JIDE Components Developer Guide

Purpose of This Document

This document is for developers who want to develop applications using *JIDE Components*.

What is JIDE Components

Generally speaking, each JIDE product focuses in a particular area. For example, *JIDE Docking Framework* provides a flexible and powerful window management system. *JIDE Grids* provides an extension on top of *JTable* and so on. The only product that doesn't have a focus is the *JIDE Components*. We put any component that doesn't belong to a particular area under this umbrella. It contained quite a number of components at the beginning because we introduced all components in *JIDE Common Layer* as part of *JIDE Components*. As of April 2007, we open sourced *JIDE Common Layer* thus reduced the size of *JIDE Components*. So right now, *JIDE Components* only has several major components left. In the future, we will introduce more generic components and add to this product.

Packages

The table below lists the packages in *JIDE Components*. All packages are in jide-components.jar

Packages	**Description**
com.jidesoft.document (in jide-components.jar)	DocumentPane – tabbed-document interface implementation similar to what you see in Visual Studio .NET IDE
com.jidesoft.status (in jide-common.jar)	StatusBar – A generic status bar implementation.
com.jidesoft.pane (in jide-components.jar)	CollapsiblePane, FloorTabbedPane, OutlookTabbedPane etc.
com.jidesoft.alert (in jide-components.jar)	Alert component
com.jidesoft.tooltip (in jide-components.jar)	BalloonTip and ExpandedTip component
com.jidesoft.animation	Animation related classes

COPYRIGHT © 2002-2011 JIDE SOFTWARE. ALL RIGHTS RESERVED

(in jide-components.jar)	
com.jidesoft.navigation (in jide-components.jar)	Navigation related components, such as BreadcrumbBar, NavigationList, NavigationTree and NavigationTable

Document Pane

Background

Most software applications have a "document" concept. For example, in a typical Java IDE, the Java file is the "document". In a photo processing application, an image file such as a gif or jpeg is the "document". The document is usually the center of an application, so selecting the right model to manage those documents is very important.

Looking at the history of how applications organize documents, there are two popular models, known as SDI and MDI.

SDI – Single-Document Interface

Most of early IDEs used SDI - they can only deal with one document at a time. If you want to open another document, you must either close the current one or open a completely new instance of the IDE. Notepad.exe is an example of this type of application. Simple as it is, the drawbacks are obvious. One big drawback is that the user can only view one document at a time. However, because SDI is simple to use there are still a lot of applications using SDI, especially applications for consumers and home users.

MDI – Multiple-Document Interface

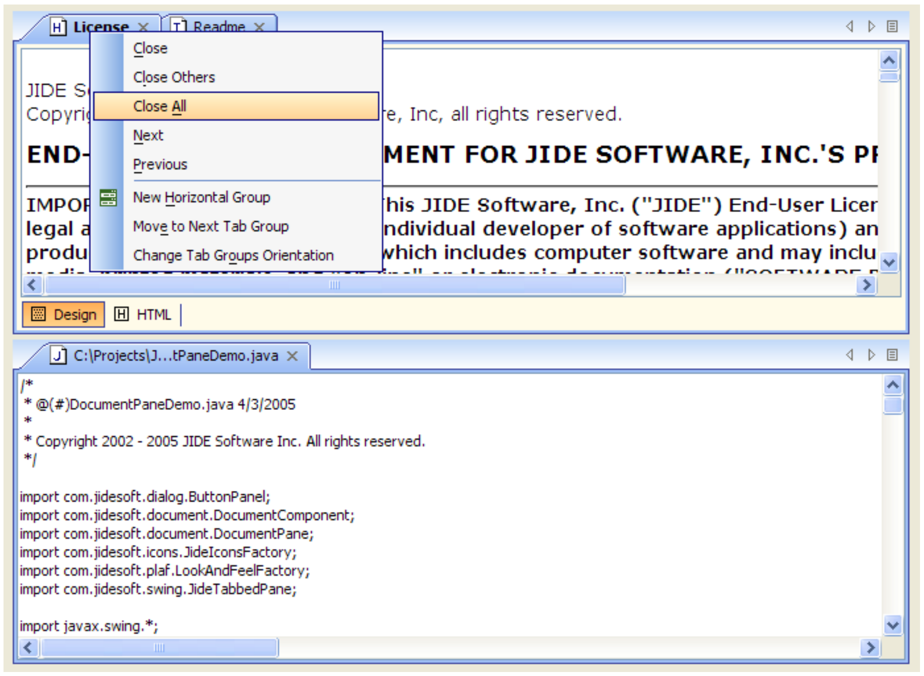
In MDI applications, you can view/edit multiple documents at the same time. You can either use a Windows menu or hotkeys (such as Ctrl-Tab) to navigate between those documents. MDI overcomes the major drawbacks of SDI, but has some drawbacks of its own. The two biggest issues are that it wastes screen space if not maximized because there will be unused space around child windows, and that it is difficult to navigate if maximized. Many users are frustrated when child windows are locked into the parent window in an MDI interface and find it very hard to use. Pure MDI got popular for several years and then died down because of those drawbacks.

If your application is complicated and you are considering using an MDI interface, you should investigate alternative, MDI-like designs. We provide a Tabbed-Document Interface (TDI) to satisfy this need.

In *JIDE Components*, we looked at a new way to view/edit multiple documents – Tabbed-Document Interface. We didn't invent this approach; many IDEs already use an interface of this kind, such as Visual Studio .NET, IntelliJ IDEA, and JBuilder. However, it's an obvious trend in recent software applications. We just want to make it so that everyone can have it.

What does a Tabbed-Document Interface look like?

To understand the tabbed-document interface, you should first understand tabbed panes. Each tab is a document or a document view. We call a tab dedicated to holding a document (as opposed to a side window) a document tab. A document tab shows the document title and an optional icon. You can use the icon to indicate the document type, such as a Java file or XML file. Several document tabs form a document group. Our implementation allows multiple document groups, so that you can view multiple sets of documents at the same time.


License
Readme
Close
Close Others
Close All
Next
Previous
New Horizontal Group
Move to Next Tab Group
Change Tab Groups Orientation
JIDE S
Copyri
e, Inc, all rights reserved.
END-
MENT FOR JIDE SOFTWARE, INC.'S PI
IMPOF
his JIDE Software, Inc. ("JIDE") End-User Licer
legal a
ndividual developer of software applications) an
produ
which includes computer software and may inclu
Design
HTML
C:\Projects\J...tPaneDemo.java

```
/*
 * @(#)DocumentPaneDemo.java 4/3/2005
 *
 * Copyright 2002 - 2005 JIDE Software Inc. All rights reserved.
 */

import com.jidesoft.dialog.ButtonPanel;
import com.jidesoft.document.DocumentComponent;
import com.jidesoft.document.DocumentPane;
import com.jidesoft.icons.JideIconsFactory;
import com.jidesoft.plaf.LookAndFeelFactory;
import com.jidesoft.swing.JideTabbedPane;

import javax.swing.*;
```

The User Interface

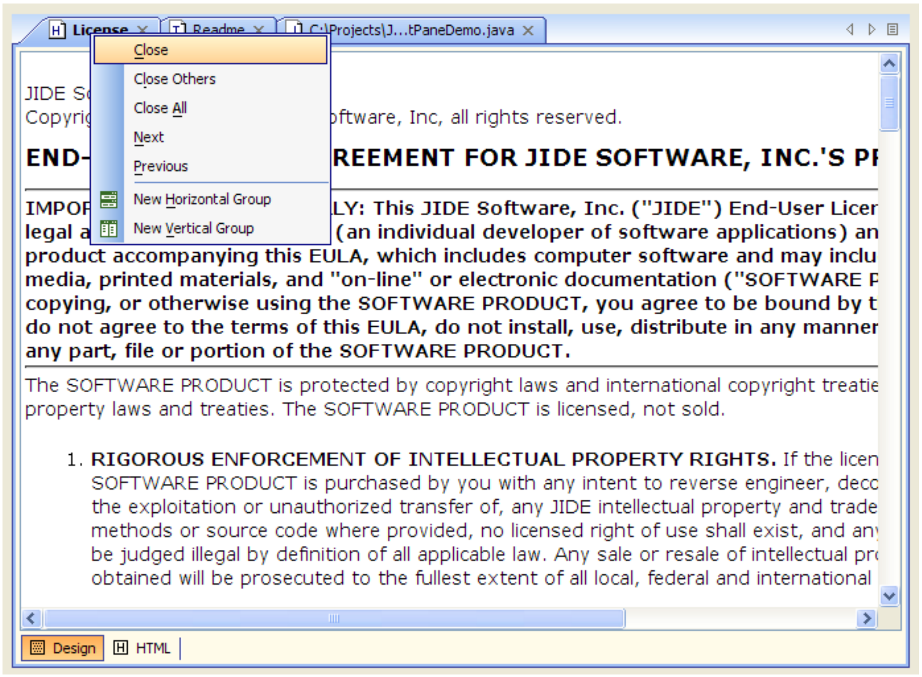
Most commonly used operations related to a document pane can be done through the document tabs.

- Clicking on the tab will activate that document.
- Right-clicking on a document tab will popup a context menu. The built-in menu already comes with a lot of built-in functionality and can be extended.

The screen shot above shows a context menu that appears when a user right-clicks on the "Readme" tab. In the menu, "Close" means close the currently selected document and "Close All" means close all documents (i.e. all the tabs that are within the document tabbed pane). The "x" button to the right is the equivalent to "Close". The "Next" and "Previous" buttons mean set the focus on the next and previous document, respectively. "New Horizontal Group" means create a document group and put the currently selected document into that new group.

Using the screen shot above as an example, after you choose "New Horizontal Group", there will be three document groups, with each group having one document. "Move to Next Tab Group" means move the selected document to next document group. Please note that the context menu only lists actions that are allowed (see below for another example). Since there is only one document group in the example below, you are allowed to create a new document group either vertically or horizontally. That's why you see both "New Horizontal Group" and "New Vertical Group" in the context menu.

COPYRIGHT © 2002-2011 JIDE SOFTWARE. ALL RIGHTS RESERVED



JIDE So
Copyri ftware, Inc, all rights reserved.

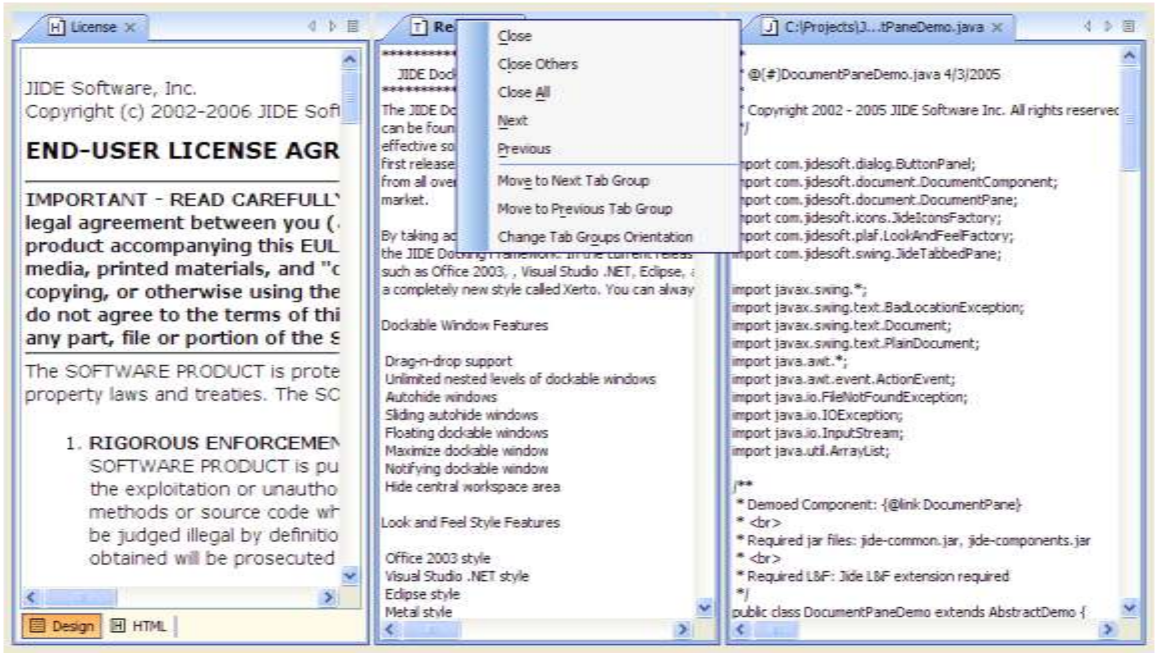
END- REEMENT FOR JIDE SOFTWARE, INC.'S PI

IMPOR LY: This JIDE Software, Inc. ("JIDE") End-User Licer legal a (an individual developer of software applications) an product accompanying this EULA, which includes computer software and may inclu media, printed materials, and "on-line" or electronic documentation ("SOFTWARE P copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by t do not agree to the terms of this EULA, do not install, use, distribute in any manner any part, file or portion of the SOFTWARE PRODUCT.

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treatie property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1. **RIGOROUS ENFORCEMENT OF INTELLECTUAL PROPERTY RIGHTS.** If the licen SOFTWARE PRODUCT is purchased by you with any intent to reverse engineer, deco the exploitation or unauthorized transfer of, any JIDE intellectual property and trade methods or source code where provided, no licensed right of use shall exist, and any be judged illegal by definition of all applicable law. Any sale or resale of intellectual pr obtained will be prosecuted to the fullest extent of all local, federal and international

In the next example, there is no menu choice for either “New Horizontal Group” or “New Vertical Group”. Since there is only one document in the current document group, it doesn’t make sense to create another group. However, you can move it to another group.


License
Re
C:\Projects\J...tPaneDemo.java
Close
Close Others
Close All
Next
Previous
Move to Next Tab Group
Move to Previous Tab Group
Change Tab Groups Orientation
Design
HTML

JIDE Software, Inc.
Copyright (c) 2002-2006 JIDE Soft

END-USER LICENSE AGR

IMPORTANT - READ CAREFULL legal agreement between you (product accompanying this EUL media, printed materials, and "c copying, or otherwise using the do not agree to the terms of thi any part, file or portion of the S

The SOFTWARE PRODUCT is prote property laws and treaties. The SC

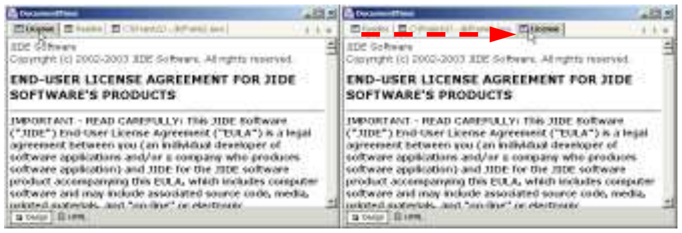
1. **RIGOROUS ENFORCEMEN** SOFTWARE PRODUCT is pu the exploitation or unautho methods or source code wh be judged illegal by definitio obtained will be prosecuted

```
**********
  JIDE Doc
**********
The JIDE Do
can be foun
effective so
first release
from all over
market.

By taking ac
the JIDE Do
such as Office 2003, , Visual Studio .NET, Eclipse, a
a completely new style called Xerto. You can alway

Dockable Window Features

 Drag-n-drop support
 Unlimited nested levels of dockable windows
 Autohide windows
 Sliding autohide windows
 Floating dockable windows
 Maximize dockable window
 Notifying dockable window
 Hide central workspace area

Look and Feel Style Features

 Office 2003 style
 Visual Studio .NET style
 Eclipse style
 Metal style
```

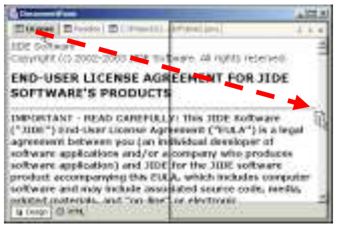
```
*
 * @(#)DocumentPaneDemo.java 4/3/2005
 *
 * Copyright 2002 - 2005 JIDE Software Inc. All rights reservec
 */

port com.jidesoft.dialog.ButtonPanel;
port com.jidesoft.document.DocumentComponent;
port com.jidesoft.document.DocumentPane;
port com.jidesoft.icons.JideIconsFactory;
port com.jidesoft.plaf.LookAndFeelFactory;
mport com.jidesoft.swing.JideTabbedPane;

import javax.swing.*;
import javax.swing.text.BadLocationException;
import javax.swing.text.Document;
import javax.swing.text.PlainDocument;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;

/**
 * Demoed Component: {@link DocumentPane}
 * <br>
 * Required jar files: jide-common.jar, jide-components.jar
 * <br>
 * Required L&F: Jide L&F extension required
 */
public class DocumentPaneDemo extends AbstractDemo {
```

Another, simpler way to move windows and create groups is to use drag and drop.

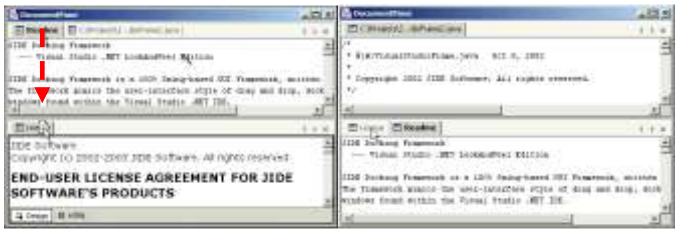
- Drag a document tab horizontally within a tab area and the tab order will change.



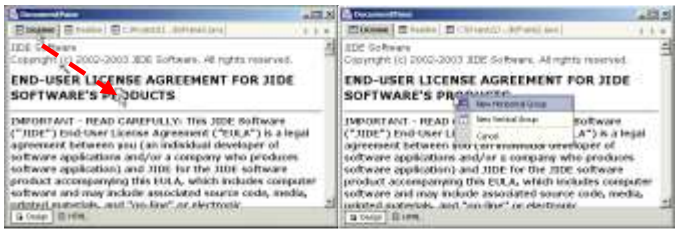
- Dragging a tab and dropping it near the south border or east border will create a new horizontal or vertical tab group, respectively.



- Dragging a tab and dropping it in another document group's tab area will move the document to that document group.



- Dragging a tab and dropping it in the middle of a document will popup a menu allowing you choose what to do.



- Anytime during dragging, pressing ESC will cancel the dragging.

The APIs

The Document

In order to represent the "document" concept, we introduced a class called *DocumentComponent*. It represents the concept of a "document" in a *DocumentPane*. It may not be the actual document in your application. But no matter what kind of document you are dealing with, you can easily wrap it into *DocumentComponent*. A typical usage is to read in a document, create a java.awt.Component that views/edits it, construct a *DocumentComponent* around the Component, and pass the *DocumentComponent* to your DocumentPane.

This is the constructor of a *DocumentComponent*.

```
DocumentComponent(Component component, String name, String title, Icon icon)
```

You must give a unique name to each document component, even if you open the same document twice. If you add a document with a duplicate name, you will get a run-time exception. It's up to you whether you want to catch it or not. You can always check to see if the name is in use by calling *isDocumentOpened(String name)*. Note that this isn't a significant restriction because the name is only a default value for the visible strings.

The class for tabbed-document interface is com.jidesoft.document.DocumentPane.

To open a document (i.e. display it in a given pane), call *openDocument(DocumentComponent document)*.

```
DocumentPane panel = new DocumentPane();
panel.openDocument(new DocumentComponent(
        new  JScrollPane(createTextArea("Readme.txt")),
        "Readme", "Readme", fileIcon));
```

To close a document (i.e. remove it from the panel), call *closeDocument(String name)*. To close all documents, call *closeAll()*.

To activate a document, call *setActiveDocument(String name)*.

To retrieve the current active document, call *getActiveDocument()* or *getActiveDocumentName()*. The former will give you the *DocumentComponent*; the latter will give you just the name. If there is no active document, both return null.

To navigate between open documents, you can use *nextDocument()* and *prevDocument()*.

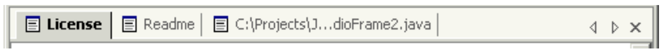
Right mouse clicking on the tab will pop up a context menu. We provide a default menu, but you can customize it by calling *setPopupMenuCustomizer(PopupMenuCustomizer)*.

```
    public static interface PopupMenuCustomizer {
      void  customizePopupMenu(JPopupMenu  menu,  final  IDocumentPane  pane,  final  String
dragComponentName, final IDocumentGroup dropGroup, final boolean onTab);
    }
```

The *getDocumentNames()* method will give you an array of all opened documents, in the order that they were added.

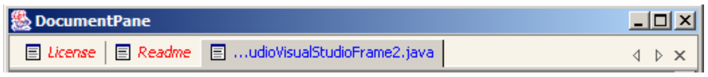
At any time, if you know the name of the document, you can call *getDocument(name)* to get the *DocumentComponent*. If you have a reference to the actual component, calling *getNameOf(component)* will give you the name if the component is opened.

You can also customize the appearance of the title. For the example below, the title of the Java file is too long, so we put "…" in the middle to replace some characters. You can still tell or guess where the file is located and what the file name is.



We understand you may have different requirements, so we allow you to customize both the content and format of the title. Depending on what you want to customize, you can use one of two methods:

- If you just want to display a title with different text but keep the color and style the same (bold for active document and plain for other documents), you can call *DocumentPane.setTitleConverter(StringConverter)*. *StringConverter* is an interface that can convert from an input string to an output string. The input string is the actual title and the output string is what is to be displayed.
- If you want a complete customization of the title, including color, style and text, you can write your own class extending *DocumentComponent* and override the *getDisplayTitle()* method. Since you can use html in the returned display title, you have a lot of flexibility. Below is example source code for coloring the title. In this example, not only is "…" in the beginning of the title, rather than the middle, but also the color is changed.



Below is the code that is used to display the screenshot above.

```
public String getDisplayTitle() {
   // just convert the string to make it shorter for example.
    String title = getDocumentPane().getTitleConverter().convert(getTitle());

     // for active document, display as blue, else display as red and italic font.
    if (getName().equals(getDocumentPane().getActiveDocumentName())) {
        title = "<html><font color=blue>" + title + "</font><html>";
    }
    else {
        title = "<html><font color=red><i>" + title + </i></font>< html >";
   }
    return title;
}
```

Document Group

DocumentPane supports grouping several documents as a document group. There is an *IDocumentGroup* interface for it. By default, we put all documents in the same group in a subclass of *JideTabbedPane* - *TdiGroup*. In fact, you can also use other components to represent document groups such as split pane. You can do that by overriding *createDocumentGroup* method of DocumentPane.

The reason for document groups is for the user to view two documents side-by-side. If your application is simple and there is no requirement to view two documents at the same time, you can disable the feature by calling *setGroupAllowed(false)*. You can also call *setRearrangeAllowed* to disable the whole drag-n-drop or context menu to create new group. This only affects the user. As a developer, you can still use the API to create document group. There is also a *setReorderAllowed* which allows you to disable the drag-n-drop to reorder the document order in the same document group.

To create a new document group programmatically, you can call *newDocumentGroup*. Please note, we don't allow an empty document group. You must put at least one *DocumentComponent* in the group. That's why the *newDocumentGroup* method takes a document name as the first parameter as it will put that document into the newly created document group.

Please note, there is no *removeDocumentGroup* method. When you remove or close the last document from the group, the document group will be removed automatically.

By default, the tabs of documents are on top. However you can call *setTabPlacement(DocumentPane.BOTTOM)* to change them to be at the bottom. If you want more control, you can customize it by creating your own *TabbedPaneCustomizer*. Call *setTabbedPaneCustomizer()* to set your customizer.

DocumentComponentEvent

We also provide listener and event support. A *DocumentComponentEvent* is fired before closing, after opening, after closed, after activated, or after deactivated. Opening a document means *openDocument* is called; closing it means the user triggered the close button or *closeDocument* was called. If you handle the before-closing event *(DOCUMENT_COMPONENT_CLOSING)*, you can call *setAllowClosing(false)* to prevent the document from being closed. Note that this function only affects the current attempt to close. If the user tries to close again, you will need to call *setAllowClosing(false)* again to prevent closing again. Closed means the document was just closed. Activated means the document is selected and deactivated means the opposite. Only one document can be selected or activated at a time. We support five events that are specific to document components.

DOCUMENT_COMPONENT_OPENED

DOCUMENT_COMPONENT_CLOSED

DOCUMENT_COMPONENT_CLOSING

DOCUMENT_COMPONENT _ACTIVATED

DOCUMENT_COMPONENT _DEACTIVATED

COPYRIGHT © 2002-2011 JIDE SOFTWARE, INC. ALL RIGHTS RESERVED

Call *addDocumentComponentListener(listener)* on any *DocumentComponent* instance to add a listener. You can use the same or different instances of listeners on different document components.

Persistence

We provided a very easy way to persist the opened documents. The things we persist include the exact number of document groups and exact documents in each group, selected document in each group, the active group, orientation of the splits, and the dimension of each group. However, since the *DocumentPane* doesn't know how to create *DocumentComponent*, it's your responsibility to remember the list of opened documents when the application closes and create them when the application starts.

Let's use an example. Say you have three files opened in your DocumentPane when application exits. You need to call *documentPane.getDocuments()* to get an iterator of all opened documents, save it somewhere. You should keep enough information so that next time you can create those *DocumentComponents*. After that, call *documentPane.getLayoutPersistence().saveLayoutData()*.

When the application starts, you need to recreate those *DocumentComponents*, add them to a java.utils.List, and set it to the *DocumentPane (see below)*

```
// Create a list of DocumentComponent
List list = new ArrayList();
list.add(documentComponent1);
list.add(documentComponent2);
list.add(documentComponent3);
....

documentPane.setOpenedDocuments(list);
// load the layout of DocumentPane
documentPane.getLayoutPersistence().loadLayoutData();
```

The *LayoutPersistence* is the same class we used to persistent the layout for JIDE Docking Framework and JIDE Action Framework. Knowing that, you can find many ways to save and load the layout. For example, if in your application, you can easily persist a String, you can use *documentPane.getLayoutPersistence().getLayoutData()* to get the layout as a String and save it somewhere. When the application starts, you can call *documentPane.getLayoutPersistence()*. *setLayoutData(String)* to reload the layout.

For those who also use JIDE Docking Framework or JIDE Action Framework, if you want to save all those layouts information in a single file, you can do it too. Just create a *LayoutPersistenceManager* and add all instances of *LayoutPersistence* to this manager.

```
_layoutPersistence = new LayoutPersistenceManager();
```

```
    _layoutPersistence.addLayoutPersistence(getDockableBarManager()); // action
    _layoutPersistence.addLayoutPersistence(getDockingManager()); // dock
    _layoutPersistence.addLayoutPersistence(_documentPane.getLayoutPersistence());              //
DocumentPane

    ……
    // call _layoutPersistence to save and load layout
    // In this case, layoutPersistence.saveLayout() will save the layouts of
    // command bars, dockable frame and documents layout in DocumentPane
    // in a single layout file.
```

UI Defaults used by DocumentPane

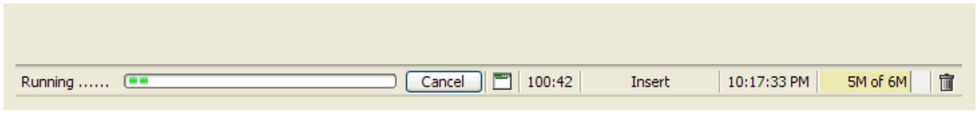
Name	Type	Description
DocumentPane.groupBorder	Border	The border of each document group.
DocumentPane.newHorizontalGroupIcon	Icon	Icon used for action to new a horizontal group
DocumentPane.newVerticalGroupIcon	Icon	Icon used for action to new a vertical group
DocumentPane.boldActiveTab	Boolean	Whether the text on active tab is bold.

Status Bar

A *StatusBar* is never a central part of an application but almost every application has it. A well-designed status bar can make a user interface more user-friendly because it gives immediate, non-intrusive responses to user actions.

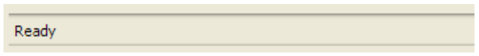
StatusBar and StatusBarItem

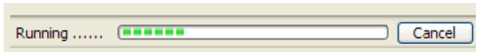
The main class of *StatusBar* is com.jidesoft.swing.StatusBar. It is divided into a series of items of type *StatusBarItem*. In the example below, the whole thing is the StatusBar and each gray box is a *StatusBarItem*. You can use one of several *StatusBarItem's* that we developed or you can create your own *StatusBarItem*.



ProgressStatusBarItem

ProgressStatusBarItem is usually the widest item on status bar. It can display a text message or it can display progress in an optional progress bar. You can activate the progress bar by calling *setProgressStatus* to some string. This method call will show the built-in progress bar, then you can set the percentage finished using *setProgress(int)*. In the below status bars, the second one is in running mode.



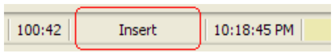


Method	**Description**
setStatus(String)	Set the status such as "Ready" in the screen shot above
setProgressStatus(String)	Set the status mode when progress bar is visible, such as "Running" in the screenshot above
setProgress(int)	int is a value from 1 to 100, representing the percentage on the progress bar. When this reaches 100, the progress bar will be hidden automatically (so you might want to round down).
setCancelCallback(CancelCallback)	At the end of progress bar there is a Cancel button. You can add a callback to do something if the user presses the Cancel button. If the cancelCallback you passed in is null then the Cancel button will not be shown. See interface *CancelCallback* in the JavaDoc for more details.

COPYRIGHT © 2002-2011 JIDE SOFTWARE. ALL RIGHTS RESERVED

LabelStatusBarItem

If you just want to display a text on status bar, you can use this. It is simply a *JLabel* inside a *JPanel*. Below is an example.



Method	Description
setText(String)	Sets the text to be displayed on the label
setToolTip(String)	Sets the tool tip to be displayed on the label
setIcon(Icon)	Sets the icon to be displayed on the label
addMouseListener(MouseListener)	Adds the specified mouse listener to receive mouse events
setAlignment(int)	Sets the alignment. The value can be JLabel.CENTER, JLabel.LEFT, or JLabel.RIGHT.
getComponent()	Gets the actual component - in this case it's the JLabel.

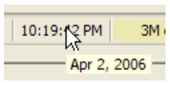
ButtonStatusBarItem

This is similar to *LabelStatusBarItem*, just replace the *JLabel* with a *JButton*.

Method	Description
setText(String)	Sets the text to be displayed on the button
setToolTip(String)	Sets the tool tip to be displayed on the button
setIcon(Icon)	Sets the icon to be displayed on the button
addMouseListener(MouseListener)	Adds the specified mouse listener to receive mouse events
getComponent()	Gets the actual component - in this case it's the JButton.

TimeStatusBarItem

TimeStatusBarItem is used to display a clock or calendar on the status bar. You can customize the format and display characteristics of the time and date.

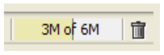


Method	Description
setUpdateInterval(int)	How often to do a refresh, in ms (the default is 500 ms). If you don't display the seconds field, or you just display the date then you can change the interval to a much larger number.
setTextFormat(DateFormat)	Set the DateFormat that formats the text displayed on the status bar. **Note: you can use a localized version of DateFormat to do the localization.**
setTooltipFormat (DateFormat)	Set the DateFormat that formats the text of the tool tip displayed when the mouse hovers over the date.

TimeStatusBarItem extends *LabelStatusBar* so it also has all methods of *LabelStatusBarItem* that you can use.

MemoryStatusBarItem

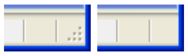
We borrowed the idea of *MemoryStatusBarItem* from IntelliJ IDEA. The *MemoryStatusBarItem* is used to display currently used memory vs. total memory in the current JVM. It also allows the user to manually run garbage collection by pressing the garbage can button. In the example below, 3M is the amount of used memory and 4M is the total memory.



Method	Description
No public methods	

ResizableStatusBarItem

ResizableStatusBarItem is a special status bar item that can be dragged to resize the *JFrame*. It should be used as the last status bar item on status bar. If the *JFrame* is not maximized and resizable is true, you will see a triangle shadow like below to indicate you can drag to resize (the first screenshot below). If the *JFrame* is maximized thus not resizable, the triangle shadow will disappear to indicate it is not resizable (the second screenshot below).



Method	Description
No public methods	

General Comments on StatusBar

Size of StatusBarItem

StatusBar uses *JideBoxLayout* (if you prefer, you can skip this part and jump to Layout section for how to use JideBoxLayout). Thus, when you add the *StatusBarItem*, you can add it as VARY, FLEXIBLE or FIX. Usually, *ProgressStatusBarItem* should add as VARY because you want it to be the only one that resizes. All the others should be added as FLEXIBLE. If you want the width of a FLEXIBLE StatusBarItem to be fixed, you can call *setPreferredWidth(int)*. If you don't do so, *StatusBarItem* will resize as the content changes.

The gap between *StatusBarItem's* is also defined in *UIDefaults* as "StatusBar.gap" (with the default being 2 pixels).

Border of StatusBarItem

No border should be specified by a *StatusBarItem* itself. When it is added to StatusBar, the StatusBar will call *setBorder* on it, so that all items have the same border. The border is also defined in UIDefaults as "StatusBarItem.border".

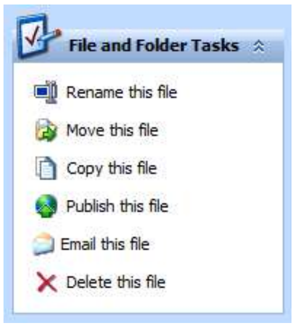
UI Defaults used by StatusBar

Name	Type	Description
StatusBarItem.border	Border	The border of each status bar item
StatusBar.margin	Insets	The margin of the whole status bar
StatusBar.gap	Integer	The gap between each status bar item
StatusBar.background	Color	The background of status bar
StatusBar.font	Font	The font used by status bar

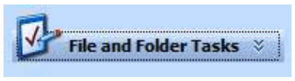
CollapsiblePane

CollapsiblePane, as the name indicates, is a pane which can be collapsed (see below for an example).

Before collapsing …



After collapsing …



It's actually very easy to create a CollapsiblePane - see below for the code that creates the CollapsiblePane shown above. All you need to do are to set the title bar title (and title bar icon if any) and the content pane. Then the content pane will be hidden or shown when user clicks on the collapse button on title bar.

```
CollapsiblePane panel = new CollapsiblePane("File and Folder Tasks");
JPanel labelPanel = new JPanel();
labelPanel.setLayout(new GridLayout(6, 1, 1, 0));
labelPanel.add(createHyperlinkButton("Rename this file",
SampleIconsFactory.getImageIcon(SampleIconsFactory.CollapsiblePane.RENAME)));
labelPanel.add(createHyperlinkButton("Move this file",
SampleIconsFactory.getImageIcon(SampleIconsFactory.CollapsiblePane.MOVE)));
labelPanel.add(createHyperlinkButton("Copy this file",
SampleIconsFactory.getImageIcon(SampleIconsFactory.CollapsiblePane.COPY)));
labelPanel.add(createHyperlinkButton("Publish this file",
SampleIconsFactory.getImageIcon(SampleIconsFactory.CollapsiblePane.PUBLISH)));
labelPanel.add(createHyperlinkButton("Email this file",
SampleIconsFactory.getImageIcon(SampleIconsFactory.CollapsiblePane.EMAIL)));
labelPanel.add(createHyperlinkButton("Delete this file",
SampleIconsFactory.getImageIcon(SampleIconsFactory.CollapsiblePane.DELET)));
labelPanel.setOpaque(true);
labelPanel.setBackground(Color.WHITE);
panel.setContentPane(labelPanel);
return panel;
```

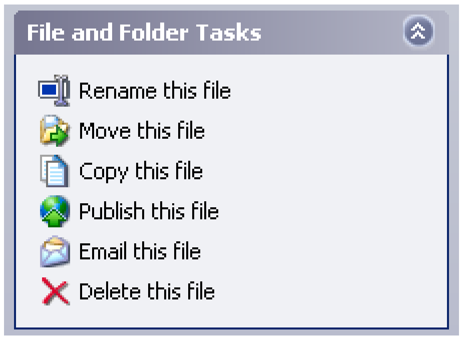
CollapsiblePane has a corresponding Component UI class, called CollapsiblePaneUI. Here are keys of the UIDefaults which you can customize.

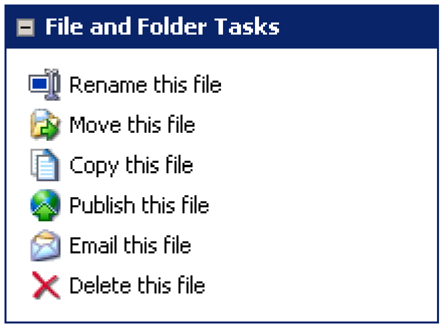
Name	Type	Description
CollapsiblePane.background	Color	Background
CollapsiblePane.foreground	Color	Foreground
CollapsiblePane.border	Border	Border of the collapsible pane
CollapsiblePane.font	Font	Font
CollapsiblePane.contentBorder	Border	Border of the content pane
CollapsiblePane.titleBorder	Border	Border of the title bar
CollapsiblePane.titleFont	Font	Font of the title bar.
CollapsiblePane.collapseText	String	Text for the collapse/expand button

CollapsiblePane.collapseToolTip	String	ToolTip of the collapse/expand button
CollapsiblePane.upIcon	Icon	Icon of the button when it means collapse
CollapsiblePane.downIcon	Icon	Icon of the button when it means expand

When the content pane is collapsed or expanded a CollapsiblePaneEvent will be fired. There are four types of events, which will be fired at different times. When the content pane begins to expand, COLLAPSIBLE_PANE_EXPANDING is fired. When it finished expanding, COLLAPSIBLE_PANE_EXPANDED will be fired. Correspondingly when content is collapsed, there COLLAPSIBLE_PANE_COLLAPSING and COLLAPSIBLE_PANE_COLLAPSED events will be fired. You can use these events to control the content pane - to initialize, load or save etc. You just call collapsiblePane.addCollapsiblePaneListener to add such a listener.

To enrich the visual effect, you can also set a different style to CollapsiblePane. For example, if you call pane.setStyle(CollapsiblePane.TREE_STYLE), you will see a tree-style CollapsiblePane as shown in the screenshot below. There are three styles available – DROPDOWN_STYLE, TREE_STYLE and PLAIN_STYLE. The default style is DROPDOWN_STYLE. You can see an example of three styles in the picture below.





COPYRIGHT © 2002-2011 JIDE SOFTWARE, INC. ALL RIGHTS RESERVED

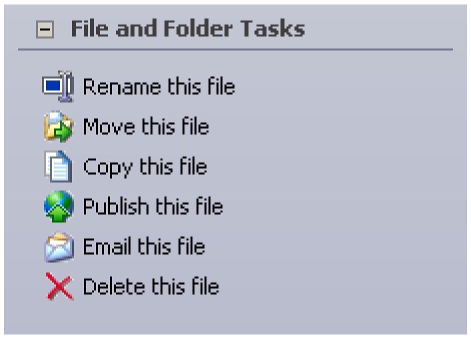


Figure 1 Different styles

You can also set an icon for the *CollapsiblePane*, which will appear on the title bar before the title text. Although this uses a gap of 4 pixels by default, you can change this by calling *setIconTextGap()*. In addition, you can set different alignments by calling *setVerticalAlignment()*, *setVerticalTextPosition(), setHorizontalAlignment(),* or *setHorizontalTextPosition()*;

CollapsiblePane has a special model called emphasized mode. In this mode, the title bar of collapsible pane will have different background and foreground from regular modes so that it can stand out among other collapsible panes. To set this mode, simply call *setEmphasized(true)*.

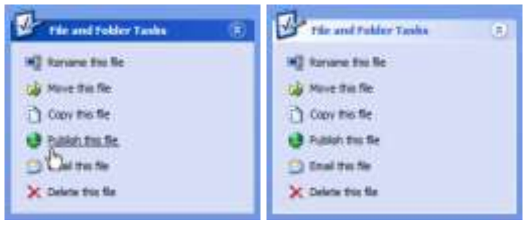


Figure 2 Emphasized v.s. Non-emphasized CollapsiblePane

CollapsiblePane supports several different LookAndFeels such as regular Windows L&F, Windows XP L&F, Eclipse L&F and Metal L&F (see below)

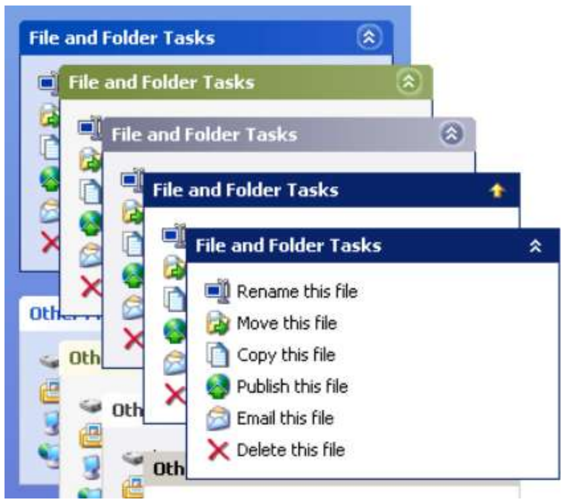


Figure 3 Collapsible Pane under different L&Fs

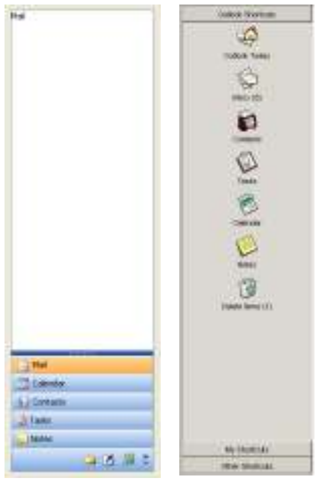
Using several CollapsiblePanes as a group

Usually several *CollapsiblePane*s are used together in a group. We created a special panel called *CollapsiblePanes* to make this usage easier. It's nothing but JPanel using JideBoxLayout allowing you to add several *CollapsiblePane* to it. The panel has different backgrounds under different L&Fs. See below for how to use it.

```
CollapsiblePanes panes = new CollapsiblePanes();
panes.add(collapsiblePane1);
collapsiblePane1.setEmphasized(true); // option to make one of the collapsible pane emphasized.
panes.add(collapsiblePane2);
panes.add(collapsiblePane3);
panes.addExpansion(); // to push collapsible panes upward when they are collapsed.
```

COPYRIGHT © 2002-2011 JIDE SOFTWARE, INC. ALL RIGHTS RESERVED

OutlookTabbedPane/FloorTabbedPane



Both *OutlookTabbedPane* and *FloorTabbedPane* extend *JTabbedPane*. A typical tabbed pane has many panels and corresponding tabs. The user can click on a tab to choose which panel to view. Although an *OutlookTabbedPane* or a *FloorTabbedPane* also has many panels, instead of using tabs, it just uses buttons to switch between panels. The buttons are organized vertically, as floors in a storied building (that's how it gets the name of *FloorTabbedPane*). In the case of *OutlookTabbedPane*, the button position remains at the same location. In the case of *FloorTabbedPane*, the button location changes. One famous example of it is the Outlook Navigation Pane in the Microsoft Outlook product. *FloorTabbedPane* implements the navigation pane style in Outlook 2000. *OutlookTabbedPane* implements the same navigation pane in Outlook 2003.

Although *OutlookTabbedPane* and *FloorTabbedPane* extend *JTabbedPane*, there are no additional public methods on them, so you can use it in place of an existing *JTabbedPane*.

There is no corresponding ComponentUI of this class. The buttons will have the same style as the JideButton, and the components will have whatever style was specified when the user created them.

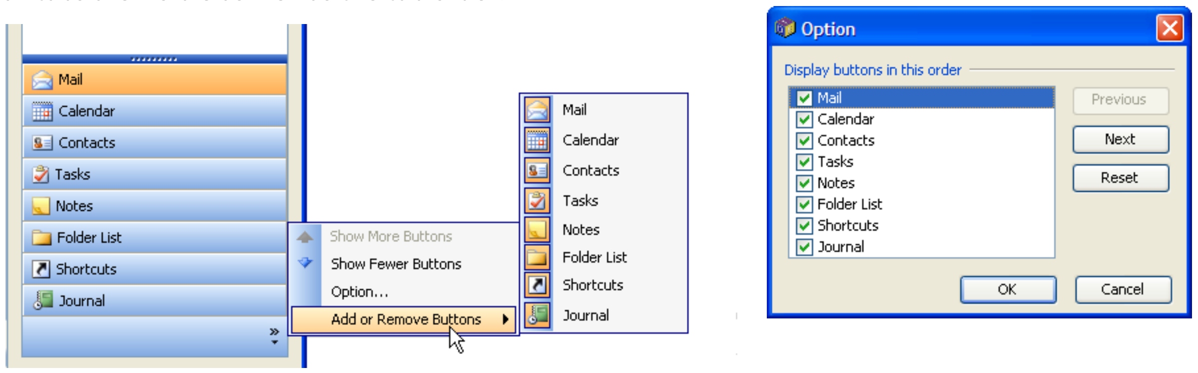
In case you want to provide your own button replace the buttons used by *FloorTabbedPane*, you can extend *FloorTabbedPane* and override *createButton()* method. This is the default implementation.

```
    protected AbstractButton createButton(Action action) {
        return new FloorButton(action);
  }

    private class FloorButton extends JideButton implements UIResource {
        public FloorButton(Action a) {
            super(a);
            setButtonStyle(TOOLBOX_STYLE);
        }
    }
```

In your overridden implementation, you can create whatever button you want. However the button must implement *UIResource*. If not, our code will check it and throw an *IllegalArgumentException*.

For *OutlookTabbedPane*, there are two types of buttons – the full size buttons that appear on top and the icon only buttons which appear on the bottom panel. You can customize those two buttons by overriding *customizeButton(AbstractButton button)* and *customizeBottomButton(AbstractButton button)* respectively.

Depending on the available vertical space, *OutlookTabbedPane* can be adjusted to show tabs as full size buttons or as icon only buttons. *OutlookTabbedPane* also allows user to choose which tabs are visible as well as the tab order.



Alert

Alert is a special popup component which can be used to display messages such as new email notification, warning or error messages. The function is very similar to Swing's *JOptionPane* message box except *Alert* component doesn't block.

For example, you just want to inform your user that a job they submitted a while ago is completed. You could use *StatusBar* but in this particular case, you want something more obvious than StatusBar as people tend to ignore the message on *StatusBar*. Before you have *Alert* component, you will use *JOptionPane* with OK button. From usability point of view, this is a bad design because user has to stop what he/she is working on and click on the OK button to dismiss the dialog. If you use *Alert* in this use case, it will be much better. User can still work on whatever he/she is working on and just take a peek at the *Alert* and know the job is complete. After a few seconds, the *Alert* window will disappear automatically.

To make it interesting, *Alert* supports all kinds of custom animations when entering and exiting. For example, it supports fly-in or fly-out, fade-in or fade-out. You can also position it any location of the screen including nice pre-set locations (four corners, four sides and center). When it flies in or out, you can also set different path - straight line or curve or wave. All those features will enhance the appearance and effect of *Alert* so that it can deliver the right message to end users.

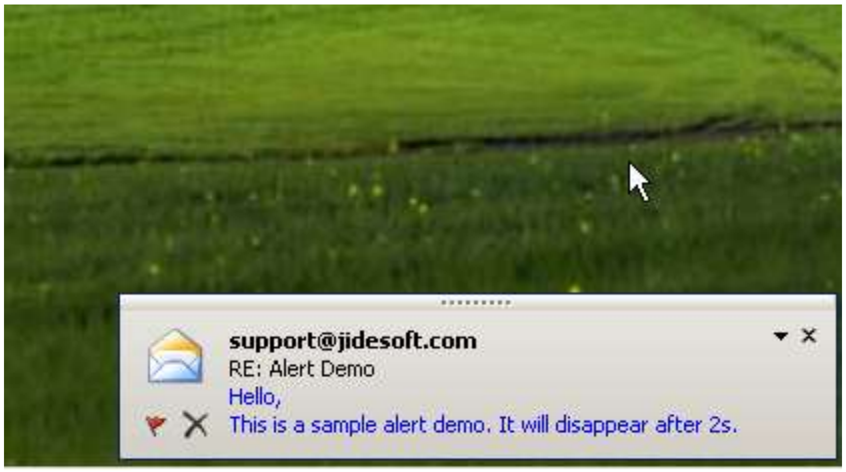

support@jidesoft.com
RE: Alert Demo
Hello,
This is a sample alert demo. It will disappear after 2s.

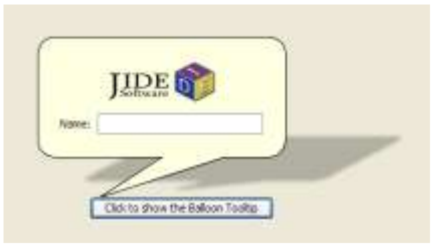
Figure 4 Alert example

To create your own *Alert*, you just need to *new Alert()*, then add contents to its content pane. You probably should setup a few things such as *setTimeout(int ms)* so that it will hide automatically after a few seconds. To display the *Alert*, you call one of the *showPopup()* methods.

If you didn't set the custom animation, no animation will be used. If you want to learn how to create *CustomAnimation*, please see next section.

BalloonTip

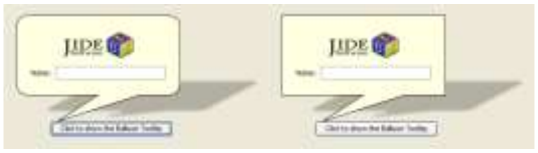
BalloonTip is a special tooltip that has a balloon shape. See below.

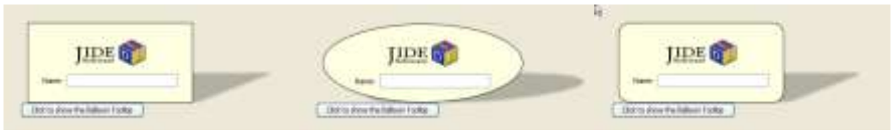


You can put any component inside a *BalloonTip*. It can be done either through the constructor or use setContentPane() method. Then all you need to do is to call show(owner, x, y) to display it. Please note the x and y position is relative to the owner

Different balloon shape

BalloonTip supports different balloon shapes. By default we support several shapes. You can use *setBalloonShape* method to change it (see below).

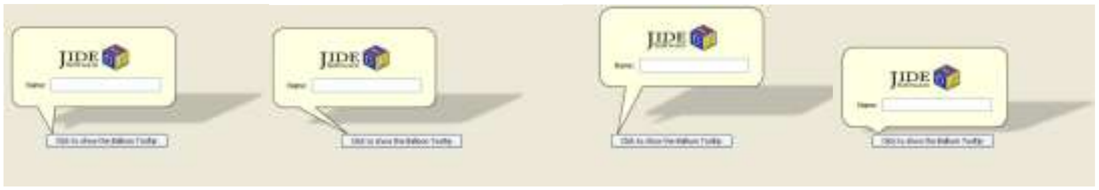




They are *RoundedRectangularBalloonShape*, *RectangularBalloonShape*, *RectangularShape*, *OvalShape* and *RoundedRectangularShape*, and respectively.

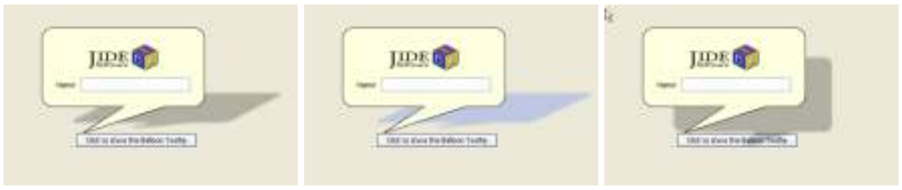
You can also define your own shapes by implementing *BalloonShape* interface or extending one of our existing shapes with overridden methods.

The *RoundedRectangularBalloonShape* and *RectangularBalloonShape* supports adjustments of several values. See below for a few examples.



Shadow Style

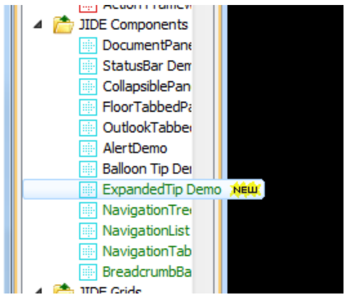
BalloonTip also support two different kinds of shadows – perspective shadow and drop shadow. It can be changed using *setShadowStyle*. Same as *BalloonShape*, *ShadowStyle* is also an interface. You can use one of the existing *ShadowStyle* we created or you can create your own style. For our default shadow styles, you can adjust various settings.

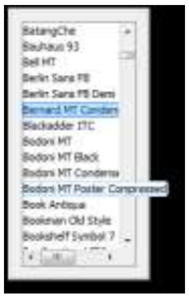


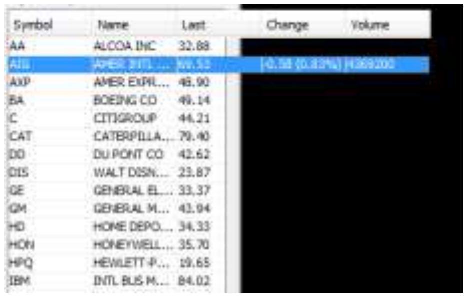
COPYRIGHT © 2002-2011 JIDE SOFTWARE, INC. ALL RIGHTS RESERVED

ExpandedTip

ExpandedTip makes a component showing an expanded tool-tip when the mouse is a node or a cell. *ExpandedTip* is the base class and has subclasses to implement the feature for different components. It is useful for JList, JTree and JTable when they are narrow but the content is wide.







Symbol	Name	Last	Change	Volume
AA	ALCOA INC	32.88		
AIG	AMER INTL ...	[illegible]	[illegible]	[illegible]
AXP	AMER EXPR...	48.90		
BA	BOEING CO	49.14		
C	CITIGROUP	44.21		
CAT	CATERPILLA...	79.40		
DD	DU PONT CO	42.62		
DIS	WALT DISN...	23.87		
GE	GENERAL EL...	33.37		
GM	GENERAL M...	43.94		
HD	HOME DEPO...	34.33		
HON	HONEYWELL...	35.70		
HPQ	HEWLETT-P...	19.65		
IBM	INTL BUS M...	84.02		

It is extremely easy to use. If you have a JList, JTree, or a JTable, Just call

```
ExpandedTipUtils.install(list);
```

or

```
ExpandedTipUtils.install(tree);
```

or

```
ExpandedTipUtils.install(table);
ExpandedTipUtils.install(table.getTableHeader());
```

For the table, it doesn't help too much if only the cell values are shown. That's why we also make it possible to show the expanded tip for the table header along with the cells.

The *ExpandedTipUtils* will call the corresponding *ListExpandedTip*, *TreeExpandedTip* and *TableExpandedTip*. Of course you can use these classes directly if you want. It would be

```
new ListExpandedTip(list);
```

ExpandedTip is general enough to handle any component that is inside a *JScrollPane*. You can subclass *ExpandedTip* and implement the following two methods.

```
protected abstract Rectangle getRowBounds(int index);
protected abstract int rowAtPoint(Point point);
```

COPYRIGHT © 2002-2011 JIDE SOFTWARE. ALL RIGHTS RESERVED

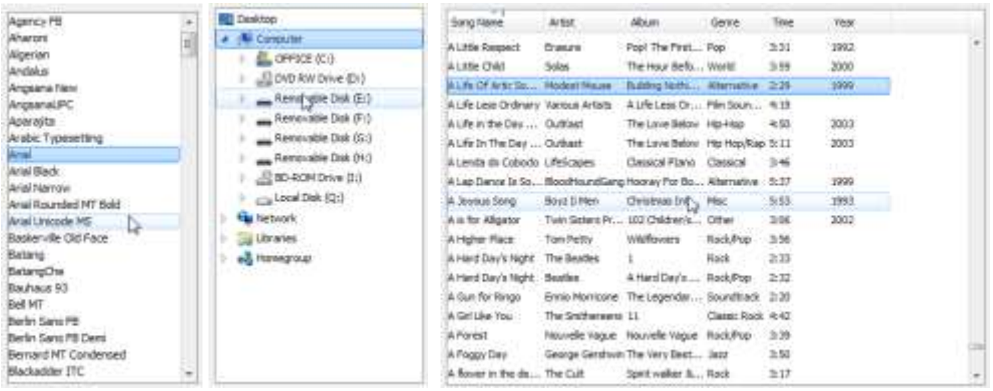
Because the tip is expanded horizontally outside the scroll pane, that's why it expected the component can be sliced into rows. First, *ExpandedTip* will install a mouse motion listener and call *rowAtPoint* to find out which row the mouse is on. We called it row but it doesn't have to be row. For the JList, it will be the row; For the JTree, it will be the tree node; for a text area or code editor, it could be the line. Then *getRowBounds* will be called to figure out if the bounds is larger than the visible bounds. If it is smaller, do nothing as it means the content of the row is fully visible. If the bounds is larger than the visible bounds, *ExpandedTip* will prepare a tip window and paint the content that is outside the viewport of the scroll pane.

For example, let's say you have a large image *JLabel* inside a *JScrollPane* and the image is only partially visible. You can write code like below.

```
JLabel imageLabel = new JLabel(an image);
new ExpandedTip<JLabel>(imageLabel){
        protected Rectangle getRowBounds(int index) {
           return new Rectangle(0,0, imageLabel.getWidth(), imageLabel.getHeight());
        }

        @Override
        protected int rowAtPoint(Point point) {
           return 0;
        }
    };
```

Node that I treat the whole thing as just one row and the row bounds is the whole label's bounds. If you run the code, you will see the whole image is expanded outside the *JScrollPane* so that you can see the whole image when the mouse is over it.

Navigation Components



Song Name	Artist	Album	Genre	Time	Year
A Little Respect	Erasure	Pop! The First...	Pop	3:31	1992
A Little Child	Solas	The Hour Befo...	World	3:59	2000
A Life Of Artic So...	Modest Mouse	Building Nothi...	Alternative	2:29	1999
A Life Less Ordinary	Various Artists	A Life Less Or...	Film Soun...	4:13	
A Life in the Day ...	Outkast	The Love Below	Hip-Hop	4:50	2003
A Life In The Day ...	Outkast	The Love Below	Hip Hop/Rap	5:11	2003
A Lenda do Cobodo	LifeScapes	Classical Piano	Classical	3:46	
A Lap Dance Is So...	BloodHoundGang	Hooray For Bo...	Alternative	5:37	1999
A Joyous Song	Boyz II Men	Christmas Inf...	Misc	5:53	1993
A is for Alligator	Twin Sisters Pr...	102 Children's...	Other	3:06	2002
A Higher Place	Tom Petty	Wildflowers	Rock/Pop	3:56	
A Hard Day's Night	The Beatles	1	Rock	2:33	
A Hard Day's Night	Beatles	A Hard Day's ...	Rock/Pop	2:32	
A Gun for Ringo	Ennio Morricone	The Legendar...	Soundtrack	2:20	
A Girl Like You	The Smithereens	11	Classic Rock	4:42	
A Forest	Nouvelle Vague	Nouvelle Vague	Rock/Pop	3:39	
A Foggy Day	George Gershwin	The Very Best...	Jazz	3:50	
A flower in the da...	The Cult	Spirit walker &...	Rock	3:17	

Navigation Components are components that are designed to assist navigation in the user interface. Navigation components are often displayed on the left hand side of the application to help guide users to various items inside the application. Major Java IDEs and many other famous

COPYRIGHT © 2002-2011 JIDE SOFTWARE, INC. ALL RIGHTS RESERVED

applications such as Microsoft Outlook and iTunes do it this way. The navigation components could be a list, a tree, or a table. However, in Swing, the regular *JList*, *JTree* and *JTable* are not friendly enough to be used for the navigation purpose. For example, in *JTree*, the selection only highlights the node. If the tree node is very narrow, the selection is barely noticeable. Or the opposite when the tree node is very wide but the tree is narrow - the user can't see the whole node except by scrolling it horizontally (or using the *ExpandedTip* feature described above). In order to make these components more suitable for navigation, we changed their default behavior and created three subclasses - *NavigationList*, *NavigationTree* and *NavigationTable*. All three have a unique selection/rollover effect. The *NavigationTree* even has the tree icon fading effect that you can find on Windows 7's File Explorer. If you happen to use *JList*, *JTree* or *JTable* as your navigation components, you can replace them directly with these new components without breaking the existing code.

NavigationList

NavigationList is a special *JList* that is designed for the navigation purpose. It has the following features that are different from what *JList* has.

- It has row rollover effect. This feature gives user a clear indication of what will be selected when he/she clicks.
- The row selection covers the whole row. This feature will show the selected item more clearly. This feature is optional and can be turned off using *setWideSelection(false)*.
- The selection highlight is different when focused and not focused. If several components are used in the navigation area, this feature will allow user to find out which component is currently active when they all have selection.
- It supports expanded tip so that content is still visible when the component is very narrow. This is very important as in most cases the navigation area is narrow. This feature is optional and can be turned off using *setExpandedTip(false)*.

The selection and rollover effect is painted inside the *paintComponent* methods of the *NavigationList* after the original list content is painted. We don't to use the cell renderer to paint the effect as users might create their own cell renderer. However in order to prevent the cell renderer from painting the default selection effect, we set a custom cell renderer that extends *DefaultListCellRenderer* and pass in false for the *cellHasFocus* and *isSelected* parameter in when calling *getListCellRendererComponent* method. If you have your own cell renderer, please make sure you do the same thing.

NavigationTree

NavigationTree is a special *JTree* that is designed for the navigation purpose. It has the following features that are different from what *JTree* has.

- It has row rollover effect. This feature gives user a clear indication of what will be selected when he/she clicks.
- The tree icons will be hidden when the mouse is not over and the tree is not focused. And it has fade effect when the mouse moves away from the tree. This feature is to

avoid the icons cluttering the user interface. This feature is optional and can be turned off using *setFadeIcon(false)*.

- The tree lines are hidden by default. This is again to avoid the cluttering of the user interface. This feature is optional and can be turned off using *setShowTreeLines(false)*.
- The row selection covers the whole row. This feature will show the selected item more clearly. This feature is optional and can be turned off using *setWideSelection(false)*.
- The selection highlight is different when focused and not focused. If several components are used in the navigation area, this feature will allow user to find out which component is currently active when they all have selection.
- It supports expanded tip so that node content is still visible when the component is very narrow. This is very important as in most cases the navigation area narrow. This feature is optional and can be turned off using *setExpandedTip(false)*.

The selection and rollover effect is painted inside the *paintComponent* methods of the *NavigationTree* after the original tree content is painted. We can't use the cell renderer to paint the effect as it is beyond the bounds of the cell renderers. So in order to prevent the cell renderer from painting the default selection effect, we override *isRowSelected(int)* method to always return false. If your code depends on this method to find out if a row is selected, you can use *isRowSelectedOriginal(int)* method.

NavigationTable

NavigationTable is a special *JTable* that is designed for the navigation purpose. It has the following features that are different from what *JTable* has.

- It has row rollover effect. This feature gives user a clear indication of what will be selected when he/she clicks.
- The row selection covers the whole row instead of each cell has selection individually in the case of JTable.
- The selection highlight is different when focused and not focused. If several components are used in the navigation area, this feature will allow user to find out which component is currently active when they all have selection.
- It supports expanded tip so that content is still visible when the list is very narrow. This is very important as in most cases the navigation area are narrow. This feature is optional and can be turned off using *setExpandedTip(false)*.

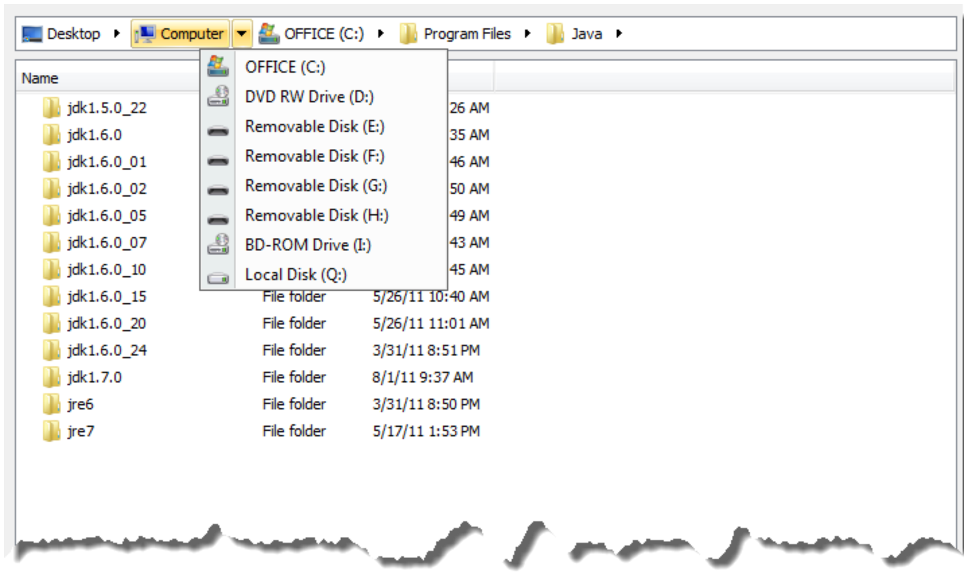
The selection and rollover effect is painted inside the *paintComponent* methods of the *NavigationTable* after the original table content is painted. However in order to prevent the cell renderer from painting the default selection effect, we override *prepareRenderer* method to pass in false for both *isSelected* and *hasFocus* methods when getting the renderer component from the cell renderers.

We have many table subclasses in JIDE Grids product. Unfortunately Java doesn't multiple inheritance so we have to create *NavigationSortableTable* and *NavigationTreeTable* to archive

COPYRIGHT © 2002-2011 JIDE SOFTWARE, INC. ALL RIGHTS RESERVED

the navigation feature in *SortableTable* and *TreeTable* respectively. We could do it for all other table subclasses but we think these two are more likely to be used for the navigation purpose.

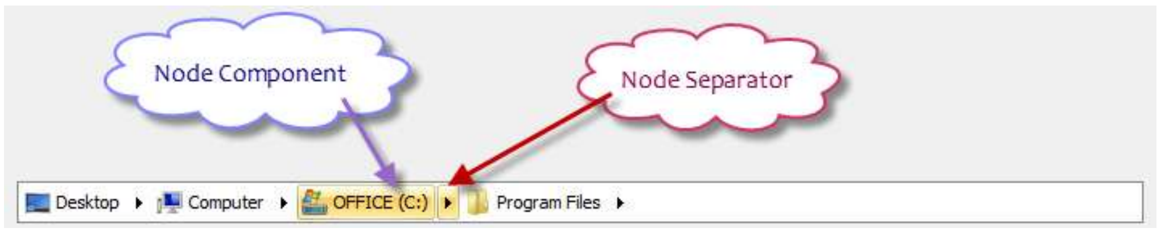
BreadcrumbBar



Breadcrumbs are navigation aids used in user interfaces. They allow users to keep track of their location within programs or documents. The term comes from the trail of breadcrumbs left by Hansel and Gretel in the popular fairytale. Both Windows 7 and Mac OS X introduced breadcrumb components in the file explorer and the finder respectively.

JIDE's *BreadcrumbBar* implements the breadcrumb component using Swing. To leverage the existing Swing code, it uses the exactly the same *TreeModel* used in *JTree* as the model class. The path displayed in the breadcrumb is simply the current *TreePath*. If you know how to setup a *TreeModel* and are familiar with *TreePaths*, you will find it extremely easy to use the BreadcrumbBar component.

BreadcrumbBar implements the breadcrumb user interface in Swing. To leverage the existing code, it uses the exact *TreeModel* as used in JTree as the model class. The path displayed in the breadcrumb can set or get using *getSelectedPath* and *setSelectedPath*. There is a great advantage of using *TreeModel* as many applications already got *TreeModel* for the things they want. For example, if you have a TreeModel that represents the file system, just set it to *BreadcrumbBar*, you will get the same breadcrumb bar as you can find on Windows 7 or Vista's file explorer.



There are several segments in a *BreadcrumbBar*. In each segment, there are a node component and a node separator.

Usually when you press the node separator, it will show a drop down menu which has the children nodes of the node. But if you don't want to have such a drop down menu, you can disable it using *setDropDownAllowed(false)*.

When the drop down menu is shown, we change the icon from the left arrow to the down arrow. Both icons can be changed using *setNextIcon* and *setDropDownIcon* respectively.

Custom Animation

CustomAnimation defines the basic attributes of an animation. Currently we support three kinds of animations - fly, zoom, and fade - we call it effect. They all can be used during enter and exit animation type.

There are a lot of options you can control how the animation looks like. Those options include speed, smoothness (the number of frames in the animation, the more frame, the smoother), and location. In the case of fly effect animation, you also have options such as direction, start location, end location, function x and function y. Function x and function y here are just like function in mathematics. Both functions together will define the path of the fly in or fly out.

Here is an example. Say the animation will start at x coord 0 and end at x coord 100. To make it simple, let's assume there are 100 steps in this animation. If you use linear functionX, it will move steadily because in the linear function, calculate(step, totalStep) simply returns ((double) step) / totalStep. This means at step 1, move to x coord 1, at step 2, move to x coord 2 and so on. Now if you use pow2 function, it will move differently. FUNC_POW2 function returns Math.pow((double) step / totalStep, 2.0). If you calculate manually, you will find it moves slowly at the beginning and faster at the end. FUNC_POW_HALF function on the other hand will be opposite. It will be faster at the beginning and slower at the end. Please note, this is just the functionX which controls the movement on x coord. If the y coord of start and end location are different, the y coord will change along with x coord change. Combining the both functionX and functionY, it's possible to achieve any moving track as you want from the starting location to end location.

Just to make it interesting, we pre-built FUNC_BOUNCE or FUNC_VIBRATE functions you can use to create a bouncing or vibrating effect.

In the case of fade effect animation, you only have the option of end location as it stays at the same location. However you do have a function to control the fade speed at different stage. For example, if you choose FUNC_POW_HALF, it will fade faster at the beginning and slower at the end. If you choose FUNC_POW2 or FUNC_POW3, it will be the opposite.

Internationalization Support

All of the Strings used in *JIDE Components* are contained in properties files as listed below.

com/jidesoft/swing/swing.properties

COPYRIGHT © 2002-2011 JIDE SOFTWARE, INC. ALL RIGHTS RESERVED

com/jidesoft/plaf/basic/basic.properties

com/jidesoft/status/status.properties

com/jidesoft/document/document.properties

com/jidesoft/pane/pane.properties

Some users contributes localized version of these files and we put those files inside jide-properties.jar. If you want to support languages other than those we provided, just extract those properties files, translated to the language you want, add the correct postfix and then jar them back into jide-properties jar. You are welcome to send the translated properties file back to us if you want to share it.