasticLayout (int orientation, int alignment) | Creates a column layout that aligns components as specified by the second parameter, which can be one of the SwingConstants.LEFT, SwingConstants.RIGHT, SwingConstants.TOP, SwingConstants.BOTTOM, or SwingConstants.CENTER. |

When adding an elastic constraint to an object, you can use one of these constants: JCElasticLayout.HORIZONTALLY\_ELASTIC\_CONSTRAINT, JCElasticLayout.VERTICALLY\_ELASTIC\_CONSTRAINT, JCElasticLayout.COMPLETELY\_ELASTIC\_CONSTRAINT

For example:

add(c, JCElasticLayout.HORIZONTALLY\_ELASTIC\_CONSTRAINT);

###### JCGridLayout

Like GridLayout in the AWT, JCGridLayout has a two-parameter constructor in which you specify the number of rows and columns for your grid, and a four-parameter version in which you specify horizontal and vertical gaps as well. Use this constructor just as you would a GridLayout. Unlike the AWT’s GridLayout, JCGridLayout’s rows may have different heights and its columns may have different widths. See the example later on in this chapter for a visual comparison between the two layout managers.

* + 1. **Associated Components**

###### JCBox

|  |  |
| --- | --- |
| JCBox() | Creates a horizontal JCBox container. The constructor may have an optional parameter, int orientation. In this case, valid values are JCBox.HORIZONTAL or JCBox.VERTICAL. |
| createHorizontalBox() createVerticalBox() | Convenience methods for creating JCBoxes. |
| getAlignment() setAlignment(int  alignment) | Describes how the component is aligned. The alignment parameter is one of SwingConstants.LEFT, SwingConstants.CENTER, or SwingConstants.RIGHT. |
| getOrientation() setOrientation(int  orientation) | The box uses the orientation to determine whether its components are arranged horizontally or vertically. |

**JCBrace**

|  |  |
| --- | --- |
| getOrientation() setOrientation(int  orientation) | The parameter in the set method can be one of JCElasticLayout.HORIZONTAL or JCElasticLayout.VERTICAL, specifying how the container is to lay out its components. |
| getLength() setLength(int length) | This is the value of both the minimum size and the preferred size. Whether the length refers to a horizontal or a vertical dimension depends on the orientation. |

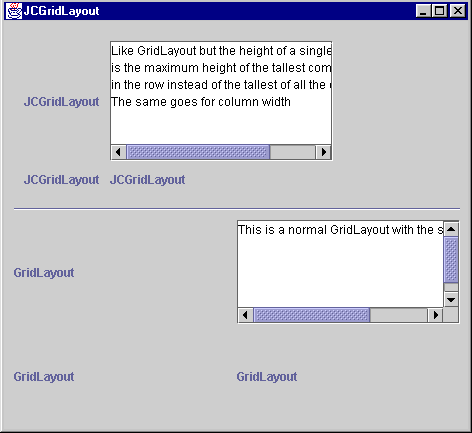
**JCSpring**

|  |  |
| --- | --- |
| get/setHorizontal Elasticity()  get/setVertical Elasticity() | The set methods take an integer parameter specifying the relative elasticity of the JCSpring. When two or more springs are used, the elasticities are used as weighted values for the springiness. |
| get/setMaximumSize() get/setMinimumSize() get/setPreferredSize() | The set methods require a Dimension parameter, which the get methods return. |

#### Examples

**JCGridLayout**

The example shown here illustrates the difference between AWT’s GridLayout and JClass Elements’ JCGridLayout, which conserves space by permitting rows to have different heights and columns to have different widths. The height of each row is determined by the height of the tallest component in that row, and the width of a column is determined by widest component in the column, independent of the width of other columns. With JCGridLayout, rows have varying heights and columns have varying widths.



*Figure 25 A comparison of JCGridLayout and GridLayout.*

###### JCAlignLayout

This layout manager makes it easy to provide a vertical arrangement of data input fields and their associated labels. You can provide for more than two columns and, as the example shows, you aren’t restricted to text fields.

Although its intended use is one with labels in the first column, JCAlignLayout lets you place any component in any column.

Use it as you would any layout manager for a frame or panel:

JCAlignLayout layout = new JCAlignLayout(2, 5, 5); setLayout(layout);



*Figure 26 Using JCAlignLayout.*

*PartII*

*Utility Classes*

***16***

## Introduction to the Utility Classes

[*Utilities*](#_bookmark550)

#### 16.1 Utilities

JClass Elements’s utilities live for the most part in two packages: com.klg.jclass.util, and com.klg.jclass.util.swing. Some components, like JCTreeTable, rely on support classes found in a util subpackage, in this case in com.klg.jclass.util.treetable, and JCDateChooser has a package of its own, com.klg.jclass.util.calendar.

Here is a brief description of the utility classes:

|  |  |
| --- | --- |
| **Name of Utility** | **Description** |
| JCDebug | Place debug statements in your Java code that contain an optional level indication. Use a print level setting to control the detail in your debugging printout. |
| JCIconCreator | There are times when you would like to have a custom image as part of your toolbars, labels, buttons, and so on, yet you don’t want to go to the trouble of using a paint package. JCIconCreator lets you use String arrays to create image icons. The format is similar to that used in *XPixMap* (XPM) format files. |

##### 121

|  |  |
| --- | --- |
| **Name of Utility** | **Description** |
| JCEncode  Component | You can encode the image information for any component in your application with this utility. Since all the children of the chosen component are also encoded, you can capture a picture of your entire user interface for any well-behaved component hierarchy, or any single one of its child components. The utility encodes images in the public-domain Portable Network Graphics (PNG) format.  Note: If you wish to export your images in GIF format, you’ll need a license from the copyright holder, Unisys Corp. Quest Software will send you the GIF encoder class upon receipt of a copy of your license from Unisys.  You need JClass PageLayout to encode components in EPS, PS, PDF, or PCL. JClass PageLayout is available as a part of the JClass DesktopViews suite. |
| JCListenerList | JCListenerList is a class that assists with keeping track of event listeners in a thread-safe manner. |
| JCMappingSort | Sorting can be accomplished by indexing the list of objects that are going to be ordered according to some comparison policy. It can be much more efficient to sort these indices instead of sorting the objects themselves. The idea is to form an array of indices.  The utility is documented with JCSortableTable. |
| JCProgress  Helper | JCProgressHelper is a class that lets you create and manage a thread-safe progress dialog. With it, you can monitor a potentially time-consuming operation and present a visual record of its progress. If it the operation will take more than a specified time, a progress dialog will be popped up. |
| JCString  Tokenizer | JCStringTokenizer provides simple linear tokenization of a String. The set of delimiters, which defaults to common whitespace characters, can be specified either during creation or on a per-token basis. It is similar to java.util.StringTokenizer, but delimiters can be included as literals by preceding them with a backslash character (the default). It also fixes a known problem: if one delimiter immediately follows another, a null String is returned as the token instead of being skipped over. |
| JCSwing  Utilities | This class currently contains a single method: setEnabled(), which takes a Component and a Boolean as parameters. It uses the Boolean parameter to enable or disable all the children of the component, which are themselves Components. |

|  |  |
| --- | --- |
| **Name of Utility** | **Description** |
| Thread Safety | The JCSwingRunnable class helps you manage your threads by providing a class that you can subclass easily. It provides methods that simplify handling execution in the proper threads. |
| JCTreeSet | This class adds to Swing’s functionality by providing you with a way of representing the elements of a set as a binary-sorted tree. If the elements of the set have an defined ordering principle, it is used by default to construct the B-tree, but other ordering mechanisms are possible. |
| JCTypeConverter  and  JCSwingType  Converter | You frequently need to convert objects to Strings, and Strings to objects, when you are coding a user interface. For example, a user types a String as input that you would like to convert to an object. The input String might consist of a sequence of integers, delimited by commas, that you would like to convert to an array. There are also times when you need to convert an object to a String so that you can place the text on a label or a button. The JClass type converters are a collection of the most useful conversions from String to object, and from object to String. The static methods of JCTypeConverter let you retrieve parameters from an application or applet, and convert these parameters to particular data types. |
| JCWordWrap | JCWordWrap provides a static method called wrapText that performs basic word-wrap logic on a String, given a line width and new line delimiter. |

Chapter 16 Introduction to the Utility Classes **123**

# 17

## Debugging Tools

[*Features of JCDebug*](#_bookmark567)[*Class*](#_bookmark571)[*Methods*](#_bookmark572)[*Examples*](#_bookmark586)

#### 17.1 Features of JCDebug

JCDebug is a utility class that allows you to add debugging statements to your source code, in order to facilitate development tasks. Once the statements are in the code, the debugging mechanism can easily be turned on or off. To activate normal debug output, simply call setEnabled(true); to turn off debugging without removing it from the source, call setEnabled(false) to globally disable JCDebug printing.

It is also possible to group debugging statements by providing a tag parameter along with the text to be printed. For example:

JCDebug.println("tag1", "This is printed when setEnabled(true) and setTag(\"tag1\") are in effect.").

Thus, if you wish to enable print statements marked by parameter tag1, call setTag("tag1"). If you wish to turn on debugging print statements with various tags, call setTags(String new\_tags[]), passing in an array of tag names. Initially, the array of tag names is null, which means that no tagged statements will be output. Thus, once some tags have been set, they can all be turned off by calling setTag(null).

Another useful construct is the concept of “levels.” The advantage of this is that by calling, for example, setLevel(2), you will avoid seeing any debug messages marked 3 or below. This is useful for controlling the amount of detail you are viewing without removing the debug information.

Note that JClass distribution bytecode does *not* contain any references to JCDebug. JCDebug helps you accomplish the following:

Place multi-level print statements in your code for debugging purposes and remove

them after testing is complete.

Force a stack trace to occur at any point in your code.

##### 125

#### Class

com.klg.jclass.util.JCDebug contains static methods for assertions and debug statements.

#### Methods

###### Printing debug information

You can define a numerical order-of-importance to your print statements. Print statements labeled with lower numbers are deemed to be more important than those with higher- numbered labels. By setting a global print level variable at 3, for example, all print statements labeled with a number higher than three will be ignored. Those labeled with a print level variable of 1, 2, or 3 will all be printed.

Also, it’s possible to supply a list of tags. All print statements with print level variables matching a String in the list will be processed, all other print statements will be ignored.

|  |  |
| --- | --- |
| setPrintStream() | Sets the output stream to use. The default is System.out. |
| getPrintStream() | Returns the PrintStream currently in use. |
| println() | Prints debug information on the current output stream. |
| println  (String text) | Always prints a message, unless debugging is turned off. |
| println(int  plevel, String text) | The int parameter is for level-controlled diagnostic printing. Any print statement with a plevel greater than the currently set print level is not printed. |
| println(String ptag, String text) | The ptag parameter for tag-controlled diagnostic printing. Print statements whose ptag matches a tag in the tag list are printed. |
| println(int  plevel, String ptag, String text) | Prints the text according to level and tag filter options. This method is a combination of println(int plevel, String text) and println(Sting ptag, String text). |
| setEnabled(true) | Activates debug output. |
| isEnabled() | Determines if debugging is on or off. |
| setLevel(int new\_level) | Output occurs for statements marked with the specified level or lower. To see all debug statements, set new\_level to a very large number. |

|  |  |
| --- | --- |
| getLevel() | Returns the level that determines what gets printed. All levels less than or equal to the returned integer are printed. |
| setTag("myString"  ) | Sets one tag. A second call to setTag() causes the first tag to be forgotten. |
| setTags(arrayOfSt rings) | Sets the array of tags to use for the debugging session. |

Notes on tags and level numbers:

If you do not supply a tag as one of the parameters in your print statement, the tag is deemed to be null. Statements with null tags are always printed as long as debugging is enabled.

As long as you get a match from the level number or the tags you’ll have some diagnostic printout.

You don’t have to use either tags or level numbers. You can simply use unadorned JCDebug.println statements. In this case, you don’t have any selectivity other than being able to turn debugging on and off.

###### Forcing a stack trace

The following methods help force a stack trace:

|  |  |
| --- | --- |
| printStackTrace() | Forces a stack trace at the current location. |
| printStackTrace( String s) | Forces a stack trace at the current location, and prints an identifying message as a header. |
| printStackTrace( String ptag, Throwable t) | Prints a stack trace for the specified exception if the specified tag is currently enabled. |

Chapter 17 Debugging Tools **127**

#### Examples

This example illustrates the following points about the use of JCDebug:

JCDebug.setEnabled(true) must be in effect for the debug mechanism to be turned on.

If tags are used, JCDebug.setTag() controls which tagged print statements are active. The example uses tag2 and tag3 as arbitrary labels. Print statements involving tag2 will be active, but those involving tag3 will not.

import com.klg.jclass.util.JCDebug; public class TestJCDebug {

public static void main(String args[])

{

JCDebug.setEnabled(true); System.out.println("Starting the test:");

//Set a tag so that all JCDebug statements with this tag will print JCDebug.setTag("tag2");

//These should print

JCDebug.println("Debugging is on so this should be printed!");

JCDebug.println("tag2", "Label tag2 is enabled, " + "so this should be printed.");

//This should not print because it does not match the tag JCDebug.println("tag3", "The tag for this print statement is tag3 " +

"so this should not be printed.");

//Now turn off debugging JCDebug.setEnabled(false);

//The following two lines will not be printed. JCDebug.println("Debugging is off. This should not be printed!"); JCDebug.println("tag2", "This label is enabled, but debugging is off, "

+ "so this should not be printed");

}

}

The output from this test program is reproduced here.

Starting the test:

Debugging is on so this should be printed!

Label tag2 is enabled, so this should be printed.

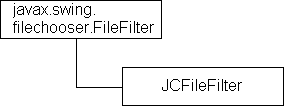
# 18

## JCFileFilter

[*Features of JCFileFilter*](#_bookmark591)[*Constructors*](#_bookmark592)[*Methods*](#_bookmark594)[*Example*](#_bookmark602)

#### Features of JCFileFilter

JCFileFilter enhances Swing’s JFileChooser by allowing you to pass it file extensions. These are the only ones that appear in the file chooser dialog. Extensions are of the type “.txt,” “.java,” and so on, which are typically found on Windows and Unix platforms, but not on the Macintosh. Case is ignored, so “.txt” is equivalent to “.TXT” as far as the filter is concerned.



The class has versatile constructors and convenience methods for setting both the extensions that are to be filtered and an optional descriptive phrase.

#### Constructors

JCFileFilter has four constructors:

|  |  |
| --- | --- |
| JCFileFilter() | The default constructor. This form of the constructor is used to list all file types. To filter files, use the addExtension() method. To provide a human-readable description for the extension, use the setDescription() method. |
| JCFileFilter(String extension) | Creates a file filter that, initially at least, shows only files of the type whose extension is specified by the String argument. The period (dot) before the extension is optional. More filename extensions may be added using the addExtension() method. |

##### 129

|  |  |
| --- | --- |
| JCFileFilter(String[] filters) | Creates a file filter for a list of file extensions given as a String array. More may be added using the addExtension() method. |
| JCFileFilter(String extension, String description) | Creates a file filter that, initially at least, shows only files of the type whose extension is specified by the String argument. The period (dot) before the extension is optional. More filename extensions may be added using the addExtension() method.  The human-readable description is retrieved using the  getDescription() method. |

#### Methods

These methods let you set or examine the elements of the filtering process:

|  |  |
| --- | --- |
| addExtension(String extension) | Places an additional filename extension in the list of those to filter against. Any filename matching this extension is listed in the file dialog, as well as those that have been added in the constructor. The method may be used multiple times to form a list of extensions to filter against. |
| accept(File f) | This Boolean method is used to determine if the given filename has an extension that is in the list of those to be filtered, that is, to be displayed in the file dialog.  Files that begin with “.” are ignored, but directories are always shown. |
| getExtension(Strings) | Returns the extension portion of a String. |
| set/getDescription() | Sets or gets the human readable description of this filter. This String is used to preface the list of file extensions that shows up in the file dialog’s “Files of type:” combo box.  For example, if the filter is set to:  String[] filterTypes = {"gif", "jpg"};  and the description is specified as "JPEG & GIF Images", then the combo box displays “JPEG & GIF Images (\*.gif,  \*.jpg)” |
| setExtensionListIn Description(  boolean b) isExtensionListIn  Description() | Determines whether the extension list, for example (\*.jpg,  \*.gif), should show up in the human-readable description. The corresponding Boolean method returns the policy currently in effect. |

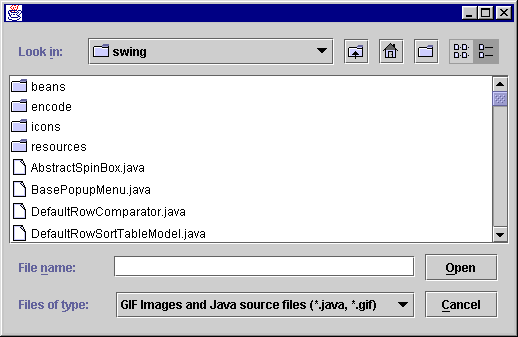
#### Example

The following code snippet sets up a filter for GIF and Java source files. The description, which appears in the file chooser’s “Files of type:” combo box along with the extensions themselves, is provided through a parameter passed to the constructor.

Note that you can control whether either part of the description actually appears through the use of setDescription() and setExtensionListInDescription().

JFileChooser chooser = new JFileChooser(); String[] filterTypes = {"gif", "java"}; JCFileFilter filter =

new JCFileFilter(filterTypes, "GIF Images and Java source files"); chooser.addChoosableFileFilter(filter);



*Figure 27 A JCFileFilter for Java source files and GIF images.*

Chapter 18 JCFileFilter **131**

# 19

## Icon Creator

[*Features of JCIconCreator*](#_bookmark606)[*Classes*](#_bookmark609)[*Constructors and Methods*](#_bookmark611)[*Examples*](#_bookmark618)

#### Features of JCIconCreator

There are times when you would like to have a custom image as part of your toolbars, labels, buttons, and so on, yet you don’t want to go to the trouble of using a paint package. JCIconCreator lets you use String arrays to create image icons. The advantages of using JCIconCreator include:

A simple and convenient way of defining an image from a String of characters.

Keeping the image information in the class that uses it, rather than having to manage the location of associated image files.

Designing small-sized custom images or diagrams without the need of a paint program.

Having a simple way of associating the image with the standard

javax.swing.ImageIcon class.

#### Classes

This utility consists of a single class, com.klg.jclass.util.swing.JCIconCreator,   
subclassed from java.lang.Object.

#### Constructors and Methods

The JCIconCreator has two constructors: JCIconCreator() and JCIconCreator(int w, int h). JCIconCreator() creates an uninitialized image icon. JCIconCreator(int w, int h) creates an image icon of the specified width and height (in pixels). The parameters should match the width and height of the two dimensional array used to hold the characters representing the image.

##### 133

###### Methods in JCIconCreator

The following is a list of the methods available for JCIconCreator:

|  |  |
| --- | --- |
| clear() | Clears the icon so that no image is associated with it. |
| getIcon() | Gets the icon created by this instance of JCIconCreator. An overloaded version of this method takes a passed-in byte array (as might be obtained from a database's image field) and attempts to convert it into an Image.  Use getIcon() method to return an ImageIcon. For example:  ImageIcon myLabelIcon = ic.getIcon(); |
| setColor() | Sets the color corresponding to a character passed as its first parameter. Its second parameter is a Color object or an RGB int.  Use the setColor() method to associate a character in the array with a color. For example, if ic is an instance of a JCIconCreator, ic.setColor('G', Color.green);  associates a “G” in the pixel map with a green pixel. |
| setPixels() | Sets the pixel data. If its parameter is an array of Strings, this represents the data for all rows. If the parameters are a row index and a String, this represents the pixel data for one row.  If pixelMap is an array of characters representing pixels, inform the instance about them with  ic.setPixels(pixelMap); |
| setSize() | Width and height int parameters are used to set the width and height for the image. |

#### Examples

The following code section shows how to declare a String array, use it as the source for defining the pixels in an icon, and how to convert the JCIconCreator object to an ImageIcon for use as the graphic part of a label.

...

private static final String testIcon[] = { " BBBBBBBBB ",

* B OOO B ", " B OOOOO B ", " B OOOOO B ", " B OOOOO B ", " B OOO B ",
* B B ",
* B B ",
* B B ",
* B B ",
* B B ",
* B B ",
* BBBBBBBBB " };

JButton b1;

public ToolbarIcons() { JToolBar bar; JLabel label;

setBackground(Color.lightGray); setLayout(new BorderLayout());

JCIconCreator ic = new JCIconCreator(13, 13); ic.setColor('B', Color.black); ic.setColor('O', Color.orange); ic.setPixels(testIcon);

ImageIcon icon = ic.getIcon();

...

bar = new JToolBar();

b1 = new JButton("Caution", icon); bar.add(b1);

...



*Figure 28 Three labels with custom icons created using JCIconCreator.*

Chapter 19 Icon Creator **135**

# 20

## Image Encoder

[*Features of JCEncodeComponent*](#_bookmark622)[*Classes and Interfaces*](#_bookmark624)[*Constructors and Methods*](#_bookmark626)[*Examples*](#_bookmark633)

#### Features of JCEncodeComponent

You can encode the image information for any component in your application with this utility. Since all the children of the chosen component are also encoded, you can capture a picture of your entire user interface for any well-behaved component hierarchy, or any single one of its child components. The utility encodes images in the public-domain Portable Network Graphics (PNG) format. Other common formats are available if you also have JClass PageLayout installed. JClass PageLayout is available as part of the JClass DesktopViews suite.

The advantages of using JCEncodeComponent include: Saving an image of a component in PNG format.

A simple way to encode a component: just call JCEncodeComponent.encode().

Please note that the JPEG format is not supported because it loses information as a result of the compression.

Note: If you wish to export your images in GIF format, you’ll need a license from the copyright holder, Unisys Corp. Quest Software will send you the GIF encoder class upon receipt of a copy of your license from Unisys.

#### Classes and Interfaces

The com.klg.jclass.util.swing.encode package contains an interface, a main class called JCEncodeComponent, and various helper classes that output the various supported image formats.

The interface that defines an image encoder contains a single method: encode(). Its parameters are the component whose image is to be encoded, and the stream on which to place the encoded information.

There is also an exception class, EncoderException. It is raised by JCEncodeComponent or one of its subclasses. The exception may be subclassed for exceptions thrown by

##### 137

subclasses of JCEncodeComponent. It represents any problem encountered while encoding an image. The message is used to state the type of error.

JCEncodeComponent has a public static inner class named Encoding that is used to provide instances of the various valid encodings or to supply an error message if an attempt fails.

#### Constructors and Methods

###### JCEncodeComponent

The Encode inner class is used to instantiate a particular encoding type, such as PDF. It defines methods that provide information about the encoder, including a failure message if the encoder fails to load.

|  |  |
| --- | --- |
| getEncoder() | Returns an encoder for this encoding type. |
| getFailureMessage() | Message stating possible reasons for encoder load failure. |
| getLongName() | Returns the fully qualified name of the supported encoding type. |
| getShortName() | Returns the descriptive name of the supported encoding type. |
| toString() | Returns both the short name and the long name in a single String. |

The array called ENCODINGS contains, as instances of Encode, the supported encoding types. You pass an element of this array to the encode method, along with your component and a Stream specifier, to produce an encoding of the component which is sent to the stream. The method is overloaded so that you can write the information to a file if you wish.

|  |  |
| --- | --- |
| encode(Encoding encoding, Component component, OutputStream output) | Invoke this method on a Java component to encode its image in the specified format, and send the encoded information to the specified stream. |
| encode(Encoding encoding, Component component,  File file) | Encodes the component’s image in the specified format. Sends the encoded information to the specified file. |

#### Examples

Below is an example that encodes an entire frame in PNG format, then stores the result in a file. Most of the code simply serves to create a frame containing a few components.

Since the process of encoding can result in an exception being thrown, the single-line command that does all the work is enclosed in a try block.

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

import java.io.File; import javax.swing.JPanel; import javax.swing.JLabel;

import javax.swing.JButton; import javax.swing.JTextField;

public class EncoderExample {

public static void main(String[] args) { JCExitFrame eFrame;

eFrame = new JCExitFrame("Encoder Example"); JPanel jp = new JPanel();

JLabel jl = new JLabel("PNG Encoding"); JButton jb = new JButton("Just a button"); JTextField jt = new JTextField(

"The entire frame will be encoded");

jp.add(jl);

jp.add(jb);

jp.add(jt); jp.setVisible(true);

eFrame.getContentPane().add(jp); eFrame.setSize(350, 100); eFrame.setVisible(true);

File efps = new File("efps.png"); try {

JCEncodeComponent.encode(JCEncodeComponent.PNG,

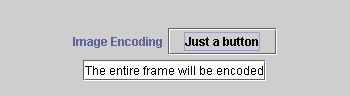
eFrame, efps);

} catch (Exception e) { System.out.println("Exception caught: " + e);

}

}

}



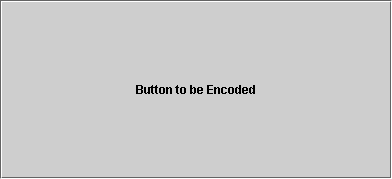
*Figure 29 The result of encoding the entire JCExitFrame.*

You can find another example, *Encode.java*, in the *examples/elements* directory. In that example, a single component, a button, is encoded. A combo box lets you choose the encoding format, a text field displays the current choice, and a button-press initiates encoding to a file. This example is more realistic in that the encoding process is initiated

Chapter 20 Image Encoder **139**

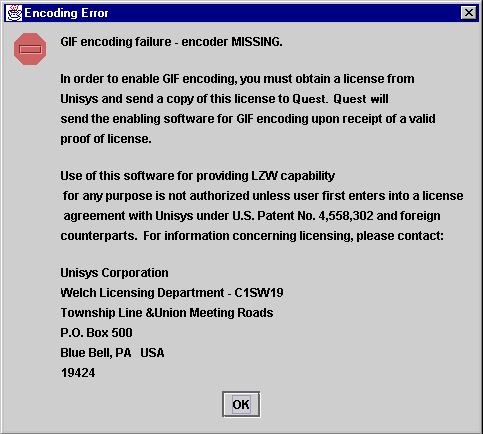
by the end user through some action, such as a menu choice, or, as in this case, by pressing a button.

The result is shown in the next figure.



*Figure 30 Encoding a single component using examples.elements.Encode.*

If you attempt to encode a component using a GIF format, you will see the following error dialog:



*Figure 31 The error dialog that appears if you do not have GIF encoding installed.*

Note: You need JClass PageLayout to encode components in EPS, PS, PDF, or PCL. JClass PageLayout is available as a part of the JClass DesktopViews suite.

# 21

## Listener List

[*Features of JCListenerList*](#_bookmark637)[*Classes*](#_bookmark638)[*Examples*](#_bookmark645)

#### Features of JCListenerList

JCListenerList is a class that assists with keeping track of event listeners in a thread-safe manner. The use of static methods on the JCListenerList class prevents any problems from occurring if the list being modified is null. To send events to the listener in the list, simply get the JCListenerListEnumeration of the list and walk through the elements.

There is no ordering guarantee.

#### Classes

The following is a list of the JCListenerList classes:

|  |  |
| --- | --- |
| JCListenerList | The thread-safe listener list class. |
| JClistenerListEnumeration | Implements java.util.Enumeration and takes a JCListenerList as its parameter. It defines methods hasMoreElements() and nextElement(). |

#### Methods

The following is a list of the JCListenerList methods:

|  |  |
| --- | --- |
| add() | Adds an element to the list of listeners. |
| remove() | Removes an element to the list of listeners. |
| elements() | Returns an Enumeration object. |

#### Examples

To add a listener using a JCListenerList: JCListenerList someList = null;

...

public synchronized void addSomeListener(SomeListener l) { someList = JCListenerList.add(someList, l);

}

To remove a listener:

public synchronized void removeSomeListener(SomeListener l) { someList = JCListenerList.remove(someList, l);

}

The use of static methods on the JCListenerList class prevents any problems from occurring if the list being modified is null.

To send events to the listener in the list, simply get the Enumeration of the list and walk through the elements. There is no ordering guarantee.

# 22

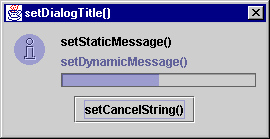
## Progress Helper

[*Features of JCProgressHelper*](#_bookmark651)[*Constructors and Associated Classes*](#_bookmark653)

[*JCProgressHelper Methods*](#_bookmark664)[*Examples*](#_bookmark679)

#### Features of JCProgressHelper

JCProgressHelper is a class that lets you create and manage a thread-safe progress dialog. With it, you can monitor some potentially time-consuming operation and present a visual indication of its progress. If it looks like the operation will take some time, a progress dialog appears. Before the operation is started the JCProgressHelper should be given a numeric range and a descriptive String. Initially, there is no JProgressBar. As the operation progresses, call the updateProgress() method to indicate how far along the [*min .. max*] range the operation is. After the first timeToDecideToPopup milliseconds (default 500) the progress monitor will predict how long the operation will take. If it is longer than timeToPopup (default 2 seconds) a JProgressBar is popped up.



*Figure 32 A JCProgressHelper showing the methods used for labelling.*

The advantages of JCProgressHelper include:

You are able to quantify the process that the JCProgressHelper is monitoring by setting two integers representing a minimum and a maximum of some value that proportionately measures the progress of some time-consuming operation. As the process continues, you call updateProgress() with a parameter indicating how far along things are.

The progress helper transforms all your calls to it into Thread-safe calls to the parent Swing component to encourage frequent updating.

A descriptive, dynamically updatable, message informs users about the progress of the operation.

The progress dialog contains a Cancel button, permitting the end-user to terminate long-running processes.

The progress dialog waits for a time that you set in setTimeToDecideToPopup() before checking whether to pop up, and does not pop up at all unless the operation is projected to take at least a minimum time, which you may set also, in setPopupTime().

#### Constructors and Associated Classes

##### Constructors

JCProgressHelper(Component parent) JCProgressHelper(Component parent,

String static\_message, int min,

int max)

JCProgressHelper(Component parent,

String static\_message, int min,

int max,

boolean show\_dynamic\_message, boolean is\_modal,

boolean is\_dismissable)

The parameters in the constructors are:

|  |  |
| --- | --- |
| parent | The object whose computations are to be monitored. |
| min, max | Integer parameters that represent a time scale for the operation. The progress bar is scaled between these values. Thus, if min = 40 and max = 90, and the current value is 50, then (50 - 40)/(90 -  40) = 1/5, and 1/5, that is 20%, of the length of the progress bar is shaded. |
| static\_message | The first message line. This message cannot be changed once the dialog has popped up. |
| show\_dynamic  \_message | The changeable part of the message in the line below the static message of the progress dialog can be turned on and off using this Boolean parameter. |
| is\_modal | If true, the progress dialog must remain the active window. |
| is\_dismissable | If false, the user cannot cancel the computation that this progress meter is monitoring. |

##### Associated Classes

Following is a list of the JCProgressHelper associated classes:

|  |  |
| --- | --- |
| JCProgressEvent | JCProgressEvent is used to monitor the status of a process. The event contains information about the process name, the current unit being processed, and the total unit count for the process. The setAbort() method allows the listener to abort the process, and the isAborted() method checks to see if the process is aborted. |
| JCProgress  Listener | The listener interface that may be used for processes which are to be monitored by a progress bar. Methods are: processingBegin(), invoked when a process has begun, processingEnd(), invoked when the process has been compeleted, processingError(), invoked when a process encounters an error, and processingUnit(), invoked when a process unit has been completed. |
| JCProgress  CancelledEvent | JCProgressCancelledEvent is used to notify interested listeners when the user has cancelled progress via the Cancel button on the JCProgressHelper. |
| JCProgress  CancelledListener | The listener interface which is used to detect a user- cancellation action in the JCProgressHelper.  JCProgressCancelledEvent is fired when the user has cancelled progress via the Cancel button on the JCProgressHelper. Implementations of this interface may detect the cancellation in their application and react to it. |
| JCProgressHelper | The progress monitor itself. |
| JCProgress  AbortedException | Used to create an exception that tells the process that it should exit. |
| JCProgressAdapter | An abstract class that acts as an adapter. It provides null implementations of all the methods defined in JCProgressListener. |

##### Using the Event and Listener Classes

Depending on your needs, there are four ways to use the progress mechanism:

Use the JCProgressHelper GUI and let it handle all updates without any need to invoke events and listeners in your code. In this case, you call startProgress() and updateProgress() to control the progress bar. See *BasicProgressHelperExample.java* in the *JCLASS\_HOME/examples/elements* directory.

Use createProgressListener() to have JCProgressHelper manage events internally. In this case, you call listener methods like processingBegin() and processingUnit(), which are implemented by the progress meter itself. You don’t have to supply the code.

Create your own addProgressListener() method and have it register JCProgressListeners. This option gives you the most control. You’ll need to have one of your classes implement JCProgressListener, or you can extend JCProgressAdapter. You override the listener methods you need, and you will have to provide implementations of addProgressListener(JCProgressListener), removeProgressListener(JCProgressListener), and fireProgressEvent(JCProgressEvent) in the class that wishes to control the progress meter. See *ProgressListenerExample.java* in the *JCLASS\_HOME/examples/elements* directory.

Use an implementation of JCProgressCancelledListener to handle what must be done if the end user clicks the JCProgressHelper’s Cancel button. A JCProgressCancelledEvent is sent to all JCProgressCancelledListeners when this button is clicked. If you use this listener your application is notified immediately when the Cancel button has been clicked, rather than having to wait until processingUnit() is called from JCProgressListener.

#### JCProgressHelper Methods

|  |  |
| --- | --- |
| completeProgress() | Call this method when your computation has finished to allow for cleanup. |
| isOkayToContinue() | Is false if the user has pressed the Cancel button. You can use it in your code to force a cancellation of the computation. |
| setDynamicMessage() | The changeable part of the progress meter’s message. |
| setCancelString | Sets the String in the Cancel button. Default is “Cancel”. |
| setDialogTitle | Sets the String in the dialog’s title bar. Default is “Progress...” |
| setMaximum() | You model the duration of a process by inventing an integer range *min .. max*. Choose the range so that it is easy to calculate the integer that represents your computation’s degree of completion. Set the maximum-time integer using this method. |
| setMinimum() | Use this method to set the minimum-time integer. |
| setRange() | A convenience method that combines setMinimum and  setMaximum. |

|  |  |
| --- | --- |
| setStaticMessage() | Use this method to define the unchanging part of the progress meter’s message. |
| setTimeToDecideTo Popup() | The progress meter waits until this time before attempting to predict how long the process it is monitoring will take. It then uses the time set in setTimeToPopup() to decide whether to pop up at all. |
| setTimeToPopup() | The progress meter won’t pop up if the progress meter’s calculation estimates that the process will take less time than the time you set here. The default is 500 ms. |
| startProgress() | Call this method within your object to inform the progress meter that timing has begun. |
| updateProgress() | Call this method with a integer parameter that represents your computation’s degree of completion. |

#### Examples

Add a JCProgressHelper to your component as follows:

JCProgressHelper jpr = new JCProgressHelper(eFrame,

"Here is a progress message", 0, 10,

true, true, true);

Set a time which it is unnecessary to display a progress meter and call setPopupTime() with this value. Decide when you want the progress helper to calculate its estimation of the monitored process’ completion time and call setTimeToDecideToPopup() with this time. This time should be long enough that the tracking variable is no longer at its minimum value, so that the progress meter has a way of estimating how long the operation will take to complete.

Call the updateProgress() method periodically to update the tracking variable:

jpr.updateProgress(5); // Progress indicator is half-way along

When the process has completed, call completeProgress() to remove the dialog.

###### A full example

The following code causes a progress dialog to appear. It sets both a static message and a dynamic message that is changed every time the progress bar is updated. Since the dialog is managed in an AWT thread, you may have to make sure that it is given a chance to

run, especially if you wish to perform some initialization of the progress dialog in the mainline thread and you wish it to appear in the dialog.

import javax.swing.\*;

import com.klg.jclass.util.swing.JCProgressHelper; public class ProgressHelper extends JFrame {

// Use the constructor that permits setting a static message public ProgressHelper() {

JCProgressHelper jpr = new JCProgressHelper(this,

"Here is a static progress message", 0, 100,

true, true, false);

// Pop up the progress dialog.

// There will be a delay, determined in part

// by the two popup time parameters. jpr.startProgress();

for (int j = 1; j < 11; j++){

// Simulate an ongoing process ... try {

Thread.sleep(1000);

} catch (Exception e) {

}

// ... and update the progress meter periodically, jpr.updateProgress(j\*10);

// changing the dynamic message as the meter updates. jpr.setDynamicMessage("Dynamic "+ j\*10);

}

// Dispose of the progress meter jpr.completeProgress();

// The mainline program can continue as required...

// ... until its tasks are completed. System.exit(0);

}

public static void main(String[] args) { ProgressHelper ph = new ProgressHelper();

}

}

# 23

## String Tokenizer

[*Features of JCStringTokenizer*](#_bookmark683)[*Classes*](#_bookmark684)[*Methods*](#_bookmark686)[*Examples*](#_bookmark695)

#### Features of JCStringTokenizer

JCStringTokenizer provides simple linear tokenization of a String. The set of delimiters, which defaults to common whitespace characters, can be specified either during creation or on a per-token basis. It is similar to java.util.StringTokenizer, but delimiters can be included as literals by preceding them with a backslash character (the default). It exhibits this useful behavior: if one delimiter immediately follows another, a null String is returned as the token instead of being skipped over.

JCStringTokenizer has these capabilities: Parses a String using a delimiter you specify.

Parses a String using the specified delimiter and escape character. Counts the number of tokens in the String using the specified delimiter.

#### Classes

This utility consists of a single class called JCStringTokenizer. Pass the String to be tokenized to the constructor:

String s = "Hello my friend";

JCStringTokenizer st = new JCStringTokenizer(s);

Process the tokens in the String tokenizer with methods hasMoreTokens() and

nextToken().

##### 149

#### Methods

These are the methods of JCStringTokenizer:

|  |  |
| --- | --- |
| countTokens() | Returns the next number of tokens in the String using the delimiter you specify. |
| getEscapeChar() | Gets the escape character (default: \). |
| getPosition() | Returns the current scan position within the String. |
| hasMoreTokens() | Used with nextToken(). Returns true if more tokens exist in the String tokenizer. |
| nextToken() | Gets the next token from the delimited String. If required, the delimiter can be “escaped” by a backslash character.  To include a backslash character, precede it by another backslash character. |
| nextToken() | Gets the next whitespace-delimited token. |
| parse() | Given a String a delimiter, and an optional escape character, this method parses the String using the specified delimiter and returns the values in an array of Strings.  Use the second form of the command if you wish to set an escape character different from the default, which is the backslash character. |
| setEscapeChar() | Sets the escape character (default: \). If 0, no escape character is used. |

#### Examples

At one point, there are two side-by-side commas in the String that is to be split into tokens. The delimiter for tokenization is a comma, so a null is returned as the token in this case. Upon encountering it, println() outputs the word “null” as part of the print stream. Note that leading spaces are not stripped from the tokenized word.

String token, s = "this, is, a,, test"; JCStringTokenizer st = new JCStringTokenizer(s); while (st.hasMoreTokens()) {

token = st.nextToken(','); System.out.println(token); }

This prints the following to the console:

this is a null

test

You can remove the leading spaces by passing each token in turn to another String tokenizer whose delimiter is a space.

In the next example, a slightly longer String is parsed based on the delimiter being the space character. As in the previous example, side-by-side spaces are interpreted as having a null token between them.

import com.klg.jclass.util.JCStringTokenizer; public class StringTokenizerExample {

public static void main(String args[]){

String token, s = "this is a test of the string " +

+ "tokenizer called JCStringTokenizer. " +

"\nThe whitespace between the repeated words is a tab tab. "; System.out.println("First, the string: " + s);

JCStringTokenizer st = new JCStringTokenizer(s); while (st.hasMoreTokens()) {

token = st.nextToken(' '); System.out.println(token);

}

}

}

Chapter 23 String Tokenizer **151**

This time, the output is:

First, the string: this is a test of the string tokenizer called JCStringTokenizer.

The whitespace between the repeated words is a tab tab. this

null null is a null test of the

string tokenizer called

JCStringTokenizer.

The whitespace between the repeated words

is a tab

tab.

# 24

## Thread Safety Utilities

[*Features of the Thread Safety Classes*](#_bookmark699)[*Methods*](#_bookmark704)

#### Features of the Thread Safety Classes

JCMessageHelper lets you build a message dialog based on JOptionPane. The advantages of the JCMessageHelper are:

JCMessageHelper invokes JOptionPane in a thread-safe manner.

You can set an audible indication when the dialog appears as a parameter in the constructor.

JCMessageHelper utilizes JCSwingRunnable, an abstract runnable class that provides the run() method. You can create an object of this type, and call the runSafe method to get it going.

#### Methods

###### JCMessageHelper

JCMessageHelper has static methods that resemble those in [JOptionPane](http://www.javasoft.com/products/jdk/1.2/docs/api/javax/swing/JOptionPane.html). These are:

|  |  |
| --- | --- |
| showError() | Shows an error message, given a title and the message String as parameters. |
| showInformation () | Parameters are two Strings: title and message. The message appears in an information dialog. |
| showWarning() | Parameters are two Strings: title and message. The message appears in a warning dialog. |

|  |  |
| --- | --- |
| showMessage() | There is an optional first parameter that lets you specify a parent Component. The next two parameters are Strings, title and message. The next parameter is an int specifying the message type, and the optional last parameter is a Boolean that specifies whether to emit an audible beep when the dialog appears. The possible message types are  javax.swing.JOptionPane.ERROR\_MESSAGE javax.swing.JOptionPane.INFORMATION\_MESSAGE javax.swing.JOptionPane.WARNING\_MESSAGE javax.swing.JOptionPane.QUESTION\_MESSAGE javax.swing.JOptionPane.PLAIN\_MESSAGE  See javax.swing.JOptionPane for a description of the various dialogs. |

###### JCSwingRunnable

|  |  |
| --- | --- |
| run() | Since this class implements Runnable and is used to create a thread, its run() method will be called to start the thread. |

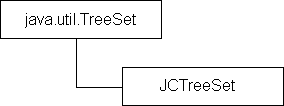
***25***

**Tree Set**

[*Features of JCTreeSet*](#_bookmark715)[*Constructors and Methods*](#_bookmark716)[*Examples*](#_bookmark718)

#### Features of JCTreeSet

This class adds a convenience constructor to java.util.TreeSet, which provides you with a way of representing an array of Objects as a sorted set. If the elements of the set have a defined ordering principle, it is used by default to construct the tree, or you can provide a Comparator. The ordering must be compatible with the conditions for a java.util.TreeSet. If your array contains duplicate items, JCTreeSet will ensure that only one of them is placed in the sorted set.



JCTreeSet adds this convenience constructor:

Construct a new JCTreeSet containing the elements in a specified array, sorted according to the elements’ natural ordering principle.

JCTreeSet includes the following TreeSet standard constructors:

Construct an empty TreeSet using a specified Collection, sorted according to the elements’ natural ordering principle.

Construct a new TreeSet containing the elements in the specified Collection, sorted according to the elements’ natural ordering principle.

Construct a new JCTreeSet containing the same elements as the given SortedSet, sorted according to the same ordering.

#### Constructors and Methods

JCTreeSet has constructors that allow you to form a sorted set from the elements of a Collection, an array of Objects, or a SortedSet. A no-parameter constructor lets you instantiate an empty tree set using natural ordering, or you can supply a Comparator to specify how the elements are to be sorted.

JCTreeSet defines no methods of its own. Its methods are inherited from java.util.TreeSet, java.util.AbstractSet, java.util.AbstractCollection, and java.lang.Object.

#### Examples

The example shown here illustrates passing an array of Strings to the constructor. Some Strings are duplicates, but the resulting sorted set contains no duplicates.

import com.klg.jclass.util.JCTreeSet; public class TreeSetExample {

public static void main(String args[]){ System.out.println("Starting TreeSetExample"); String[] items =

{"moe", "joe", "poe", "zoe", "aoe", "poe", "joe", "moe"}; JCTreeSet ts = new JCTreeSet(items);

System.out.println("The number of items in the array is: "

+ items.length); System.out.println("The number of items in JCTreeSet "

+ ts + " is: " + ts.size()); System.out.println("The last element of " + ts + " is: "

+ ts.last());

}

}

The output of the program is:

Starting TreeSetExample

The number of items in the array is: 8

The number of items in JCTreeSet [aoe, joe, moe, poe, zoe] is: 5 The last element of [aoe, joe, moe, poe, zoe] is: zoe

# 26

## Type Converters

[*Features of JCTypeConverter*](#_bookmark722)[*Features of JCSwingTypeConverter*](#_bookmark723)

[*Classes*](#_bookmark725)[*Methods*](#_bookmark726)[*Examples*](#_bookmark757)

#### Features of JCTypeConverter

There is frequently a need to convert objects to Strings and Strings to objects when you are coding a user interface. For example, a user types a String as input that you would like to convert to an object. The input String might consist of a sequence of integers, delimited by commas, that you would like to convert to an array. There are also times when you need to convert an object to a String so that you can place the text on a label or a button. The JClass type converters are a collection of the most useful conversions from String to object, and from object to String. The static methods of JCTypeConverter let you retrieve parameters from an application or applet and convert these parameters to particular data types.

JCTypeConverter performs these functions:

Returns a trimmed String, with trailing nulls removed.

Converts a String to an integer.

Converts a String to a double.

Converts a String to a Boolean.

Converts a String to an array of Strings.

Converts a String to an array of integers or Integer objects.

Converts a String to an array of Double objects.

Converts all occurrences of "\n" to newlines.

Converts all occurrences of char '\n' to String "\n".

Converts a delimited list of String tokens to a Vector.

Converts a String to an enum, or a list of enums.

Converts an enum to a String.

Converts an object to a String.

Converts a String to a Date.

Removes “escape” characters (backslashes) from the String. Allows parsing errors to be printed or shown in a dialog.

#### Features of JCSwingTypeConverter

JCSwingTypeConverter can perform these functions:

Converts a String to a Color, or an array of Colors.

Converts color to one of the Color enums, or RGB format.

Converts list to a comma-separated list of tokens.

Converts a font name to a font instance, or a Font to a *name-style-size* String, or a

String like *Helvetica-plain-10* to a Font.

Converts a String to an Insets instance, or creates a String from an AWT Insets

value.

Converts a String to a Dimension instance. Converts a String to a Point instance.

#### Classes

The two type converter classes are com.klg.jclass.util.JCTypeConverter and com.klg.jclass.util.swing.JCSwingTypeConverter. Both contain static methods for converting from one standard type to another. JCTypeConverter is for converting Java types, and JCSwingTypeConverter is for Swing types.

#### Methods

###### JCTypeConverter

JCTypeConverter contains static methods for retrieving parameters from a source file or applet, and for converting parameters to particular data types.

The methods in JCTypeConverter are:

|  |  |
| --- | --- |
| checkEnum() | Checks the validity of an enum. |
| error() | Writes a parse error message to the standard output device. |
| fromEnum() | Converts an enum to a String. |
| fromNewLine() | Converts all occurrences of char '\n' to String "\n" |
| removeEscape() | Removes escape characters (backslashes) from the String. |

|  |  |
| --- | --- |
| toBoolean() | Converts a String to a Boolean. The method takes two parameters: the String representation of the Boolean, and a boolean default value to use if a parse error occurs. |
| toDate() | Converts a String to a Date. |
| toDouble() | Converts a String to a double. The method takes two parameters: the String representation of the number, and a Double default value to use if a parse error occurs. |
| toDoubleList() | Converts a String to an array of Double objects based on the provided delimiter. An optional third parameter is the default value, returned if a parse error occurs. |
| toEnum() | Converts a String to an enum. If the String cannot be converted, an error message is written to the console. The first three of its six parameters are the String to be converted, the enum type specified as a String, and a PARAM name for the enum (used in an error message). The next two are two-dimensional arrays that link enum types and their corresponding values. The last parameter is the value that should be returned it the Strings cannot be converted.  The method has other signatures as well. See the API for details. |
| toEnumList() | Converts a String to a list of enums. If the String cannot be converted, an error message is written to the console. |
| toInt() | Converts a String to an integer. The method takes two parameters: the String representation of the integer, and an integer default value to use if a parse error occurs. |
| toIntegerList() | Converts a String to an array of Integers based on the provided delimiter. An optional third parameter is the default value, returned if a parse error occurs. |
| toIntList() | Converts a String to an array of Integer objects based on the provided delimiter. An optional third parameter is the default value, returned if a parse error occurs. |
| toNewLine() | Converts all occurrences of “\n” to newlines. |
| toString() | Converts an object to a String. If a String, newlines are replaced by “\n”. If a Vector, it is converted to a comma-separated list. |
| toStringList() | Converts a String to an array of Strings. There are three signatures: (a) a comma-separated String, (b) a String and a delimiter, and (c) the String, delimiter, and a Boolean indicating whether the String should be trimmed. Strings are trimmed by default. |

|  |  |
| --- | --- |
| toVector() | Converts a delimited list of tokens to a Vector. The second parameter is the delimiter that separates the tokens in the String. An optional third parameter is the default value, returned if a parse error occurs. |
| trim() | Returns a trimmed String, with trailing nulls removed. |

###### JCSwingTypeConverter

The methods in JCSwingTypeConverter are:

|  |  |
| --- | --- |
| toInsets() | Converts a String to an Insets instance. |
| fromInsets() | Creates a String from an AWT Insets value. |
| toDimension() | Converts a String in the form “40x30” to a Dimension instance. |
| toPoint() | Converts a String to a Point instance. |
| toColorList() | Converts a String to an array of Colors. An optional second parameter allows you to specify a default Color list if an error occurs while parsing the String. |
| toColor() | Converts a String to a Color. An optional second parameter allows you to specify a default Color if an error occurs while parsing the String. |
| fromColorList() | Converts list to a comma-separated list of tokens, to one of the  Color enums, or to RGB format. |
| toFont() | Converts a font name to a font instance, or a font name in format “name-style-size”, for instance, “Helvetica-plain-10.” |
| fromFont() | Returns font in format “name-style-size.” |

#### Examples

The following example gives you an indication of how the static methods in

JCSwingTypeConverter can be used.

import java.awt.Dimension; import java.awt.Font; import java.awt.Color;

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

class SwingTypeConverterExamples {

public static void main(String[] args){ System.out.println("++++++++++++++++++++++++++++++++++++++++++++++++");

String s;

Font f = new Font("System", 10, 10); s = JCSwingTypeConverter.fromFont(f);

System.out.println("The name of the font is " + s);

System.out.println("++++++++++++++++++++++++++++++++++++++++++++++++");

String colors = "red, blue, green";

Color[] colorarray = JCSwingTypeConverter.toColorList(colors,

null);

for (int i=0; i<colorarray.length; i++) System.out.println("The array of colors is: " +

colorarray[i].toString( System.out.println("++++++++++++++++++++++++++++++++++++++++++++++++");

Color[] mycolors = JCSwingTypeConverter.toColorList(new String

("black, blue, cyan"));

for (int i=0; i<mycolors.length; i++) System.out.println("The Color array is: " +

mycolors[i].toString());

System.out.println("++++++++++++++++++++++++++++++++++++++++++++++++");

Color yourcolor = JCSwingTypeConverter.toColor("darkGray",

Color.gray); System.out.println("The color is: " + yourcolor.toString());

System.out.println("++++++++++++++++++++++++++++++++++++++++++++++++");

Dimension dim = JCSwingTypeConverter.toDimension("40x30", null); System.out.println("The dimension is: " + dim.toString());

}

}

The output of this program is:

+++++++++++++++++++++++++++++++++++++++++++++++++++

The name of the font is System-PLAIN-10

+++++++++++++++++++++++++++++++++++++++++++++++++++

The array of colors is: java.awt.Color[r=255,g=0,b=0] The array of colors is: java.awt.Color[r=0,g=0,b=255] The array of colors is: java.awt.Color[r=0,g=255,b=0]

+++++++++++++++++++++++++++++++++++++++++++++++++++

The Color array is: java.awt.Color[r=0,g=0,b=0] The Color array is: java.awt.Color[r=0,g=0,b=255]

The Color array is: java.awt.Color[r=0,g=255,b=255]

+++++++++++++++++++++++++++++++++++++++++++++++++++

The color is: java.awt.Color[r=64,g=64,b=64]

+++++++++++++++++++++++++++++++++++++++++++++++++++

The dimension is: java.awt.Dimension[width=40,height=30]

The static methods of JCTypeConverter are called in a similar fashion, as illustrated next.

import java.util.Date; import java.text.DateFormat;

import com.klg.jclass.util.JCTypeConverter; class TypeConverterExamples {

public static void main(String[] args){ System.out.println("+++++++++++++++++++++++++++++++++++++++++");

String s = "10.777"; double dd = 10;

double d = JCTypeConverter.toDouble(s, dd); System.out.println("The value of the double is: " + d);

System.out.println("+++++++++++++++++++++++++++++++++++++++++");

s = "Abel, Ben, Curry, Dave"; String[] sa = JCTypeConverter.toStringList(s, ',', true); for (int i=0; i<sa.length; i++)

System.out.println("The array element is: " + sa[i]); System.out.println("+++++++++++++++++++++++++++++++++++++++++");

s = "1, 1, 2, 3, 5, 8, 13";

int [] da = {1,1,1,1,1,1,1};

int[] ii = JCTypeConverter.toIntList(s, ',', da); for (int i=0; i<ii.length; i++)

System.out.println("The Integer array element is: " +

ii[i]);

System.out.println("+++++++++++++++++++++++++++++++++++++++"); s = "Feb 30, 2000";

Date today = new Date("June 12, 1999");

Date myDate = JCTypeConverter.toDate(s, today); System.out.println("The date is: " + myDate.toString());

}

}

Here is the output:

DOS: %JAVA\_HOME%\bin\java TypeConverterExamples

+++++++++++++++++++++++++++++++++++++++++

The value of the double is: 10.777

+++++++++++++++++++++++++++++++++++++++++

The array element is: Abel The array element is: Ben The array element is: Curry The array element is: Dave

+++++++++++++++++++++++++++++++++++++++++

The Integer array element is: 1 The Integer array element is: 1 The Integer array element is: 2

The Integer array element is: 3 The Integer array element is: 5 The Integer array element is: 8 The Integer array element is: 13

+++++++++++++++++++++++++++++++++++++++++

The date is: Sat Jun 12 00:00:00 EDT 1999

# 27

## Word Wrap

[*Features of JCWordWrap*](#_bookmark761)[*Methods*](#_bookmark763)[*Examples*](#_bookmark767)

#### Features of JCWordWrap

JCWordWrap provides a static method called wrapText() that performs basic word-wrap logic on a String, given a line width in pixels and a delimiter to insert just before the line width is reached. While the delimiter is most often a newline, it can be any String.

The resulting String produced by JCWordWrap has lines no longer than the line width supplied as one of the parameters. Since the width of a line is measured in pixels, the number of words in a line depends on the FontMetrics currently in effect.

The other static method in this class is replace(). Its parameters are three Strings. The first parameter is the text String to be searched for occurrences of the second parameter. Any such occurrences are replaced with the third parameter.

#### Methods

Following is a list of the JCWordWrap methods:

|  |  |
| --- | --- |
| wrapText() | This static method returns a word-wrapped String, given an input String, a FontMetrics object, a line width in pixels, a delimiter such as a newline, and a Boolean to indicate whether left-alignment is in effect. Word-wrap logic breaks lines by spaces, but provides no hyphenation logic. The original String is returned if the number of characters is less than 10. |
| replace() | Returns a String stripped of a delimiter, or replaces one String with another. |

#### Examples

The code fragment shown here takes one rather long text String and constructs a reformatted one by adding newlines every so often. The new String s inserts newlines so that the lines never exceed 100 pixels, based on the current font.

...

FontMetrics fm = g.getFontMetrics(f);

String text = "It has flown away"; text += "The nightingale that called "; text += "Waking me at midnight ";

text += "Yet its song seems " text += "Still by my pillow.";

s = JCWordWrap.wrapText(text, fm, 100, delimiter, true);

*Figure 33 It has flown away*

*The nightingale that called Waking me at midnight Yet its song seems Still by my pillow.*

If the length for word wrapping is decreased to 50 pixels,

s = JCWordWrap.wrapText(text, fm, 50, delimiter, true);

The output String s is formatted as shown:

It has flown away The

nightingale that called Waking

me at midnight Yet its song seems

Still by my pillow.

Taking this second case, use replace() to put the word -STOP- in place of a newline:

String s1 = JCWordWrap.replace(s, "\n", "-STOP-");

This would yield:

It has -STOP-flown -STOP-away The -STOP-nightengale -STOP-that called - STOP-Waking -STOP-me at -STOP-midnight -STOP-Yet its -STOP-song -STOP-seems

-STOP-Still by my -STOP-pillow.

*Part III*

*Reference* *Appendices*

***A***

## Bean Properties Reference

[*Beans in the Swing Package*](#_bookmark777)[*Beans in the com.klg.jclass.util.swing Package*](#_bookmark791)

The following is a listing of the JClass Elements Bean properties and their default values. The properties are arranged alphabetically by property name. The second entry on any given row names the data type returned by the method. Note that a small number of properties are really read-only variables, and therefore only have a *get* method. These properties are marked with a “(G)” following the property name.

#### Beans in the Swing Package

##### Properties of JCMultiSelectList

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about (G) | String | com.klg.jclass.swing x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | (null) |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | false |
| enabled | Boolean | true |
| fixedCellHeight | int | -1 |
| fixedCellWidth | int | -1 |
| font | Font | (null) |
| foreground | Color | (null) |
| maximumSize | Dimension | [32767, 32767] |
| minimumSize | Dimension | [153, 42] |
| model | javax.swing.ListModel | SetValue |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | false |
| preferredSize | Dimension | [232, 200] |
| prototypeCellValue | java.lang.Object | (null) |
| requestFocusEnabled | Boolean | true |
| selectionBackground | Color | 204,204,255 |
| selectionForeground | Color | 0,0,0 |
| toolTipText | String | (null) |

* + 1. **Properties of JCMD****IFrame**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about (G) | String | com.klg.jclass.swing x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | (null) |
| border | javax.swing.border. Border | javax.swing.plaf.metal. MetalBorders  $InternalFrameBorder |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | false |
| enabled | Boolean | true |
| font | Font | (null) |
| foreground | Color | (null) |
| maximumSize | Dimension | [MAX\_VALUE, MAX\_VALUE] |
| minimumSize | Dimension | [120, 24] |
| name | String | (null) |
| nextFocusableCompone nt | Component | (null) |
| opaque | Boolean | false |
| preferredSize | Dimension | [10, 34] |
| requestFocusEnabled | Boolean | true |
| toolTipText | String | (null) |

* + 1. **Properties of JCMD****IPane**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about (G) | String | com.klg.jclass.swing x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | 153,153,204 |
| border | javax.swing.border. Border | (null) |
| considerIconsWhen Tiling | boolean | false |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | false |
| dragMode | int | LIVE\_DRAG\_MODE  (corresponds to DEFAULT for frameManuipulationStyle) |
| enabled | Boolean | true |
| font | Font | (null) |
| foreground | Color | (null) |
| frameManipulationStyle | int | DEFAULT |
| maximumSize | Dimension | [100, 100] |
| minimumSize | Dimension | [300, 200] |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | true |
| preferredSize | Dimension | [300, 200] |
| requestFocusEnabled | Boolean | true |
| toolTipText | String | (null) |

* + 1. **Properties of JCTre****eExplorer**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about (G) | String | com.klg.jclass.swing x.x.x |
| alignmentX | float | 0.0 |
| alignmentY | float | 0.0 |
| background | Color | (null) |
| border | javax.swing.border. Border | javax.swing.plaf.basic. BasicBorders$SplitPane Border |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | false |
| enabled | Boolean | true |
| font | Font | (null) |
| foreground | Color | (null) |
| maximumSize | Dimension | [MAX\_VALUE, MAX\_VALUE] |
| minimumSize | Dimension | [46, 24] |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | false |
| preferredSize | Dimension | [515, 405] |
| requestFocusEnabled | Boolean | true |
| toolTipText | String | (null) |

* + 1. **Properties of JCTre****eTable**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about (G) | String | com.klg.jclass.swing x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| autoSort | Boolean | true |
| background | Color | 255,255,255 |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | false |
| enabled | Boolean | true |
| font | Font | null |
| foreground | Color | 0,0,0 |
| maximumSize | Dimension | [0, 20] |
| minimumSize | Dimension | [0, 20] |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | true |
| preferredSize | Dimension | [0, 20] |
| requestFocusEnabled | Boolean | true |
| rootVisible | Boolean | true |
| scrollsOnExpand | Boolean | true |
| showNodeLines | int | Use Plaf |
| showsRootHandles | Boolean | false |
| sortable | Boolean | true |
| toolTipText | String | (null) |
| view | int | Tree |

* + 1. **Properties of** **JCWizard**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about (G) | String | com.klg.jclass.swing x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | 204,204,204 |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | true |
| enabled | Boolean | true |
| font | Font | null |
| foreground | Color | 0,0,0 |
| maximumSize | Dimension | [MAX\_VALUE, MAX\_VALUE] |
| minimumSize | Dimension | [0, 0] |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | true |
| preferredSize | Dimension | [0, 0] |
| requestFocusEnabled | Boolean | true |
| toolTipText | String | (null) |

* + 1. **Properties of JCWi****zardPage**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about (G) | String | com.klg.jclass.swing x.x.x |
| alignmentX | float | 0.0 |
| alignmentY | float | 0.0 |
| background | Color | 204,204,204 |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | true |
| enabled | Boolean | true |
| font | Font | null |
| foreground | Color | 0,0,0 |
| maximumSize | Dimension | [33174, 131068] |
| minimumSize | Dimension | [407, 37] |
| name | String | VerticalBox0 |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | true |
| preferredSize | Dimension | [407, 39] |
| requestFocusEnabled | Boolean | true |
| toolTipText | String | (null) |

#### Beans in the com.klg.jclass.util.swing Package

* + 1. **Properties of** **JCBox**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignment | int | Top |
| alignmentX | float | 0.0 |
| alignmentY | float | 0.0 |
| background | Color | 204,204,204 |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | True |
| enabled | Boolean | True |
| font | java.awt.Font | null |
| foreground | Color | 0,0,0 |
| maximumSize | Dimension | [0, 0] |
| minimumSize | Dimension | [0, 0] |
| name | String | HorizontalBox0 |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | True |
| orientation | int | Horizontal |
| preferredSize | Dimension | [0,0] |
| requestFocusEnabled | Boolean | True |
| toolTipText | String | (null) |

* + 1. **Properties of** **JCBrace**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | (null) |
| doubleBuffered | Boolean | False |
| enabled | Boolean | True |
| font | java.awt.Font | (null) |
| foreground | Color | (null) |
| length | int | 10 |
| maximumSize | Dimension | [10, 32767] |
| minimumSize | Dimension | [10, 0] |
| name | String | (null) |
| opaque | Boolean | True |
| orientation | int | Horizontal |
| preferredSize | Dimension | [10, 0] |

* + 1. **Properties of** **JCCheckBoxList**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | 255,255,255 |
| border | javax.swing.border.Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | False |
| enabled | Boolean | True |
| fixedCellHeight | int | -1 |
| fixedCellWidth | int | -1 |
| font | java.awt.Font | null |
| foreground | Color | 0,0,0 |
| maximumSize | Dimension | [0, 0] |
| minimumSize | Dimension | [0, 0] |
| model | javax.swing.ListModel | SetValue |
| null |  |  |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | True |
| preferredSize | Dimension | [0, 0] |
| prototypeCellValue | java.lang.Object | (null) |
| requestFocusEnabled | Boolean | True |
| selectionBackground | Color | 204,204,255 |
| selectionForeground | Color | 0,0,0 |
| selectionMode | int | Multiple Interval |
| toolTipText | String | (null) |
| visibleRowCount | int | 8 |

* + 1. **Properties of** **JCExitFrame**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | 204,204,204 |
| doubleBuffered | Boolean | False |
| enabled | Boolean | True |
| exitOnClose | Boolean | True |
| font | java.awt.Font | (null) |
| foreground | Color | (null) |
| maximumSize | Dimension | [MAX\_VALUE, MAX\_VALUE] |
| minimumSize | Dimension | [0, 0] |
| name | String | frame0 |
| opaque | Boolean | True |
| preferredSize | Dimension | [0, 0] |

* + 1. **Properties of** **JCFontChooserBar**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| UIClassID | String | (null) |
| accessibleContext | javax.accessibility. AccessibleContext | (null) |
| alignmentX | float | (null) |
| alignmentY | float | (null) |
| autoscrolls | Boolean | (null) |
| background | Color | (null) |
| border | javax.swing.border.Border | (null) |
| component | (null) | null |
| componentCount | int | (null) |
| components | Component[] | (null) |
| debugGraphicsOptions | int | (null) |
| doubleBuffered | Boolean | (null) |
| enabled | Boolean | (null) |
| focusCycleRoot | Boolean | (null) |
| focusTraversable | Boolean | (null) |
| font | java.awt.Font | (null) |
| foreground | Color | (null) |
| graphics | Graphics | (null) |
| height | int | (null) |
| insets | java.awt.Insets | (null) |
| layout | java.awt.LayoutManager | (null) |
| managingFocus | Boolean | (null) |
| maximumSize | Dimension | (null) |
| minimumSize | Dimension | (null) |
| name | String | (null) |
| nameList | String[] | (null) |

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | (null) |
| optimizedDrawingEnabled | Boolean | (null) |
| paintingTile | Boolean | (null) |
| preferredSize | Dimension | (null) |
| registeredKeyStrokes | javax.swing.KeyStroke[] | (null) |
| requestFocusEnabled | Boolean | (null) |
| rootPane | javax.swing.JRootPane | (null) |
| selectedFont | java.awt.Font | (null) |
| toolTipText | String | (null) |
| topLevelAncestor | java.awt.Container | (null) |
| underline | Boolean | (null) |
| validateRoot | Boolean | (null) |
| visible | Boolean | (null) |
| visibleRect | java.awt.Rectangle | (null) |
| width | int | (null) |
| x | int | (null) |
| y | int | (null) |

* + 1. **Properties of JCFo****ntChooserPane**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | (null) |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | False |
| enabled | Boolean | True |
| font | java.awt.Font | (null) |
| foreground | Color | (null) |
| maximumSize | Dimension | [32767, 32767] |
| minimumSize | Dimension | [274, 236] |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | False |
| preferredSize | Dimension | [274, 369] |
| requestFocusEnabled | Boolean | True |
| selectedFont | java.awt.Font | null |
| styleControls | int | null |
| toolTipEnabled | Boolean | True |
| toolTipText | String | (null) |

* + 1. **Properties of** **JCHelpPane**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.0 |
| alignmentY | float | 0.0 |
| background | Color | (null) |
| border | javax.swing.border. Border | javax.swing.plaf.basic. BasicBorders$SplitPane Border@16b328a8 |
| contentsPage | java.net.URL | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | False |
| enabled | Boolean | True |
| font | java.awt.Font | (null) |
| foreground | Color | (null) |
| maximumSize | Dimension | [MAX\_VALUE, MAX\_VALUE] |
| minimumSize | Dimension | [46, 24] |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | False |
| preferredSize | Dimension | (null) |
| requestFocusEnabled | Boolean | True |
| titlePage | java.net.URL | (null) |
| toolTipText | String | (null) |
| useToolBar | Boolean | True |
| viewPage | java.net.URL | (null) |

* + 1. **Properties of** **JCHTMLPane**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | 255,255,255 |
| border | javax.swing.border. Border | javax.swing.plaf.basic. BasicBorders  $MarginBorder instance |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | False |
| enabled | Boolean | True |
| font | java.awt.Font | null |
| foreground | Color | 0,0,0 |
| maximumSize | Dimension | [MAX\_VALUE, MAX\_VALUE] |
| minimumSize | Dimension | (null) |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | True |
| page | java.net.URL | (null) |
| preferredSize | Dimension | (null) |
| requestFocusEnabled | Boolean | True |
| toolTipText | String | (null) |

* + 1. **Properties of JCSo****rtableTable**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| autoSort | Boolean | False |
| background | Color | 255,255,255 |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | False |
| enabled | Boolean | True |
| font | java.awt.Font | null |
| foreground | Color | 0,0,0 |
| maximumSize | Dimension | [0, 0] |
| minimumSize | Dimension | [0, 0] |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | True |
| preferredSize | Dimension | [0, 0] |
| requestFocusEnabled | Boolean | True |
| toolTipText | String | (null) |

* + 1. **Properties of** **JCSpinBox**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| actionCommand | String | spinBoxChanged |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| arrowKeySpinningAllowed | Boolean | True |
| background | Color | (null) |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | False |
| editable | Boolean | True |
| enabled | Boolean | True |
| font | java.awt.Font | (null) |
| foreground | Color | (null) |
| maximumSize | Dimension | [32767, 32767] |
| minimumSize | Dimension | [0, 0] |
| model | com.klg.jclass.util. swing.JCSpinBoxModel | null |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | False |
| preferredSize | Dimension | [24, 21] |
| requestFocusEnabled | Boolean | True |
| selectedIndex | int | -1 |
| toolTipText | String | (null) |

* + 1. **Properties of JCSp****inNumberBox**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| arrowKeySpinningAllowed | Boolean | True |
| background | Color | (null) |
| border | javax.swing.border. Border | (null) |
| debugGraphicsOptions | int | 0 |
| doubleBuffered | Boolean | False |
| editable | Boolean | True |
| enabled | Boolean | True |
| font | java.awt.Font | (null) |
| foreground | Color | (null) |
| maximumSize | Dimension | [32767, 32767] |
| minimumSize | Dimension | [0, 0] |
| name | String | (null) |
| nextFocusableComponent | Component | (null) |
| opaque | Boolean | False |
| operation | int | Integer |
| preferredSize | Dimension | [173, 21] |
| requestFocusEnabled | Boolean | True |
| spinStep | java.lang.Number | 1.0 |
| toolTipText | String | (null) |
| value | java.lang.Number | (null) |

* + 1. **Properties of** **JCSpring**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | String | com.klg.jclass.util x.x.x |
| alignmentX | float | 0.5 |
| alignmentY | float | 0.5 |
| background | Color | (null) |
| doubleBuffered | Boolean | False |
| enabled | Boolean | True |
| font | java.awt.Font | (null) |
| foreground | Color | (null) |
| horizontalElasticity | int | 1 |
| maximumSize | Dimension | [32767, 32767] |
| minimumSize | Dimension | [0, 0] |
| name | String | BidirecionalSpring0 |
| opaque | Boolean | True |
| preferredSize | Dimension | [0, 0] |
| verticalElasticity | int | 1 |

* + 1. **Properties of** **JCDateChooser**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| about | java.lang.String | com.klg.jclass.util x.x.x Preview |
| alignmentX | float | 0.0 |
| alignmentY | float | 0.0 |
| background | java.awt.Color | 204,204,204 |
| border | javax.swing.border. Border | (null) |
| days | String[] | locale default array |
| debugGraphicsOptions | int | 0 |

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Default Value** |
| doubleBuffered | Boolean | True |
| enabled | Boolean | True |
| font | java.awt.Font | null |
| foreground | java.awt.Color | 0,0,0 |
| maximumDate | java.util.Calendar | (null) |
| maximumSize | java.awt.Dimension | java.awt.Dimension [width=203, height=231] |
| minimumDate | java.util.Calendar | (null) |
| minimumSize | java.awt.Dimension | java.awt.Dimension [width=90, height=54] |
| months | String[] | locale default array |
| name | java.lang.String | (null) |
| nextFocusableComponent | java.awt.Component | (null) |
| opaque | Boolean | True |
| preferredSize | java.awt.Dimension | java.awt.Dimension [width=203, height=231] |
| requestFocusEnabled | Boolean | True |
| shortMonths | String[] | locale default array |
| toolTipText | java.lang.String | (null) |
| value | java.util.Calendar | null |

***B***

**Distributing Applets and Applications**

The size of the archive and its related download time are important factors to consider when deploying your applet or application.

When you create an applet or an application using third-party classes such as JClass components, your deployment archive will contain many unused class files unless you customize your JAR. Optimally, the deployment JAR should contain only your classes and the third-party classes you actually use. For example, the *jchigrid.jar*, which you may have used to develop an applet or application, contains classes and packages that are only useful during the development process and that are not referenced by your application. These classes include the Property Editors and BeanInfo classes. JClass JarMaster helps you create a deployment JAR that contains only the class files required to run your application.

###### Using JarMaster to Customize the Deployment Archive

JClass JarMaster is a robust utility that allows you to customize and reduce the size of the deployment archive quickly and easily. Using JClass JarMaster you can select the classes you know must belong in your JAR, and JarMaster will automatically search for all of the direct and indirect dependencies (supporting classes).

When you optimize the size of the deployment JAR with JClass JarMaster, you save yourself the time and trouble of building a JAR manually and determining the necessity of each class or package. Your deployment JAR will take less time to load and will use less space on your server as a direct result of excluding all of the classes that are never used by your applet or application.

For more information about using JarMaster to create and edit JARs, please consult its online documentation.

JClass JarMaster is included with the JClass DesktopViews suite of products. For more details please refer to [*Quest Software’s Web site*](http://www.quest.com/).

# C

## Colors and Fonts

[*Colorname Values*](#_bookmark823)[*RGB Color Values*](#_bookmark825)[*Fonts*](#_bookmark828)

This section provides information on common colorname values, specific RGB color values and fonts applicable to all Java programs. You may find it useful as a guide for choosing colors for cells.

#### Colorname Values

The following lists all the colornames that can be used within Java programs. The majority of these colors will appear the same (or similar) across different computing platforms.

black lightGray

blue lightBlue

cyan magenta

darkGray orange

darkGrey pink

gray red

grey white

green yellow

lightGray

#### RGB Color Values

The following lists all the main RGB color values that can be used within

JClass Elements. RGB color values are specified as three numeric values representing the red, green and blue color components; these values are separated by dashes (“-”).

The following RGB color values describe the colors available to Unix systems. It is recommended that you test these color values in a JClass program on a Windows or Macintosh system before utilizing them.

The list begins with all of the variations of white, then blacks and grays, and then describes the full color spectrum ranging from reds to violets.

Example code from an HTML file:

<PARAM NAME=backgroundList VALUE="(4, 5 255-255-0)">

###### RGB Value Description

255-250-250 Snow

248-248-255 Ghost White

245-245-245 White Smoke

220-220-220 Gainsboro

255-250-240 Floral White

253-245-230 Old Lace

250-240-230 Linen

250-235-215 Antique White

255-239-213 Papaya Whip

255-235-205 Blanched Almond

255-228-196 Bisque

255-218-185 Peach Puff

255-222-173 Navajo White

255-228-181 Moccasin

255 248-220 Cornsilk

255-255-240 Ivory

255-250-205 Lemon Chiffon

255-245-238 Seashell

240-255-240 Honeydew

245-255-250 Mint Cream

240-255-255 Azure

240-248-255 Alice Blue

230-230-250 Lavender

255-240-245 Lavender Blush

255-228-225 Misty Rose

###### RGB Value Description

255-255-255 White

0-0-0 Black

47-79-79 Dark Slate Grey

105-105-105 Dim Gray

112- 128-144 Slate Grey

119- 136-153 Light Slate Grey

190- 190-190 Grey

211- 211-211 Light Gray

25-25-112 Midnight Blue

0-0-128 Navy Blue

100- 149 237 Cornflower Blue

72-61-139 Dark Slate Blue

106-90-205 Slate Blue

123- 104 238 Medium Slate Blue

132-112- 255 Light Slate Blue

0-0-205 Medium Blue

65-105-225 Royal Blue

0-0-255 Blue

30-144-255 Dodger Blue

0-19 -255 Deep Sky Blue

135-206-235 Sky Blue

135-206-250 Light Sky Blue

70-130-180 Steel Blue

176-196- 222 Light Steel Blue

173-216-230 Light Blue

176-224-230 Powder Blue

175-238-238 Pale Turquoise

0-206-209 Dark Turquoise

72-209-204 Medium Turquoise

64-224-208 Turquoise

0-255-255 Cyan

224-255-255 Light Cyan

###### RGB Value Description

95-158-160 Cadet Blue

102-205-170 Medium Aquamarine

127-255-212 Aquamarine

0-100-0 Dark Green

85-107-47 Dark Olive Green

143-188-143 Dark Sea Green

46-139-87 Sea Green

60-179-113 Medium Sea Green

32-178-170 Light Sea Green

152-251-152 Pale Green

0-255-127 Spring Green

124-252- 0 Lawn Green

0-255-0 Green

127-255- 0 Chartreuse

0-250-154 Medium Spring Green

173-255-47 Green Yellow

50-205-50 Lime Green

154-205-50 Yellow Green

34-139-34 Forest Green

107-142-35 Olive Drab

189-183-107 Dark Khaki

240-230-140 Khaki

238-232-170 Pale Goldenrod

250-250-210 Light Goldenrod Yellow

255-255-224 Light Yellow

255-255-0 Yellow

255-215-0 Gold

238-221-130 Light Goldenrod

218-165-32 Goldenrod

184-134-11 Dark Goldenrod

188-143-143 Rosy Brown

205-92-92 Indian Red

###### RGB Value Description

139-69-19 Saddle Brown

160-82-45 Sienna

205-133-63 Peru

222-184- 135 Burlywood

245-245-220 Beige

245-222-179 Wheat

244-164-96 SandyBrown

210-180-140 Tan

210-105-30 Chocolate

178-34-34 Firebrick

165-42-42 Brown

233-150-122 Dark Salmon

250-128-114 Salmon

255-160-122 Light Salmon

255-165- 0 Orange

255-140-0 Dark Orange

255-127-80 Coral

240-128-128 Light Coral

255-99-71 Tomato

255-69-0 Orange Red

255-0-0 Red

255-105-180 Hot Pink

255-20-147 Deep Pink

255-192-203 Pink

255-182-193 Light Pink

219-112-147 Pale Violet Red

176-48-96 Maroon

199-21-133 Medium Violet Red

208-32-144 Violet Red

255-0-255 Magenta

238-130-238 Violet

221-160-221 Plum

###### RGB Value Description

218-112-214 Orchid

186-85-211 Medium Orchid

153-50-204 Dark Orchid

148-0-211 Dark Violet

138-43-226 Blue Violet

160- 32-240 Purple

147-112-219 Medium Purple

216-191-216 Thistle

#### Fonts

There are nine different logical font names that can be specified in any Java 2 program. They are:

Courier Dialog DialogInput Helvetica Monospaced SansSerif Serif TimesRoman ZapfDingbats

Note: Font names are case-sensitive.

There are also four standard font style constants that can be used. The valid Java 2 font style constants are:

Font.BOLD

Font.BOLD | Font.ITALIC Font.ITALIC

Font.PLAIN

These values are strung together with dashes (“-”) when used with the VALUE attribute. You must also specify a point size by adding it to other font elements. To display a text using a 12-point italic Helvetica font, use the following:

Helvetica-italic-12

All three elements (font name, font style and point size) must be used to specify a particular font display; otherwise, the default font is used instead.

Note: Font display may vary from system to system. If a font does not exist on a system, the default font is displayed instead.

**A**

AbstractLabel [30](#_bookmark97)

AbstractSpinBox

super class of JCSpinBox and JCSpinNumberBox [84](#_bookmark326)

accept

JCFileFilter method [130](#_bookmark597)

actionCommand

property of JCSpinBox [85](#_bookmark330)

distributing [191](#_bookmark815)

JarMaster [191](#_bookmark815)

arrangeIcons

method in JCMDIPane [77](#_bookmark282)

### B

Beans

## Index

activateFrame

method in JCMDIPane [77](#_bookmark281)

add

method in JCWizard [108](#_bookmark475)

method of JCListenerList [141](#_bookmark642)

addActionListener

method in JCSpinBox [87](#_bookmark350)

addExtension

JCFileFilter method [130](#_bookmark596)

addItem

method in JCSpinBox [87](#_bookmark353)

addItemListener

method in JCSpinBox [87](#_bookmark356)

addListSelectionListener

method in JCMultiSelectList [81](#_bookmark303)

addSelectionInterval

method in JCMultiSelectList [81](#_bookmark304)

addSelectionPath JCTreeTable method [100](#_bookmark411)

addSelectionPaths JCTreeTable method [100](#_bookmark412)

addSpecialDate

property of JCCalendar [32](#_bookmark121) addTableHeaderMouseListener JCTreeTable method [100](#_bookmark413)

addTreeExpansionListener JCTreeTable method [100](#_bookmark414)

addTreeWillExpandListener JCTreeTable method [100](#_bookmark415)

addWizardListener

method in JCWizard [108](#_bookmark476)

alignment

JCBox property [113](#_bookmark520)

API [12](#_bookmark8)

applets

distributing [191](#_bookmark815)

JarMaster [191](#_bookmark815)

applications

com.klg.jclass.util.swing package [177](#_bookmark792)

JCBox [177](#_bookmark793)

JCBrace [178](#_bookmark794)

JCCheckBoxList [179](#_bookmark796)

JCDateChooser [189](#_bookmark812)

JCExitFrame [180](#_bookmark798)

JCFontChooserBar [181](#_bookmark800)

JCFontChooserPane [183](#_bookmark802)

JCHelpPane [184](#_bookmark803)

JCHTMLPane [185](#_bookmark805)

JCSortableTable [186](#_bookmark806)

JCSpinBox [187](#_bookmark808)

JCSpinNumberBox [188](#_bookmark810)

JCSpring [189](#_bookmark811)

property reference [169](#_bookmark774)

Swing package [170](#_bookmark778)

JCMDIFrame [171](#_bookmark781)

JCMDIPane [172](#_bookmark784)

JCMultiSelectList [170](#_bookmark780)

JCTreeExplorer [173](#_bookmark786)

JCTreeTable [174](#_bookmark788)

JCWizard [175](#_bookmark789)

JCWizardPage [176](#_bookmark790)

borders

with JCBorder [112](#_bookmark512)

box component [112](#_bookmark513)

### C

CalendarComponent interface [31](#_bookmark108)

cancel

method in JCWizard [108](#_bookmark477)

canceled

method in JCWizardListener [109](#_bookmark498)

cascadeWindows

method in JCMDPane [77](#_bookmark283)

##### 201

checkbox-list component [25](#_bookmark71)

checkEnum

method in JCTypeConverter [158](#_bookmark728)

chooserType

property of JCDateChooser [32](#_bookmark112)

classes

JCBorder, associated component [112](#_bookmark512) JCBox, associated component [112](#_bookmark513) JCBrace, associated component [113](#_bookmark514) JCCheckBoxList [26](#_bookmark75)

JCEncodeComponent [137](#_bookmark625)

JCFontChooser [48](#_bookmark164)

JCHelpPane [52](#_bookmark184)

JCHTMLPane [52](#_bookmark184)

JCListenerList [141](#_bookmark639)

JCMappingSort [60](#_bookmark211)

JCProgressHelper [145](#_bookmark655)

JCSortableTable [61](#_bookmark215)

JCSpinBox, helper [84](#_bookmark325)

JCSpinNumberBox, helper [84](#_bookmark325)

JCSplashScreen [89](#_bookmark367)

JCStringTokenizer [149](#_bookmark685)

JCTreeExplorer [97](#_bookmark380)

JCTreeTable [97](#_bookmark380)

JCWizard [36](#_bookmark136), [107](#_bookmark467)

utility [121](#_bookmark548)

clear

method in JCIconCreator [134](#_bookmark613)

clearSelection

method in JCMultiSelectList [81](#_bookmark305)

closeFrame

method in JCMDIPane [77](#_bookmark284)

collapsePath

JCTreeTable method [100](#_bookmark416)

collapseRow

JCTreeTable method [100](#_bookmark417)

CollectionIntComparator [60](#_bookmark212)

colors

colorname values [193](#_bookmark824) RGB color value list [193](#_bookmark827) RGB values [193](#_bookmark826)

com.klg.jclass.util.calendar [30](#_bookmark96)

com.klg.jclass.util.treetableBranchTree [98](#_bookmark385)

com.klg.jclass.util.treetableNodeChildrenTable [98](#_bookmark389)

completeProgress

method in JCProgressHelper [146](#_bookmark666)

components checkbox-list [25](#_bookmark71)

description of SwingSuite’s [17](#_bookmark25)

list [18](#_bookmark26)

table [93](#_bookmark375)

tree [93](#_bookmark375)

constructors [38](#_bookmark139), [107](#_bookmark473)

JCEncodeComponent [138](#_bookmark627)

JCExitFrame [44](#_bookmark154)

JCFileFilter [129](#_bookmark593)

JCHelpPane [53](#_bookmark189)

JCHTMLPane [53](#_bookmark188)

JCIconCreator [133](#_bookmark612)

JCMulitSelectList [81](#_bookmark301)

JCProgressHelper [144](#_bookmark654)

JCSortableTable [61](#_bookmark221)

JCSpinBox [86](#_bookmark348)

JCSpinNumberBox [86](#_bookmark348)

JCSplashScreen [90](#_bookmark369)

JCTreeSet [156](#_bookmark717)

continuousScroll

property of JCSpinBox and JCSpinNumberBox [85](#_bookmark331)

countTokens

method in JCStringTokenizer [150](#_bookmark688)

createDefaultColumnsFromModel method in JCSortableTable [62](#_bookmark224)

createSortableTableColumn JCTreeTable method [100](#_bookmark418)

### D

days

property of JCDateChooser [32](#_bookmark113)

DayTable [30](#_bookmark98)

deactivateFrame

method in JCMDIPane [77](#_bookmark285)

debugging [125](#_bookmark565)

printing debug information [126](#_bookmark574)

stack trace [127](#_bookmark584)

DefaultTreeIconRenderer [98](#_bookmark386)

DefaultTreeTableSelectionModel [97](#_bookmark381)

deselectAll

method in JCMultiSelectList [81](#_bookmark306)

deselectItem

method in JCMultiSelectList [81](#_bookmark307) distributing applets and applications [191](#_bookmark815) dual spin [29](#_bookmark91)

### E

elements

method of JCListenerList [141](#_bookmark644)

error

method in JCWordWrap [158](#_bookmark729)

events

JCProgressHelper [145](#_bookmark663)

JCWizard [109](#_bookmark490)

spin box [88](#_bookmark360)

examples JCAlignLayout [117](#_bookmark542)

JCCheckBoxList [26](#_bookmark81)

JCGridLayout [116](#_bookmark541)

JCListenerList [142](#_bookmark646)

JCMappingSort [64](#_bookmark240)

JCSortableTable [65](#_bookmark241)

exit frame [43](#_bookmark143)

expandPath

JCTreeTable method [100](#_bookmark419)

expandRow

JCTreeTable method [100](#_bookmark420)

### F

feature overview [9](#_bookmark2)

finish

method in JCWizard [108](#_bookmark478)

finished

method in JCWizardListener [109](#_bookmark497)

fireSelectionValueChanged method in JCMultiSelectList [81](#_bookmark308)

first

method in JCWizard [108](#_bookmark479)

fontChanging

method in JCFontListener [49](#_bookmark175)

fonts

chooser [47](#_bookmark157)

names [198](#_bookmark829)

point size [198](#_bookmark832)

style constants [198](#_bookmark831)

frame

exit [43](#_bookmark143)

multiple document [71](#_bookmark244)

fromColorList

method in JCSwingTypeConverter [160](#_bookmark754)

fromEnum

method in JCTypeConverter [158](#_bookmark730)

fromFont

method in JCSwingTypeConverter [160](#_bookmark756)

fromInsets

method in JCSwingTypeConverter [160](#_bookmark748)

fromNewLine

method in JCTypeConverter [158](#_bookmark731)

### G

getAllIconifiedFrames method in JCMDIPane [75](#_bookmark261)

getAllNonIconifiedFrames method in JCMDIPane [75](#_bookmark262)

getAnchorSelectionIndex

method in JCMultiSelectList [81](#_bookmark309)

getAutoSort

method in JCSortableTable [62](#_bookmark225)

getCellEditor

JCTreeTable method [100](#_bookmark421)

method in JCSortableTable [62](#_bookmark226)

getCellRenderer JCTreeTable method [100](#_bookmark422)

method in JCSortableTable [62](#_bookmark227)

getClosestPathForLocation JCTreeTable method [100](#_bookmark423)

getContentPane

method in JCMDIFrame [75](#_bookmark259)

getContentsPage

method in JCHelpPane [54](#_bookmark192) getContentsPane [54](#_bookmark193) getDayComponent

method in JCDateChooser [33](#_bookmark126)

getDescription

JCFileFilter method [130](#_bookmark599)

getDragMode

method in JCMDIPane [75](#_bookmark263)

getEditingPath

JCTreeTable method [100](#_bookmark424)

getEncoder

method in JCEncodeComponent [138](#_bookmark628)

getEscapeChar

method in JCStringTokenizer [150](#_bookmark689)

getExitOnClose

method in JCExitFrame [44](#_bookmark152)

getExpandedDescendants JCTreeTable method [100](#_bookmark425)

getExtension

JCFileFilter method [130](#_bookmark598)

getFailureMessage

method in JCEncodeComponent [138](#_bookmark629)

getFrameManipulationStyle method in JCMDIPane [75](#_bookmark265)

getIcon

method in JCIconCreator [134](#_bookmark614)

getItemAtl

property of JCSpinBox [85](#_bookmark332)

getItemCountl

property of JCSpinBox [85](#_bookmark333)

getKeyColumns

method in JCSortableTable [62](#_bookmark228)

getLabelVerticalAlignment JCAlignLayout method [114](#_bookmark530)

getLevel

method in JCDebug [127](#_bookmark581)

getLongName

method in JCEncodeComponent [138](#_bookmark630)

getMaximumValue

property of JCSpinNumberBox [86](#_bookmark340)

getMDIMenuBar

method in JCMDIFrame [75](#_bookmark256)

method in JCMDIPane [76](#_bookmark270)

getMDIToolBar

method in JCMDIFrame [75](#_bookmark257)

method in JCMDIPane [76](#_bookmark272)

getMonthComponent

method in JCDateChooser [33](#_bookmark127)

getNonSelectedIcon

method in JCMDIPane [76](#_bookmark274)

getNumberFormat

property of JCSpinNumberBox [86](#_bookmark342)

getPathForLocation JCTreeTable method [101](#_bookmark426)

getPathForRow JCTreeTable method [101](#_bookmark427)

getPosition

method in JCStringTokenizer [150](#_bookmark690)

getPreferredSize

method in JCMDIPane [76](#_bookmark278)

getPrintStream

method in JCDebug [126](#_bookmark575)

getRowForLocation JCTreeTable method [101](#_bookmark428)

getRowForPath JCTreeTable method [101](#_bookmark429)

getRowsForPaths JCTreeTable method [101](#_bookmark430)

getScrollsOnExpand JCTreeTable method [101](#_bookmark431)

getSelectedIcon

method in JCMDIPane [76](#_bookmark276)

getSelectedIndex

method in JCMultiSelectList [81](#_bookmark310)

getSelectedIndices

method in JCMultiSelectList [81](#_bookmark311)

getSelectedPath JCTreeTable method [101](#_bookmark432)

getSelectedValues

method in JCMultiSelectList [81](#_bookmark312)

getSelectionPath JCTreeExplorer method [99](#_bookmark403)

JCTreeTable method [101](#_bookmark433)

getSelectionPaths JCTreeExplorer method [99](#_bookmark404)

JCTreeTable method [101](#_bookmark434)

getShortName

method in JCEncodeComponent [138](#_bookmark631)

getShowNodeLines JCTreeTable method [101](#_bookmark435)

getShowsRootHandles JCTreeTable method [101](#_bookmark436)

getTable

JCTreeExplorer method [99](#_bookmark405)

property of JCTreeExplorer [99](#_bookmark393)

getTitlePage

method in JCHelpPane [54](#_bookmark195)

getTopFrame

method in JCMDIPane [76](#_bookmark279)

getTree

JCTreeExplorer method [99](#_bookmark406)

property of JCTreeExplorer [99](#_bookmark394)

getTreeIconRenderer JCTreeExplorer method [100](#_bookmark407)

JCTreeTable method [101](#_bookmark437)

getTreeSelectionModel

JCTreeTable method [101](#_bookmark438)

getTreeTableModel JCTreeTable method [101](#_bookmark439)

getUnsortedRow

method in JCSortableTable [63](#_bookmark229)

getValue

property of JCSpinNumberBox [86](#_bookmark345)

getView

JCTreeTable method [101](#_bookmark440)

getViewPage

method in JCHelpPane [54](#_bookmark196)

getViewPane

method in JCHelpPane [54](#_bookmark198)

getYearComponent

method in JCDateChooser [33](#_bookmark128) Graphical User Interface [17](#_bookmark23) GridLayout

JClass Elements equivalent [21](#_bookmark58)

### H

hasMoreTokens

method in JCStringTokenizer [150](#_bookmark691)

help

method in JCWizard [108](#_bookmark480), [109](#_bookmark485) method in JCWizardListener [109](#_bookmark499) panes [51](#_bookmark178)

horizontalElasticity JCSpring property [114](#_bookmark526)

HTML panes [51](#_bookmark178)

### I

icons

creating [133](#_bookmark604)

image

encoder [137](#_bookmark620)

interfaces JCEncodeComponent [137](#_bookmark625)

JCMappingSort [60](#_bookmark211)

JCSortableTable [61](#_bookmark215)

JCSpinBox [84](#_bookmark321)

JCSpinNumberBox [84](#_bookmark321)

JCSplashScreen [89](#_bookmark367)

JCTreeExplorer [96](#_bookmark379)

JCTreeTable [96](#_bookmark379)

isEditable

property of JCSpinBox [85](#_bookmark338)

isEnabled

method in JCDebug [126](#_bookmark579) isExtensionListInDescription JCFileFilter method [130](#_bookmark601)

isMaximized

method in JCMDIPane [75](#_bookmark268)

isOkayToContinue

method in JCProgressHelper [146](#_bookmark667)

isPathSelected

JCTreeTable method [101](#_bookmark441)

isRootVisible

JCTreeTable method [101](#_bookmark442)

isSortable

JCTreeTable method [101](#_bookmark443)

isSpecialDate

JCDateChooser method [32](#_bookmark122)

### J

JAR [191](#_bookmark818)

JarMaster [191](#_bookmark817)

JCAlignLayout [111](#_bookmark507), [114](#_bookmark529)

description [21](#_bookmark59)

JCBorder [112](#_bookmark512)

JCBox [112](#_bookmark513)

associated components [115](#_bookmark537)

properties [113](#_bookmark519), [177](#_bookmark793)

JCBrace [113](#_bookmark514)

associated components [116](#_bookmark538)

properties [113](#_bookmark522), [178](#_bookmark794)

JCCalendar [31](#_bookmark100)

methods [33](#_bookmark129)

properties [32](#_bookmark120)

JCCalendarContainer [31](#_bookmark99)

JCCheckBoxList [25](#_bookmark71)

classes [26](#_bookmark75)

description [18](#_bookmark27)

examples [26](#_bookmark81)

methods [26](#_bookmark79)

properties [26](#_bookmark77), [179](#_bookmark796)

JCColumnLayout [111](#_bookmark508), [114](#_bookmark534)

JCDateChooser [29](#_bookmark88), [31](#_bookmark101)

description [18](#_bookmark28), [22](#_bookmark66)

dual spin [29](#_bookmark91)

methods [33](#_bookmark125)

properties [32](#_bookmark111), [189](#_bookmark812)

quick select [29](#_bookmark92)

read only [30](#_bookmark93)

spin popdown [29](#_bookmark89)

Tool Tip [30](#_bookmark94)

types of [29](#_bookmark90)

JCDebug [121](#_bookmark551), [125](#_bookmark565)

methods [126](#_bookmark573)

setLevel [125](#_bookmark568)

JCElasticLayout [111](#_bookmark509), [113](#_bookmark517), [114](#_bookmark535)

description [21](#_bookmark62)

JCEncodeComponent [122](#_bookmark553), [137](#_bookmark623)

classes [137](#_bookmark625)

constructors [138](#_bookmark627)

interfaces [137](#_bookmark625)

methods [138](#_bookmark627)

JCExitFrame [43](#_bookmark146)

constructors [44](#_bookmark154)

description [18](#_bookmark29), [21](#_bookmark51)

methods [44](#_bookmark151)

properties [43](#_bookmark148), [180](#_bookmark798)

property of exitOnClose [43](#_bookmark149)

JCFileFilter [129](#_bookmark589)

constructors [129](#_bookmark593)

methods [130](#_bookmark595)

JCFontAdapter JCFontChooser class [48](#_bookmark168)

JCFontChooser [47](#_bookmark157)

classes [48](#_bookmark164)

description [21](#_bookmark57)

methods [49](#_bookmark174)

properties [49](#_bookmark170)

JCFontChooserBar [47](#_bookmark157)

description [21](#_bookmark60)

JCFontChooser class [48](#_bookmark165)

properties [181](#_bookmark800)

Tool Tip [47](#_bookmark161)

JCFontChooserPane [47](#_bookmark160)

description [18](#_bookmark30), [21](#_bookmark61)

JCFontChooser class [48](#_bookmark166)

properties [183](#_bookmark802)

JCFontEvent JCFontChooser class [48](#_bookmark163)

JCFontListener JCFontChooser class [48](#_bookmark167)

JCGridLayout [112](#_bookmark510), [115](#_bookmark536)

JCHelpPane [51](#_bookmark182), [54](#_bookmark191)

classes [52](#_bookmark184)

constructors [53](#_bookmark189)

description [18](#_bookmark32)

properties [52](#_bookmark186), [184](#_bookmark803)

JCHTMLPane [51](#_bookmark178)

classes [52](#_bookmark184)

constructors [53](#_bookmark188)

description [18](#_bookmark31), [21](#_bookmark46)

methods [53](#_bookmark190)

properties [52](#_bookmark186), [185](#_bookmark805)

JCIconCreator [121](#_bookmark552), [133](#_bookmark607)

advantages [133](#_bookmark608)

constructors [133](#_bookmark612)

methods [133](#_bookmark610)

JCIntComparator [60](#_bookmark213)

JClass JarMaster [191](#_bookmark819)

JCListenerList [122](#_bookmark554), [141](#_bookmark635)

classes [141](#_bookmark639)

methods [141](#_bookmark641)

JCMappingSort [59](#_bookmark206), [60](#_bookmark214), [122](#_bookmark555)

classes [60](#_bookmark211)

interfaces [60](#_bookmark211)

JCMDIFrame [71](#_bookmark247), [73](#_bookmark251)

description [18](#_bookmark33), [21](#_bookmark49)

methods [75](#_bookmark255)

properties [74](#_bookmark253), [171](#_bookmark781)

JCMDIPane [71](#_bookmark248), [72](#_bookmark250)

description [18](#_bookmark34), [21](#_bookmark47)

methods [75](#_bookmark260)

properties [74](#_bookmark253), [172](#_bookmark784)

JCMessageHelper [153](#_bookmark701)

methods [153](#_bookmark705)

JCMultiSelectList [79](#_bookmark292)

constructors [81](#_bookmark301)

description [19](#_bookmark35)

methods [81](#_bookmark302)

properties [80](#_bookmark296), [170](#_bookmark780)

JComboBox

JClass Elements equivalent [21](#_bookmark63)

JCProgressAbortedException

class associated with JCProgressHelper [145](#_bookmark661) JCProgressAdapter [145](#_bookmark662) JCProgressCancelledEvent

class associated with JCProgressHelper [145](#_bookmark658)

JCProgressCancelledListener

class associated with JCProgressHelper [145](#_bookmark659)

JCProgressEvent

class associated with JCProgressHelper [145](#_bookmark656)

JCProgressHelper [122](#_bookmark556), [143](#_bookmark652), [145](#_bookmark660)

classes [145](#_bookmark655)

constructors [144](#_bookmark654)

description [22](#_bookmark68)

events [145](#_bookmark663)

listeners [145](#_bookmark663)

methods [146](#_bookmark665)

JCProgressListener [145](#_bookmark657)

JCRowComparator [61](#_bookmark217)

used with JCSortableTable [65](#_bookmark242)

JCRowLayout [112](#_bookmark511)

JCRowSortModel [61](#_bookmark218)

JCSortableTable [59](#_bookmark209), [61](#_bookmark216), [97](#_bookmark383)

cell renderers [63](#_bookmark238)

classes [61](#_bookmark215)

constructors [61](#_bookmark221)

description [19](#_bookmark38)

interface [61](#_bookmark215)

methods [62](#_bookmark223)

properties [186](#_bookmark806)

using your own comparator with [61](#_bookmark219)

JCSpinBox [83](#_bookmark316)

classes, helper [84](#_bookmark325)

constructors [86](#_bookmark348)

description [19](#_bookmark36), [21](#_bookmark64)

interface [84](#_bookmark321)

methods [87](#_bookmark349)

properties [84](#_bookmark329), [187](#_bookmark808)

JCSpinBoxEditor

interface for JCSpinBox [84](#_bookmark322)

JCSpinBoxModel

interface for JCSpinBox [84](#_bookmark323)

JCSpinBoxMutableModel

interface for JCSpinBox [84](#_bookmark324)

JCSpinNumberBox [83](#_bookmark317)

classes, helper [84](#_bookmark325)

constructors [86](#_bookmark348)

description [19](#_bookmark37), [21](#_bookmark65)

interface [84](#_bookmark321)

properties [85](#_bookmark339), [188](#_bookmark810)

JCSplashScreen [89](#_bookmark365)

classes [89](#_bookmark367)

constructors [90](#_bookmark369)

description [19](#_bookmark39)

interfaces [89](#_bookmark367)

methods [90](#_bookmark370)

JCSpring

associated components [116](#_bookmark539)

classes

JCSpring, associated component [113](#_bookmark515)

properties [114](#_bookmark525), [189](#_bookmark811)

JCStringTokenizer [122](#_bookmark557), [149](#_bookmark681)

classes [149](#_bookmark685)

methods [150](#_bookmark687)

JCSwingRunnable [153](#_bookmark703)

methods [154](#_bookmark710)

JCSwingTypeConverter [123](#_bookmark562), [158](#_bookmark724)

methods [160](#_bookmark747)

JCSwingUtilities [122](#_bookmark558)

JCTreeExplorer [93](#_bookmark375), [98](#_bookmark387)

classes [97](#_bookmark380)

description [19](#_bookmark40), [21](#_bookmark56)

interfaces [96](#_bookmark379)

methods [99](#_bookmark402)

properties [99](#_bookmark392), [173](#_bookmark786)

JCTreeSet [123](#_bookmark560), [155](#_bookmark713)

constructors [156](#_bookmark717)

methods [156](#_bookmark717)

JCTreeTable [93](#_bookmark375), [98](#_bookmark388)

classes [97](#_bookmark380)

description [20](#_bookmark41), [21](#_bookmark53)

interfaces [96](#_bookmark379)

methods [100](#_bookmark410)

properties [99](#_bookmark396), [174](#_bookmark788)

JCTypeConverter [123](#_bookmark561), [157](#_bookmark720)

methods [158](#_bookmark727)

JCValueEvent

for JCSpinBox and JCSpinNumberBox [84](#_bookmark327)

JCWizard [38](#_bookmark139), [105](#_bookmark463), [107](#_bookmark468)

classes [36](#_bookmark136), [107](#_bookmark467)

description [20](#_bookmark42)

events [109](#_bookmark490)

methods [108](#_bookmark474)

properties [175](#_bookmark789)

JCWizardEvent [107](#_bookmark470), [109](#_bookmark491)

JCWizardListener [107](#_bookmark471), [109](#_bookmark492)

JCWizardPage [107](#_bookmark469)

properties [176](#_bookmark790)

JCWordWrap [123](#_bookmark563), [165](#_bookmark762)

description [22](#_bookmark67)

methods [165](#_bookmark764)

JEditorPane

JClass Elements equivalent [21](#_bookmark45)

JFrame

JClass Elements equivalent [21](#_bookmark52)

JInternalFrame

JClass Elements equivalent [21](#_bookmark50)

JOptionPane [153](#_bookmark702)

JPane

JClass Elements equivalent [21](#_bookmark48)

JTable

JClass Elements equivalent [21](#_bookmark55)

JTree

JClass Elements equivalent [21](#_bookmark54)

### L

last

method in JCWizard [108](#_bookmark481), [109](#_bookmark486)

layout managers [111](#_bookmark505), [114](#_bookmark528)

classes [111](#_bookmark506)

description of SwingSuite’s [17](#_bookmark25), [20](#_bookmark43)

JCAlignLayout [111](#_bookmark507)

JCColumnLayout [111](#_bookmark508)

JCElasticLayout [111](#_bookmark509)

JCGridLayout [112](#_bookmark510)

JCRowLayout [112](#_bookmark511)

list [18](#_bookmark26)

length

JCBrace property [113](#_bookmark524)

levels [125](#_bookmark569)

listeners JCProgressHelper [145](#_bookmark663)

list [141](#_bookmark635)

lists

multi-select [79](#_bookmark292)

### M

makeVisible

JCTreeTable method [101](#_bookmark444)

maximize

method in JCMDIPane [77](#_bookmark286)

maximumDate

property of JCDateChooser [32](#_bookmark115)

MDIPane [71](#_bookmark244)

methods JCCalendar [33](#_bookmark129)

JCCheckBoxList [26](#_bookmark79)

JCDateChooser [33](#_bookmark125)

JCDebug [126](#_bookmark573)

JCEncodeComponent [138](#_bookmark627)

JCExitFrame [44](#_bookmark151)

JCFileFilter [130](#_bookmark595)

JCFontChooser [49](#_bookmark174)

JCHelpPane [54](#_bookmark191)

JCHTMLPane [53](#_bookmark190)

JCIconCreator [133](#_bookmark610)

JCListenerList [141](#_bookmark641)

JCMDIFrame [75](#_bookmark255)

JCMDIPane [75](#_bookmark260)

JCMessageHelper [153](#_bookmark705)

JCMultiSelectList [81](#_bookmark302)

JCProgressHelper [146](#_bookmark665)

JCSortableTable [62](#_bookmark223)

JCSpinBox [87](#_bookmark349)

JCSplashScreen [90](#_bookmark370)

JCStringTokenizer [150](#_bookmark687)

JCSwingRunnable [154](#_bookmark710)

JCSwingTypeConverter [160](#_bookmark747)

JCTreeExplorer [99](#_bookmark402)

JCTreeSet [156](#_bookmark717)

JCTreeTable [100](#_bookmark410)

JCTypeConverter [158](#_bookmark727)

JCWizard [108](#_bookmark474)

JCWordWrap [165](#_bookmark764)

minimumDate

property of JCDateChooser [32](#_bookmark114)

minimumValue

property of JCSpinNumberBox [86](#_bookmark341)

model

property of JCMultiSelectList [80](#_bookmark297)

property of JCSpinBox [85](#_bookmark334)

MonthLabel [31](#_bookmark102)

MonthPopdown [31](#_bookmark103)

months

property of JCDateChooser [32](#_bookmark116)

MonthSpin [31](#_bookmark104)

MonthTable [31](#_bookmark105)

multiple document frame [71](#_bookmark244) multiple document interface [71](#_bookmark249) multi-select list [79](#_bookmark292)

### N

next

method in JCWizard [108](#_bookmark482), [109](#_bookmark487)

nextBegin

method in JCWizardListener [109](#_bookmark493)

nextComplete

method in JCWizardListener [109](#_bookmark494)

nextToken

method in JCStringTokenizer [150](#_bookmark692)

### O

object

list [21](#_bookmark44)

organization

of the manual [17](#_bookmark22)

orientation

JCBox property [113](#_bookmark521)

property of JCMultiSelectList [80](#_bookmark298)

### Q

JCBrace property [113](#_bookmark523)

### P

panes

help [51](#_bookmark178)

HTML [51](#_bookmark178)

parse

method in JCStringTokenizer [150](#_bookmark693)

previous

method in JCWizard [108](#_bookmark483), [109](#_bookmark488)

previousBegin

method in JCWizardListener [109](#_bookmark495)

previousComplete

method in JCWizardListener [109](#_bookmark496) printing debug information [126](#_bookmark574) println

method in JCDebug [126](#_bookmark577)

printStackTrace

method for forcing a stack trace, in JCDebug [127](#_bookmark585)

progress helper [143](#_bookmark649)

properties Beans

reference [169](#_bookmark774)

Color [193](#_bookmark826)

Font [198](#_bookmark830), [199](#_bookmark833)

JCBox [113](#_bookmark519), [177](#_bookmark793)

JCBrace [113](#_bookmark522), [178](#_bookmark794)

JCCalendar [32](#_bookmark120)

JCCheckBoxList [26](#_bookmark77), [179](#_bookmark796)

JCDateChooser [32](#_bookmark111), [189](#_bookmark812)

JCExitFrame [43](#_bookmark148), [180](#_bookmark798)

JCFontChooser [49](#_bookmark170)

JCFontChooserBar [181](#_bookmark800)

JCFontChooserPane [183](#_bookmark802)

JCHelpPane [52](#_bookmark186), [184](#_bookmark803)

JCHTMLPane [52](#_bookmark186), [185](#_bookmark805)

JCMDIFrame [171](#_bookmark781)

JCMDIPane [172](#_bookmark784)

JCMultiSelectList [80](#_bookmark296), [170](#_bookmark780)

JCSortableTable [186](#_bookmark806)

JCSpinBox [84](#_bookmark329), [187](#_bookmark808)

JCSping [189](#_bookmark811)

JCSpinNumberBox [85](#_bookmark339), [188](#_bookmark810)

JCSpring [114](#_bookmark525)

JCTreeExplorer [99](#_bookmark392), [173](#_bookmark786)

JCTreeTable [99](#_bookmark396), [174](#_bookmark788)

JCWizard [175](#_bookmark789)

JCWizardPage [176](#_bookmark790)

prototypeCellValue

quick select [29](#_bookmark92)

### R

read only [30](#_bookmark93)

related documents [13](#_bookmark11)

remove

method of JCListenerList [141](#_bookmark643)

removeActionListener method in JCSpinBox [87](#_bookmark351)

removeAllItems

method in JCSpinBox [87](#_bookmark352)

removeEscape

method in JCTypeConverter [158](#_bookmark732)

removeItem

method in JCSpinBox [87](#_bookmark354)

removeItemAt

method in JCSpinBox [87](#_bookmark355)

removeItemListener method in JCSpinBox [87](#_bookmark357)

removeSpecialDate

property of JCCalendar [32](#_bookmark123) removeTreeExpansionListener JCTreeTable method [101](#_bookmark445)

removeTreeWillExpandListener JCTreeTable method [102](#_bookmark446)

renderer

property of JCSpinBox [85](#_bookmark335)

replace

method in JCWordWrap [165](#_bookmark766)

RGB values [193](#_bookmark826)

rootVisible

property of JCTreeTable [99](#_bookmark399)

run

method in JCSwingRunnable [154](#_bookmark711)

### S

selectedFont

property in JCFontChooserBar [49](#_bookmark172)

selectedIndex

property of JCSpinBox [85](#_bookmark336)

selectedItem

property of JCSpinBox [85](#_bookmark337)

setAutoSort

method in JCSortableTable [63](#_bookmark230)

setCancelString

method in JCProgressHelper [146](#_bookmark669)

setColor

method in JCIconCreator [134](#_bookmark615)

setContentsPage

method in JCHelpPane [54](#_bookmark194)

setDescription

JCFileFilter method [130](#_bookmark599)

setDialogTitle

method in JCProgressHelper [146](#_bookmark670)

setDragMode

method in JCMDIPane [75](#_bookmark264)

setDynamicMessage

method in JCProgressHelper [146](#_bookmark668)

setEnabled

method in JCDebug [126](#_bookmark578)

setEscapeChar

method in JCStringTokenizer [150](#_bookmark694)

setExitOnClose

method in JCExitFrame [44](#_bookmark153)

setExtensionListInDescription JCFileFilter method [130](#_bookmark600)

setFrameManipulationStyle method in JCMDIPane [75](#_bookmark266)

setInitialLayout

method in JCMDIPane [75](#_bookmark267), [76](#_bookmark280)

setKeyColumns

method in JCSortableTable [63](#_bookmark231)

property of JCTreeExplorer [99](#_bookmark395)

setLabelVerticalAlignment JCAlignLayout method [114](#_bookmark533)

setLevel

method in JCDebug [126](#_bookmark580)

setMaximized

method in JCMDIPane [75](#_bookmark269)

setMaximum

method in JCProgressHelper [146](#_bookmark671)

setMDIMenuBar

method in JCMDIFrame [75](#_bookmark256)

method in JCMDIPane [76](#_bookmark271)

setMDIToolBar

method in JCMDIFrame [75](#_bookmark258)

method in JCMDIPane [76](#_bookmark273)

setMinimum

method in JCProgressHelper [146](#_bookmark672)

setModel

method in JCMultiSelectList [81](#_bookmark313)

method in JCSortableTable [63](#_bookmark232)

setNonSelectedIcon

method in JCMDIPane [76](#_bookmark275)

setOperation

property of JCSpinNumberBox [86](#_bookmark343)

setPixels

method in JCIconCreator [134](#_bookmark616)

setPrintStream

method in JCDebug [126](#_bookmark576)

setRange

method in JCProgressHelper [146](#_bookmark673)

setResizeHeight

JCAlignLayout method [114](#_bookmark531)

setResizeWidth JCAlignLayout method [114](#_bookmark532)

setRootVisible

JCTreeTable method [102](#_bookmark447)

setScrollsOnExpand JCTreeTable method [102](#_bookmark448)

setSelectedIcon

method in JCMDIPane [76](#_bookmark277)

setSelectionPath JCTreeTable method [102](#_bookmark449)

setSelectionPaths JCTreeTable method [102](#_bookmark450)

setShowNodeLines JCTreeTable method [102](#_bookmark451)

setShowsRootHandles JCTreeTable method [102](#_bookmark452)

setSize

method in JCIconCreator [134](#_bookmark617)

setSortable

JCTreeTable method [102](#_bookmark453)

setSpinStep

property of JCSpinNumberBox [86](#_bookmark344)

setStaticMessage

method in JCProgressHelper [147](#_bookmark674)

setSwitchPolicy JCTreeTable method [102](#_bookmark454)

setTableHeader

method in JCSortableTable [63](#_bookmark233)

setTag

method in JCDebug [127](#_bookmark582)

setTags

method in JCDebug [127](#_bookmark583)

setTimeToDecideToPopup

method in JCProgressHelper [147](#_bookmark675)

setTimeToPopup

method in JCProgressHelper [147](#_bookmark676)

setTitlePage

method in JCHelpPane [54](#_bookmark197)

setTreeIconRenderer JCTreeExplorer method [100](#_bookmark408)

JCTreeTable method [102](#_bookmark455)

setTreeTableModel JCTreeTable method [102](#_bookmark456)

setTreeTableSelectionModel JCTreeTable method [102](#_bookmark457)

setUI

JCTreeExplorer method [100](#_bookmark409)

JCTreeTable method [102](#_bookmark458)

setUseToolBar

method in JCHelpPane [54](#_bookmark200)

setView

JCTreeTable method [102](#_bookmark459)

setViewPage

method in JCHelpPane [54](#_bookmark199)

setVisible

method in JCSplashScreen [90](#_bookmark371)

shortMonths

property of JCDateChooser [32](#_bookmark117)

show

method in JCWizard [108](#_bookmark484)

showError

method in JCMessageHelper [153](#_bookmark706)

showInformation

method in JCMessageHelper [153](#_bookmark707)

showMessage

method in JCMessageHelper [154](#_bookmark709)

showsRootHandles

property of JCTreeTable [99](#_bookmark400)

showWarning

method in JCMessageHelper [153](#_bookmark708)

sort

mechanism [98](#_bookmark390)

method in JCSortableTable [62](#_bookmark222), [63](#_bookmark234)

utilities [59](#_bookmark203)

SpecialDate interface [32](#_bookmark109)

spin box [83](#_bookmark316)

events [88](#_bookmark360)

spin popdown [29](#_bookmark89)

splash screen [89](#_bookmark365)

stack trace [125](#_bookmark570)

forcing [127](#_bookmark584)

startProgress

method in JCProgressHelper [147](#_bookmark677)

string tokenizer [149](#_bookmark681)

support [14](#_bookmark13)

JClass Community [14](#_bookmark14)

Swing package Beans [170](#_bookmark778)

SwingSuite’s layout managers [111](#_bookmark505)

### T

table components [93](#_bookmark375)

tableChanged

method in JCSortableTable [63](#_bookmark235)

technical support [14](#_bookmark13)

thread safety [123](#_bookmark559)

thread safety classes [153](#_bookmark700)

tileWindowsHorizontally method in JCMDIPane [77](#_bookmark287)

tileWindowsVertically method in JCMDIPane [77](#_bookmark288)

toBoolean

method in JCTypeConverter [159](#_bookmark733)

toColor

method in JCSwingTypeConverter [160](#_bookmark753)

toColorList

method in JCSwingTypeConverter [160](#_bookmark752)

toDate

method in JCTypeConverter [159](#_bookmark734)

toDimension

method in JCSwingTypeConverter [160](#_bookmark750)

toDouble

method in JCTypeConverter [159](#_bookmark735)

toDoubleList

method in JCTypeConverter [159](#_bookmark736)

toEnum

method in JCTypeConverter [159](#_bookmark737)

toEnumList

method in JCTypeConverter [159](#_bookmark738)

toFont

method in JCSwingTypeConverter [160](#_bookmark755)

toInsets

method in JCSwingTypeConverter [160](#_bookmark749)

toInt

method in JCTypeConverter [159](#_bookmark739)

toIntegerList

method in JCTypeConverter [159](#_bookmark740)

toIntList

method in JCTypeConverter [159](#_bookmark741)

toNewLine

method in JCTypeConverter [159](#_bookmark742)

tool tip [30](#_bookmark94), [47](#_bookmark161)

toolTipEnabled

property in JCFontChooserBar [49](#_bookmark171)

toolTipText

property of JCDateChooser [32](#_bookmark118)

property of JCMultiSelectList [80](#_bookmark299)

toPoint

method in JCSwingTypeConverter [160](#_bookmark751)

toString

method in JCEncodeComponent [138](#_bookmark632)

method in JCTypeConverter [159](#_bookmark743)

toStringList

method in JCTypeConverter [159](#_bookmark744)

toVector

method in JCTypeConverter [160](#_bookmark745)

tree

components [93](#_bookmark375)

set [155](#_bookmark713)

treeIconRenderer

property of JCTreeTable [99](#_bookmark397)

treeTableModel

property of JCTreeTable [99](#_bookmark398)

TreeTableSupport [97](#_bookmark382)

TreeWithSortableChildren [98](#_bookmark384)

trim

method in JCTypeConverter [160](#_bookmark746)

type converter [157](#_bookmark720)

typographical conventions [10](#_bookmark5)

### U

unmaximize

method in JCMDPane [77](#_bookmark289)

unsort

method in JCSortableTable [63](#_bookmark236)

updateProgress

method in JCProgressHelper [147](#_bookmark678)

updateUI

JCTreeTable method [102](#_bookmark460)

utilities

classes [121](#_bookmark548)

sorting [59](#_bookmark203)

thread safety [153](#_bookmark700)

### V

value

property of JCDateChooser [32](#_bookmark119)

valueRange

property of JCSpinNumberBox [86](#_bookmark346)

verticalElasticity JCSpring property [114](#_bookmark526)

### W

wizard [105](#_bookmark463)

word wrap [165](#_bookmark762)

wrapText

method in JCWordWrap [165](#_bookmark765)

### Y

YearLabel [31](#_bookmark106)

YearSpin [31](#_bookmark107)