

JIDE Components Developer Guide

PURPOSE OF THIS DOCUMENT

Welcome to the *JIDE Components*, the foundation product in JIDE Software's product line.

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

WHAT IS JIDE COMPONENTS

Thousands and thousands of valuable development hours are wasted on rebuilding components that have been built elsewhere. Why not let us build those components for you so you can focus on the most value-added part of your application?

So what are those components and how do we choose them?

First of all, they are generic: our components provide solutions for building IDEs (Integrated Development Environments). If you have ever worked to build an IDE or similar product, you will find our components are very familiar to you. Not because you've seen them before but because they are very generic components. Most likely you will say "Hmm, I can use this component in my application".

Secondly, they are extensible: we never assume our components will satisfy all your requirements. So in addition to what we provide, we always leave extension points so that you can write your own code to extend the component. Believe it or not, our whole product strategy is based on the extensibility of each component we are building. We try to cover all the requirements we can find and to build truly general, useful components. At some point, users will likely find a need we didn't address, but that's fine! Our components allow you to "help yourselves".

Last, but not least, they'll save you time. You use a component because you think it will be faster to build on top of it than to start from scratch. If the component is very easy, you probably will build it yourself. If you find a component is too complex and too hard to use, you probably will still build it yourself. With this in mind, we carefully chose what components to include in *JIDE Components*. You may notice that we didn't provide a large number of components. Why? Because we are "picky". Our pickiness means that all our carefully chosen components will definitely save your precious time.

PACKAGES

The table below lists the packages in the *JIDE Components*.

Packages	Description
com.jidesoft.swing	Common components. We expose some of the classes in our documentation. For those not exposed, either they are just used internally or they are not mature enough to be used publicly.

com.jidesoft.document	DocumentPane – tabbed-document interface implementation similar to what you see in Visual Studio .NET IDE
com.jidesoft.status	StatusBar – A generic status bar implementation.
com.jidesoft.pane	CollapsiblePane, FloorTabbedPane and other pane components
com.jidesoft.icon	Icon related classes
com.jidesoft.utils	Utilities classes
com.jidesoft.tipoftheday	Tip of the Day dialog component

A general comment on our naming convention: If the class is modified from or based on an existing Swing/AWT class, we prefix the original Swing/AWT class name with Jide - for example, JideTabbedPane (you can tell that it's based on JtabbedPane from the name). If it's a completely new component that doesn't exist in Swing/AWT then we don't prefix - for example, StatusBar, SidePane.

We will add more and more components to *JIDE Components* in the future and we will keep the same package organization. If the component is complex enough then there will be a separate package for it; if the component is small or needs to be shared by other components then we will put it under com.jidesoft.swing.

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

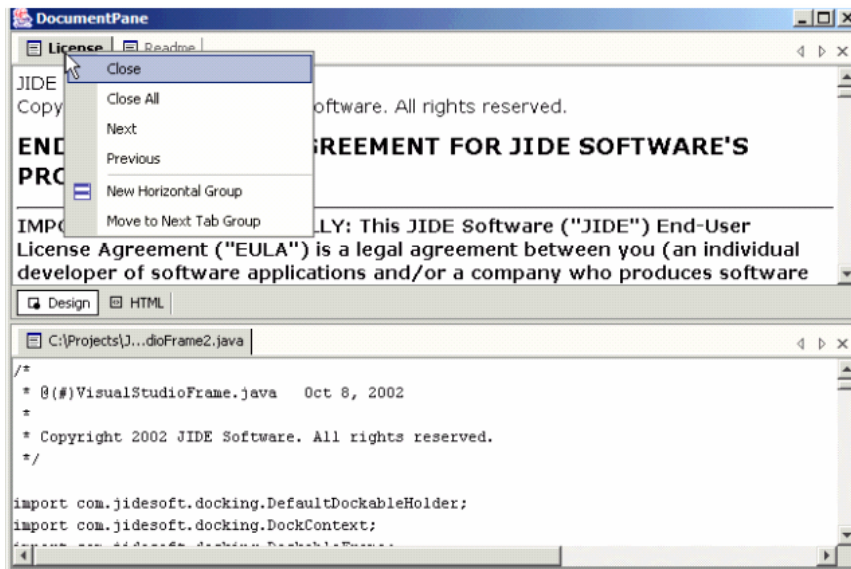
In MDI applications, you can view/edit multiple documents at the same time. You can either use Windows menus or hotkeys (such as Ctrl-Tab) to navigate between those documents. MDI overcomes the major drawbacks of SDI, but has some drawbacks of its own. The two biggest issues are that it wastes screen 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.



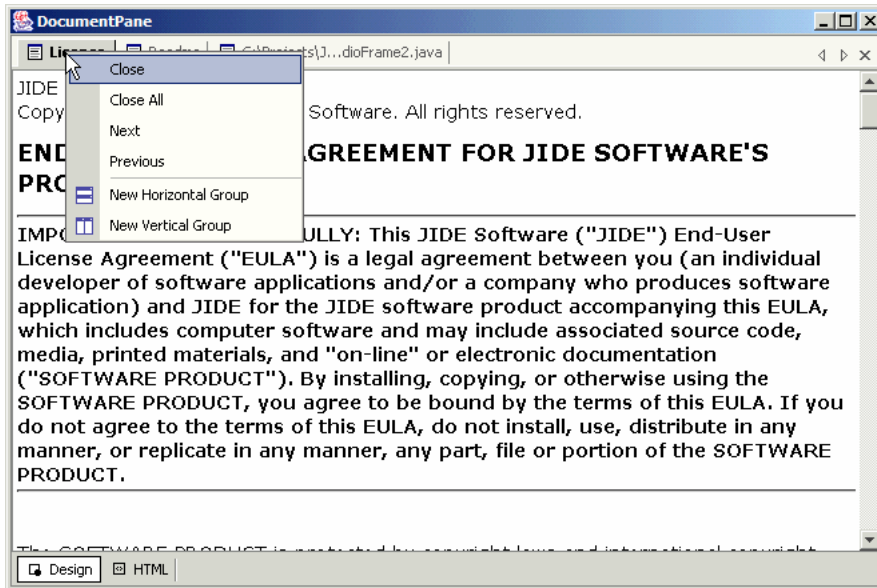
How will my users use it?

All “generic” operations related to a document pane can be done through the document tabs.

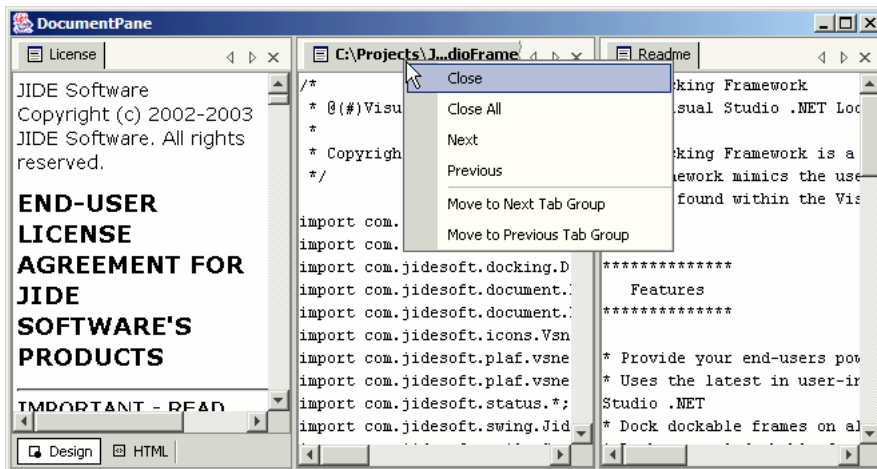
- Clicking on the tab will activate that document.
- Right-clicking on a document tab will popup a context menu. The built-in menu already comes with a lot of built-in functionality and can be extended.

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.

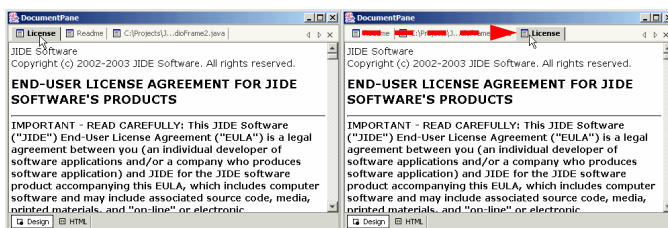


In the next example, there is no menu choice for either “New Horizontal Group” or “New Vertical Group”. Since there is only one document in the current document group, it doesn’t make sense to create another group. However you can move it to another group.

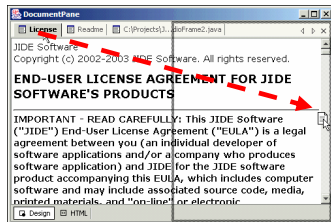


Another, simpler way to move windows and create groups is to use drag and drop.

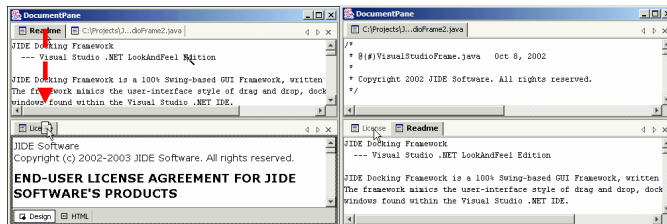
- Drag a document tab horizontally within a tab area and the tab order will change.



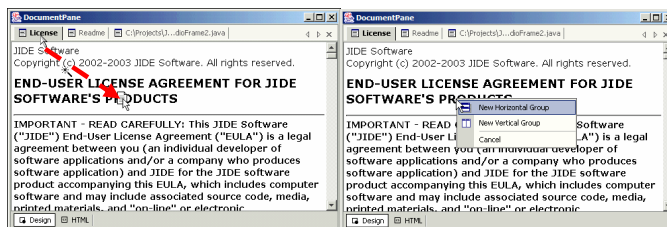
- Dragging a tab and dropping it near the south border or east border will create a new horizontal or vertical tab group, respectively.



- 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.

As a developer, how do I use it?

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 customizer)`.

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

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

By default, you have multiple document groups so that you can 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)`.

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

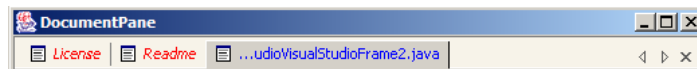
At any time, if you know the name of the document, you can call `getDocument(name)` to get the `DocumentComponent`. If you have a reference to the actual component, calling `getNameOf(component)` will give you the name if the component is opened.

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 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;
}
```



```
}
```

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`

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.

UI Defaults used by DocumentPane

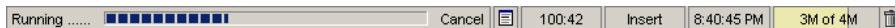
Name	Type	Description
<code>DocumentPane.groupBorder</code>	Border	The border of each document group.
<code>DocumentPane.newHorizontalGroupIcon</code>	Icon	Icon used for action to new a horizontal group
<code>DocumentPane.newVerticalGroupIcon</code>	Icon	Icon used for action to new a vertical group
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

STATUS BAR

A StatusBar is never a central part of an IDE but every IDE has one. 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 the most obvious item. It can display a text message or it can display progress in an optional progress bar. You can activate (display) the progress bar by calling `setProgressStatus("Running")`, and you can set the percentage finished using `setProgress(int)`. In the below status bars, the second one is in running mode.



Method	Description
<code>setStatus(String)</code>	Set the status such as "Ready" in the screen shot above
<code>setProgressStatus(String)</code>	Set the status mode when progress bar is visible, such as "Running" in the screenshot above
<code>setProgress(int)</code>	<code>int</code> 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).
<code>setCancelCallback(CancelCallback)</code>	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 <code>cancelCallback</code> you passed in is null then the Cancel button will not be shown. See interface <code>CancelCallback</code> in the javadoc for more details.

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
<code>setText(String)</code>	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.

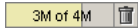


Method	Description
setUpdateInterval(int)	How often to do a refresh, in ms (the default is 500 ms). If you don't display the seconds field, or you just display the date then you can change the interval to a much larger number.
setTextFormat(DateFormat)	Set the DateFormat that formats the text displayed 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.

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	

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 added 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".

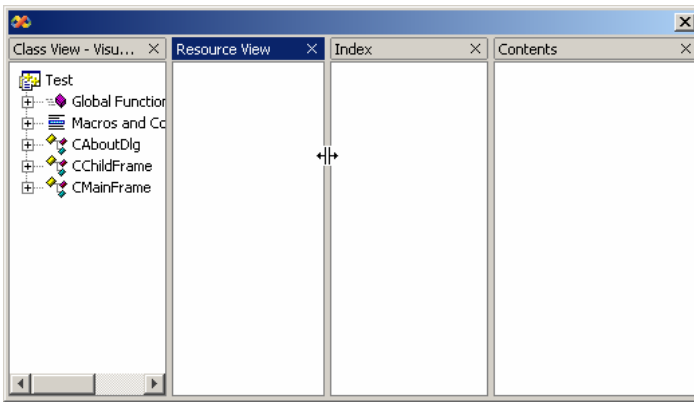
UI Defaults used by StatusBar

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

PANE

JideSplitPane

Although JSplitPane is a useful Swing component, it has one major limitation: it can only split into two windows. If you want to split into three windows, you have to use two JSplitPanes. That may be OK in most cases, but if you want to split it into four or five or more windows then you will quickly get into trouble, maintaining so many JSplitPanes. As you can see in *JIDE Docking Framework*, we need to be able to split a panel into any number of windows¹. JSplitPane obviously cannot meet this need gracefully, so we developed JideSplitPane.



Above is an example of a JideSplitPane, which is split into four parts. Each divider can be moved using the mouse, to resize the components either side of it.

JideSplitPane can split either horizontally or vertically, using the two identifiers defined in JideSplitPane as `HORIZONTAL_SPLIT` and `VERTICAL_SPLIT`. You can either specify the orientation in the constructor or call `setOrientation` after it is constructed.

Call `addPane(...)` or `insertPane(...)` or `add(...)` to add a new component. The underlying layout is `JideBoxLayout`, so you can specify the constraints as `VARY`, `FLEXIBLE` or `FIX` when you call `add(...)`.

By default, the size of the divider is 3 pixels. You can either change this by calling `setDividerSize()`, or you can change it globally in `UIDefaults` using the key `"JideSplitPane.dividerSize"`. You can also change the border and background color of the divider in `UIDefaults` using `"JideSplitPaneDivider.border"` and `"JideSplitPaneDivider.background"`.

Continuous Layout refers to painting during drag and drop actions. If this is set to true, then the child components are continuously redisplayed and laid out while moving a window. The default value of this property is false, meaning that only an outline is displayed, which provides much better performance. You can change this with `setContinuousLayout(boolean)`.

¹ You can refer to a bug in Java website for information on this particular issue.
<http://developer.java.sun.com/developer/bugParade/bugs/4155064.html>

JideTabbedPane

JideTabbedPane is similar to JtabbedPane; the differences are that JideTabbedPane:

- Has an option to hide the tab area if there is only one component in a tabbed pane.
- Has an option to resize the tab width so that all tabs can fit in one row.
- Has an option to show scroll left and scroll right buttons and a close button in the upper right of the tab area.
- Has an option to show a close button on tab
- Only allows TOP and BOTTOM tab placement in the current version.

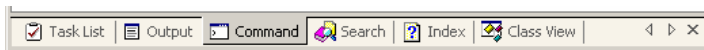
Below are two screen shots both shown with `setShrinkTab(true)` and `setShowTabButtons(false)`.



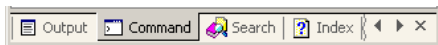
When the total available width shrinks, each tab shrinks. As the tab shrinks, the title on the tab is truncated or completely removed so that only the icon is visible.



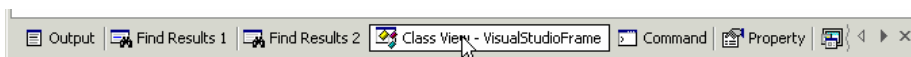
The two screen shots below are shown with `setShrinkTab(false)` and `setShowTabButtons(true)`.



When the available width shrinks, the tab size doesn't shrink. The left and right arrow buttons allow you to scroll to hidden tabs.



You can also specify box style instead of the typical tab style. This is shown below.



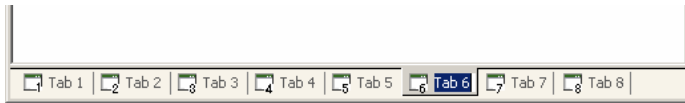
Since the box style TabbedPane looks so different from normal TabbedPane, it's useful as a contrast to the normal TabbedPane when you want to indicate different things. For example, in our implementation of DocumentPane, each document is a normal style TabbedPane (top) possibly containing multiple views of that document. The views are selectable via box style tabs (bottom), as shown in the screenshot overleaf.



You can also put a close icon on the tab. This is very useful especially each tab is a document in *DocumentPane*. To use this option, you need to call the following two calls. If you never call `setShowCloseButtonOnTab`, a default value will be used by reading it from L&F. For example, in VSNET L&F, the value is false. In Eclipse L&F, the value is true. So if you want to set it freely, you must disable the L&F by `setUseDefaultShowCloseButtonOnTab` to false. Then whatever value you set to `setShowCloseButtonOnTab` will be used.

```
tabbedPane.setUseDefaultShowCloseButtonOnTab(false);
tabbedPane.setShowCloseButtonOnTab(true);
```

JideTabbedPane also supports inline tab title editing. By default, this feature is disabled. You need to enable it by calling `setTabEditingAllowed(true)`. If enabled, user can double click on any tab to start editing the title. See below.



Since *JideTabbedPane* extends *JTabbedPane*, the usage of it is exactly the same as *JTabbedPane*, except for the differences in appearance, noted above.

UI Defaults used by JideTabbedPane

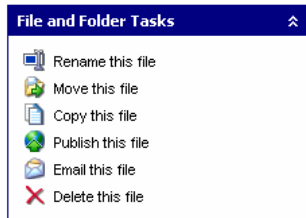
Name	Type	Description
<code>JideTabbedPane.border</code>	Border	The border of tabbed pane
<code>JideTabbedPane.background</code>	Color	The background of tabbed pane
<code>JideTabbedPane.foreground</code>	Color	The foreground of tabbed pane
<code>JideTabbedPane.light</code>	Color	One of the colors used to paint the tab border
<code>JideTabbedPane.highlight</code>	Color	One of the colors used to paint the tab border
<code>JideTabbedPane.shadow</code>	Color	One of the colors used to paint the tab border
<code>JideTabbedPane.darkShadow</code>	Color	One of the colors used to paint the tab border
<code>JideTabbedPane.tabInsets</code>	Insets	The insets of each tab
<code>JideTabbedPane.contentBorderInsets</code>	Insets	The insets of tab content

JideTabbedPane.tabAreaInsets	Insets	The insets of the area where all the tabs are
JideTabbedPane.tabAreaBackground	Insets	The tab area background
JideTabbedPane.font	Font	The font used by tabbed pane
JideTabbedPane.selectedTabFont	Font	The font used to draw the text of the selected tab
JideTabbedPane.unselectedTabTextForeground	Color	The default text color of unselected tabs. If setForegroundAt() is call to set a new color, the new color will be used. The selected tab foreground is whatever color returned from getForegroundAt().
JideTabbedPane.selectedTabBackground	Color	The selected tab background. The unselect tab background is tabAreaBackground
JideTabbedPane.textIconGap	Integer	The gap between icon and text
JideTabbedPane.showIconOnTab	Boolean	Whether to show icon on tabs
JideTabbedPane.showCloseButtonOnTab	Boolean	Whether to show close button on tabs
JideTabbedPane.closeButtonAlignment	Integer	If the close button is on tab, what is the alignment. It could be LEADING or TRAILING defined in SwingConstants.

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.

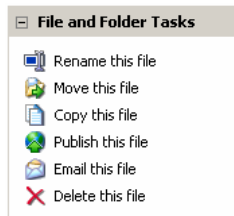
```
CollapsiblePane panel = new CollapsiblePane("File and Folder Tasks");
JPanel labelPanel = new JPanel();
labelPanel.setLayout(new GridLayout(6, 1, 1, 0));
labelPanel.add(createHyperlinkButton("Rename this file",
SampleIconsFactory.getImageIcon(SampleIconsFactory.CollapsiblePane.RENAME)));
labelPanel.add(createHyperlinkButton("Move 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

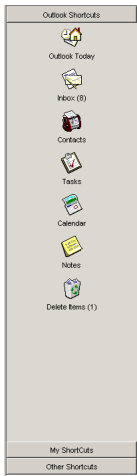
When the content pane is collapsed or expanded a CollapsiblePaneEvent will be fired. There are four types of events, which will be fired at different times. When the content pane begins to expand, COLLAPSIBLE_PANE_EXPANDING is fired. When it finished expanding, COLLAPSIBLE_PANE_EXPANDED will be fired. 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. The default style is DROPDOWN_STYLE, which was shown in the earlier screenshots.



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()*;

FloorTabbedPane



FloorTabbedPane is another type of tabbed pane. A typical tabbed pane has many panels and corresponding tabs. The user can click on a tab to choose which panel to view. Although 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). One famous example of it is the Outlook Bar in the Microsoft Outlook product.

Although FloorTabbedPane extends JtabbedPane, there are no additional public methods on FloorTabbedPane, 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 JButton, and the components will have whatever style was specified when the user created them.

TIPS OF THE DAY

A lot of applications use a “Tips of the Day” dialog to get a new user up and running quickly. Although the content of the Tips of the Day is similar to the online help document, most users prefer to read Tips of the Day because they are concise and they only take a short amount of time per day. We created the TipOfTheDayDialog component to allow more applications to easily create their own Tips of the Day.

To use this component, you will interact with both the TipOfTheDayDialog dialog and the TipOfTheDaySource information source interface. You can implement the TipOfTheDaySource by just providing two method implementations: getNextTip() and getPreviousTip(). Once you have a TipOfTheDaySource you can pass this to a TipOfTheDayDialog to display the tip of the day. You can create any type of TipOfTheDaySource and still use the same TipOfTheDayDialog to display it. We also provide a ResourceBundleTipOfTheDaySource, which implements TipOfTheDaySource (as the name indicates, it reads tips from a ResourceBundle).

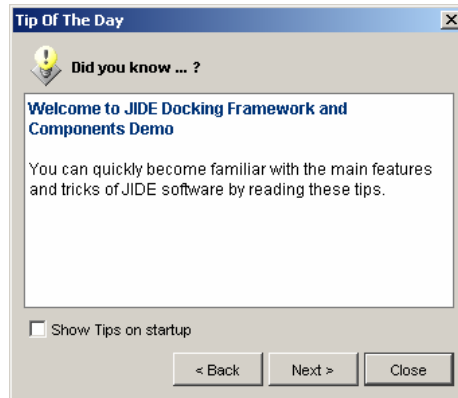


Figure 1 Tip of the Day

The TipOfTheDayDialog is very easy to use. First of all, you need to create a properties file with each tip in a separate line and with zero-based tip numbers, as shown below:

```
0=<b>Welcome to JIDE Docking Framework and Components Demo</b><p>You can quickly ...
1=All the <b>tabs</b> can be drag-n-dropped. Just press mouse button on a tab, hold and ...
.....
```

You can use html in the tips to get a better-looking result. You can also use css to customize the display. Let's say your properties file is tips.properties and your css file is tips.css. Place them at the top level of your class path. You can then write code like following, to display a tip of the day dialog.

```
ResourceBundleTipOfTheDaySource tipOfTheDaySource
    = new ResourceBundleTipOfTheDaySource(ResourceBundle.getBundle("tips"));
tipOfTheDaySource.setCurrentTipIndex(-1);
URL styleSheet = TipOfTheDayDialog.class.getResource("/tips.css");
TipOfTheDayDialog dialog
    = new TipOfTheDayDialog(_frame, tipOfTheDaySource,
        new AbstractAction("Show Tips on startup"){
            public void actionPerformed(ActionEvent e) {
                if(e.getSource() instanceof JCheckBox) {
                    JCheckBox checkBox = (JCheckBox) e.getSource();
                    setPrefBooleanValue("tip", checkBox.isSelected());
                }
            }
        }, styleSheet);
if(dialog.getShowTipCheckBox() != null) {
    dialog.getShowTipCheckBox().setSelected(getPrefBooleanValue("tip", true));
}
dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
dialog.pack();
dialog.setLocation(200, 200);
dialog.show();
```

LAYOUT

JideBoxLayout

As its name indicates, the JideBoxLayout class is similar to Swing's BoxLayout.

Similar to BoxLayout, JideBoxLayout lays components out either vertically or horizontally. Unlike BoxLayout however, there is a constraint associated with each component, set to either FIX, FLEXIBLE, or VARY. If the constraint is set to FIX then the component's width (or height if the JideBoxLayout is vertical) will always be the preferred width. By contrast, although FLEXIBLE components try to keep the preferred width, they will shrink proportionally if there isn't enough space. Finally, VARY components will expand in size to fill whatever width is left. Although you can add multiple FIX or FLEXIBLE components, only one VARY component is allowed.

CODE EXAMPLE 1:

This sample has three buttons; the first one is FIX and the second and third ones are FLEXIBLE.

```
JPanel panel = new JPanel();
panel.setLayout(new JideBoxLayout(panel, 0, 6));

JButton button = new JButton("FIX");
button.setPreferredSize(new Dimension(60, 60));
panel.add(button, JideBoxLayout.FIX);

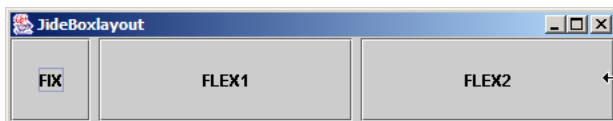
button = new JButton("FLEX1");
button.setPreferredSize(new Dimension(120, 60));
panel.add(button, JideBoxLayout.FLEXIBLE);

button = new JButton("FLEX2");
button.setPreferredSize(new Dimension(120, 60));
panel.add(button, JideBoxLayout.FLEXIBLE);
```

Original:



After resizing:



CODE EXAMPLE 2:

This example has one FIX button, one FLEXIBLE button, and one VARY button.

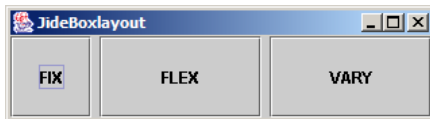
```
JPanel panel = new JPanel();
panel.setLayout(new JideBoxLayout(panel, 0, 6));

JButton button = new JButton("FIX");
button.setPreferredSize(new Dimension(60, 60));
panel.add(button, JideBoxLayout.FIX);

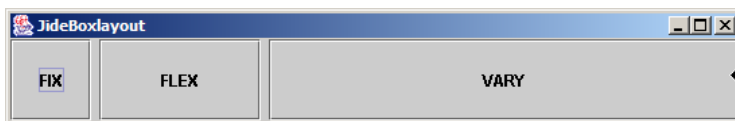
button = new JButton("FLEX");
button.setRequestFocusEnabled(false);
button.setPreferredSize(new Dimension(120, 60));
panel.add(button, JideBoxLayout.FLEXIBLE);

button = new JButton("VARY");
button.setPreferredSize(new Dimension(120, 60));
panel.add(button, JideBoxLayout.VARY);
```

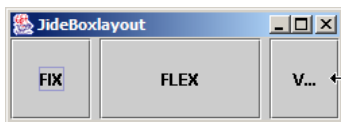
Original:



After resizing:



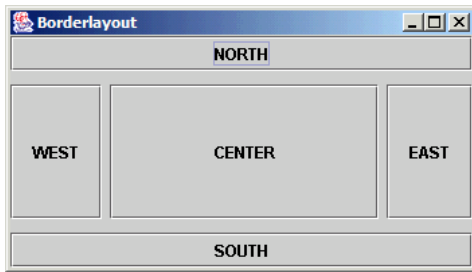
After resizing again:



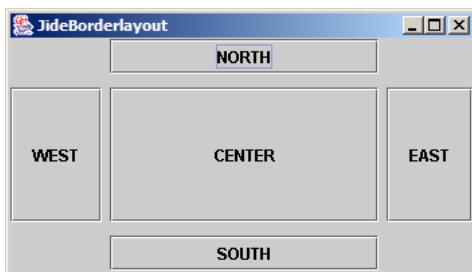
JideBorderLayout

JideBorderLayout is almost the same as the standard Swing BorderLayout except that the NORTH and SOUTH component's width is the same as the CENTER component, as shown overleaf. Please note the different between BorderLayout and JideBorderLayout.

In AWT BorderLayout, the north and south components take *all* of the horizontal space that is available.



By contrast, in JideBorderLayout the north and south components only take the same horizontal space as the center component.



IMAGES AND ICONS RELATED CLASSES

ColorFilter and GrayFilter

A disabled button will normally display a disabled icon. However it's a pain to create two icons for each button. Why not just pass in the normal icon and let the Jide framework create a disabled icon for you?

Image `ColorFilter.createDimmedImage(Image)`

Image `GrayFilter.createDisabledImage(Image)`

This is the effect of above methods

Normal Dimmed Image Disabled Image



IconsFactory

In Java/Swing, you can load an image file as a disk file or as a resource. We found that it's easier and faster to load image files as resources. This class is designed to encourage the use of images and icons as resources

The IconsFactory acts as a cache manager for ImageIcon and has three static methods:

```
public static ImageIcon getImageIcon(Class clazz, String fileName);  
public static ImageIcon getDisabledImageIcon(Class clazz, String fileName);  
public static ImageIcon getBrighterImageIcon (Class clazz, String fileName);
```

Each time you call the method, the icon that is returned will be kept in a cache.

In addition to the points mentioned above, IconsFactory also has a special usage: Applications typically use hundreds of icons and images. Management of these objects can easily get out of control. In addition, you might have issues such as duplicate icons, inconsistent use of icons, difficulty in locating the right icon etc. However with the help of IconsFactory, these issues become much less of a problem.

In the release, there is a class called VsnetIconsFactory.java³, which looks like this:

```
public class VsnetIconsFactory {

    public static class ClassElement {
        public final static String CLASS = "vsnet/msdev_class_class.gif";
        public final static String FIELD = "vsnet/msdev_class_field.gif";
        public final static String FIELD_PROTECTED = "vsnet/msdev_class_field_protected.gif";
        public final static String FIELD_PRIVATE = "vsnet/msdev_class_field_private.gif";
        public final static String METHOD = "vsnet/msdev_class_method.gif";
        public final static String METHOD_PROTECTED = "vsnet/msdev_class_method_protected.gif";
        public final static String METHOD_PRIVATE = "vsnet/msdev_class_method_private.gif";
        public final static String CONSTANT = "vsnet/msdev_class_const.gif";
        public final static String MAP = "vsnet/msdev_class_map.gif";
        public final static String GLOBAL = "vsnet/msdev_class_global.gif";
    }

    .....

    public static void main(String[] argv) {
        IconsFactory.generate(VsnetIconsFactory.class);
    }
}
```

If you follow this pattern to create your own Icons Factory, you will get two benefits:











- The first is the handy display you see below. Looking at the listing of VsnetIconsFactory above, notice that there is a ‘main’ method. Run it and an html file will be generated in the current directory, as shown in the example below. It will have a list of all icons in the factory, organized into different sections as a table. In the table, you can see what the icons look like, what the actual image file names are, and how to use them in the code. Developers should never get lost!

Icons in com.jidesoft.icons.VsnetIconsFactory

Generated by JIDE Icons

1. If you cannot view the images in this page, make sure the file is at the same directory as VsnetIconsFactory.java
2. To get a particular icon in your code, call VsnetIconsFactory.getImageIcon(FULL_CONSTANT_NAME). Replace FULL_CONSTANT_NAME with the actual full constant name as in the table below

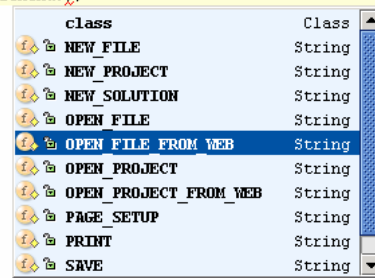
VsnetIconsFactory.ClassElement











Name	Image	File Name	Full Constant Name
CLASS		vsnet/msdev_class_class.gif	VsnetIconsFactory.ClassElement.CLASS
FIELD		vsnet/msdev_class_field.gif	VsnetIconsFactory.ClassElement.FIELD
FIELD_PROTECTED		vsnet/msdev_class_field_protected.gif	VsnetIconsFactory.ClassElement.FIELD_PROTECTED
FIELD_PRIVATE		vsnet/msdev_class_field_private.gif	VsnetIconsFactory.ClassElement.FIELD_PRIVATE
METHOD		vsnet/msdev_class_method.gif	VsnetIconsFactory.ClassElement.METHOD
METHOD_PROTECTED		vsnet/msdev_class_method_protected.gif	VsnetIconsFactory.ClassElement.METHOD_PROTECTED
METHOD_PRIVATE		vsnet/msdev_class_method_private.gif	VsnetIconsFactory.ClassElement.METHOD_PRIVATE
CONSTANT		vsnet/msdev_class_const.gif	VsnetIconsFactory.ClassElement.CONSTANT
MAP		vsnet/msdev_class_map.gif	VsnetIconsFactory.ClassElement.MAP
GLOBAL		vsnet/msdev_class_global.gif	VsnetIconsFactory.ClassElement.GLOBAL

- The second benefit is that with the help of IntelliSense in most Java IDEs, you can easily locate an icon right in your editor. See overleaf for a screenshot from IntelliJ IDEA when using IconsFactory.

³ VsnetIconsFactory is just for tutorial purpose to teach you how to create an IconsFactory. Please do not use any icons from VsnetIconsFactory in your applications because they are copyrighted by Microsoft.

```
VsnetIconsFactory.getImageIcon(VsnetIconsFactory.FileMenu.);
```



class	Class
 NEW_FILE	String
 NEW_PROJECT	String
 NEW_SOLUTION	String
 OPEN_FILE	String
 OPEN_FILE FROM WEB	String
 OPEN_PROJECT	String
 OPEN_PROJECT FROM WEB	String
 PAGE_SETUP	String
 PRINT	String
 SAVE	String

INTERNATIONALIZATION SUPPORT

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

com/jidesoft/plaf/basic/basic.properties

com/jidesoft/plaf/document/document.properties

com/jidesoft/plaf/tipoftheday/TipOfTheDay.properties

Note that we haven't done any localization: if you want to support languages other than English, just extract the properties file, translate it to the language you want, add the correct language postfix and then jar it back into the jide.jar. You are welcome to send the translated properties file back to us if you want to share it!