



InfoNode[®]

Tabbed Panel

Developer's Guide

Revision: 1.3
ITP Version: 1.4.0
Last Modified: 2005-12-04

Copyright © 2005 NNL Technology AB
All rights reserved.

InfoNode® is a registered trademark of NNL Technology AB in Sweden.

Java is a registered trademark of Sun Microsystems, Inc. in the U.S. and other countries.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

The authors assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Contents

1	Introduction	1
1.1	About This Document	1
1.2	Intended Audience	1
1.3	About InfoNode Tabbed Panel	1
1.4	Note about Document Examples	2
1.5	Examples and Demo Applications	2
1.6	Abbreviations	2
1.7	References	3
2	Basic Concepts	4
2.1	Tabbed Panel	4
2.2	Tabbed Panel's Areas	4
2.2.1	Tab Area Orientation	4
2.3	Tab Area Components	5
2.4	Tab Area Layout	5
2.4.1	Layout Policy Scrolling	5
2.4.2	Layout Policy Compression	6
2.4.3	Tab Drop Down List	6
2.5	Tab in a Tabbed Panel	7
2.5.1	Enabled or Disabled	7
2.5.2	Selected	7
2.5.3	Highlighted	7
2.6	Tabs	8
2.7	Titled Tab	8
2.7.1	Layout and Direction	8
2.7.2	Rendering States	9
3	Creating Tabbed Panels and Titled Tabs	10
3.1	Threading	10
3.2	Creating and Using. A Comparison with JTabbedPane	10
3.3	The Tabbed Panel and Tab Event Mechanism	11
4	Configuring a Tabbed Panel	14
4.1	Property Basics	14
4.2	Default Configuration	14
4.3	Properties Composition	14
4.3.1	Button Properties	15
4.3.2	Component Properties	15
4.3.3	Shaped Panel Properties	15
4.4	Tabbed Panel Properties	15
4.4.1	Example: Changing the Tab Area Orientation	16
4.4.2	Example: Changing Tab Area Layout Policy	16
4.4.3	Example: Changing Tab Selection Trigger	16
4.5	Tab Area Properties	16
4.5.1	Example: Changing Background Color	16
4.6	Tab Area Components Properties	16

4.6.1	Example: Setting Border around Tab Area Components	17
4.6.2	Example: Stretching the Tab Area Components	17
4.7	Tabbed Panel Button Properties	17
4.7.1	Example: Changing Button Tool Tip and Button Factory	17
4.8	Tabbed Panel Content Panel Properties	18
4.8.1	Example: Setting Content Area Indentation	18
4.9	Configuring Several Properties	18
4.10	Removing a Property Value	19
4.11	Setting Tab Area Components	20
4.11.1	Example: Setting Two Buttons as Tab Area Components	20
5	Configuring a Titled Tab	21
5.1	Property Basics	21
5.2	Default Configuration	21
5.3	Properties Composition	21
5.3.1	Component Properties	22
5.3.2	Shaped Panel Properties	22
5.4	Titled Tab Properties	23
5.4.1	Example: Changing Size Policy	23
5.5	Titled Tab State Properties	23
5.5.1	State Property Reference	24
5.6	Removing a Property Value	24
5.7	Setting Title Component	24
5.7.1	Example: Set a Button as Title Component for every State	25
5.7.2	Example: Set Different Title Components for the States	25
6	Custom Shapes and Component Painters	26
6.1	Shapeable Areas	26
6.1.1	Shaped Border	26
6.1.2	Component Painters	26
6.2	Example: Creating and using a Shaped Border and a Component Painter	27
7	Themes and Inheriting Properties	30
7.1	Inheriting Properties	30
7.2	Creating and Using Themes	30
7.2.1	Applying a Theme	30
7.2.2	Removing a Theme	31
7.3	Look and Feel Theme	31
8	Mouse Hover	33
8.1	Security	33
8.2	Tabbed Panel and Titled Tab Hovering	33
8.2.1	Example: Setting Hover Listeners	34
8.3	Hover Actions and Changing Properties	35
8.4	Example: Creating a Combined Color/Folding Titled Tab Hover Effect	36
9	Custom Tabbed Panel Content Area	38
9.1	Using a Custom Content Area	38
9.1.1	Custom Content Area Inside a Tabbed Panel	38
9.1.2	No Content Area or Custom Content Area Outside a Tabbed Panel	38
9.2	Example: Selection Bar and Content Area Outside a Tabbed Panel	39
10	Creating Custom Tabs	42
10.1	Tab Container	42

10.1.1 Adding Components	42
10.1.2 Event Components	42
10.2 Example: Creating a Custom Tab	42
11 Properties	45
11.1 Property	45
11.2 PropertyGroup	45
11.3 Typed Properties	45
11.4 PropertyValueHandler	45
12 Property Maps	46
12.1 Property Map Classes	46
12.1.1 PropertyMap	46
12.1.2 PropertyMapGroup	47
12.1.3 PropertyMapValueHandler	47
12.1.4 PropertyMapProperty	47
12.1.5 PropertyMapFactory	47
12.1.6 PropertyMapContainer	47
12.2 Advanced Features	47
12.2.1 Super Maps	47
12.2.2 Property Map Composition	48
12.2.3 Property Value References	49
12.2.4 Listeners	50
12.2.5 Weak Listeners	50
12.2.6 Batch Processing	51
12.2.7 Serialization	51
12.3 ComponentProperties	51

1 Introduction

1.1 About This Document

This document is a developer's guide to ITP, InfoNode Tabbed Panel. It covers the most important concepts found in ITP. For more detailed information about the classes, methods and properties referred to in this document, please see the Javadoc and complete property listing at <http://www.infonode.net/index.html?itpdoc>.

1.2 Intended Audience

The intended audience of this document is Java developers who have some experience with AWT/Swing programming. You should know the basics of Border, Color, Component etc.

1.3 About InfoNode Tabbed Panel

InfoNode Tabbed Panel is a Java Swing based component similar to the JTabbedPane component in Swing. It's released under two licenses, GPL and a commercial license. The GPL version can be used for free by any open source project released under the GPL. The commercial license is for projects that can't use the GPL. See <http://www.infonode.net> for more information about ITP and the licensing.

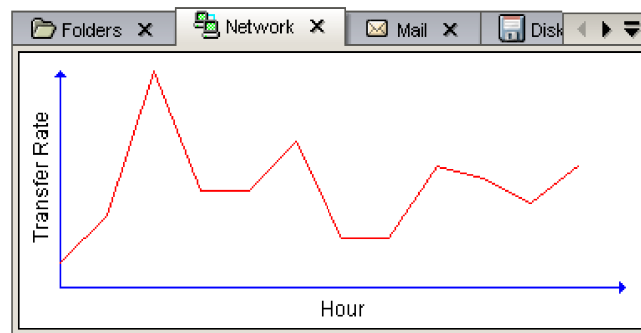


Figure 1.1

InfoNode Tabbed Panel with enabled shadow

A tabbed panel is a graphical user interface component that handles a group of components in a notebook like manor.

Each component is represented by a tab. Only the selected tab's component is shown, the other components are hidden. A component can be selected by clicking the component's tab and then that component is shown and the previously shown component is hidden.

For example, a text editor with multiple open files could let a tab represent an open file and only the selected file can be edited at a time.

One of the main benefits and differences between `JTabbedPane` and ITP is that a tab in ITP is a Swing component. This makes it possible to design different types of tabs and treat each tab as an object instead of as an index as in `JTabbedPane`. A tab can be removed from one ITP and added to another ITP without having to update any indexes or references. It's also possible to create user defined tabs, see chapter 10 *Creating Custom Tabs*.

ITP can also show a shadow, see *Figure 1.1*.

In the code, it's possible to select a tab either by calling `select` on the tab component or by selecting the tab via the tabbed panel. For example, it's possible to have a list with tabs from several tabbed panels and then just call `select` on a tab in the list and that tab will be selected in the tabbed panel that the tab is a member of.

ITP supports an extensive event mechanism where it's possible to receive events when for example a tab has been selected, added, removed, dragged etc. Both the tab and the tabbed panel use the same event mechanism so it's possible to listen to the events from either the ta, the tabbed panel or both.

The behavioural and visual concepts of ITP are configured using an extensive number of properties so that ITP can be tweaked to fulfill most application needs. It is possible to change the visual shape of the titled tabs and the different areas of the tabbed panel, see chapter 6 *Custom Shapes and Component Painters*.

ITP comes with a tab called `TitledTab`. A titled tab supports a text and an icon just as a tab in `JTabbedPane` and it also supports an optional Swing component. This makes it possible to for example add buttons to the tab. Titled tab will most likely fulfill most application needs.

1.4 Note about Document Examples

The document contains several code examples. These examples sometimes use icons and buttons that are called `PyramidIcon`, `CloseButton` and `MinimizeButton`. They are not included in ITP.

1.5 Examples and Demo Applications

The source code for an example application using ITP is included in the ITP distribution. Web Start demos of the example application and a more advanced application can be found at <http://www.infonode.net/index.html?itpdemo>.

1.6 Abbreviations

- ITP – InfoNode Tabbed Panel
- IDW – InfoNode Docking Windows

1.7 References

- [1] *"InfoNode Docking Windows Developer's Guide"*, NNL Technology AB

2 Basic Concepts

This chapter describes the basic concepts of ITP.

2.1 Tabbed Panel

A tabbed panel is a Swing component that can be added to any Swing container in the same way as for example a JPanel.

2.2 Tabbed Panel's Areas

A tabbed panel is divided into two main areas. The first area is the tab area. This is where the tabs and tab area components are displayed, see *Figure 2.1*. The second area is the content area where the tab's content is shown. The content area only shows the content for the selected tab, the other tabs' contents are hidden.

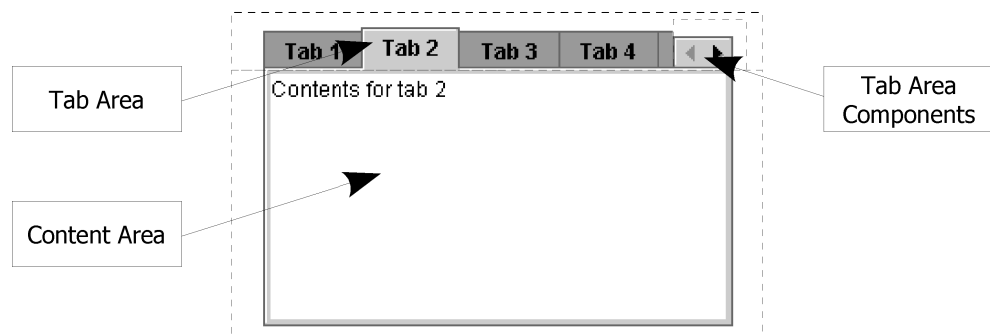


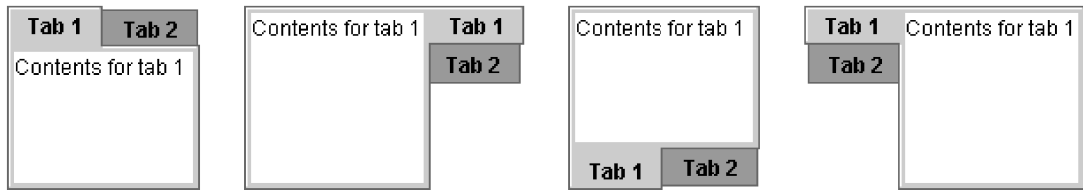
Figure 2.1

A tabbed panel's areas

It's not necessary to have a content area in a tabbed panel, see chapter 9 *Custom Tabbed Panel Content Area*. A tabbed panel without a content area could for example be used as a selection bar.

2.2.1 Tab Area Orientation

The tab area can be placed along any of the sides of the content area, see *Figure 2.2*.

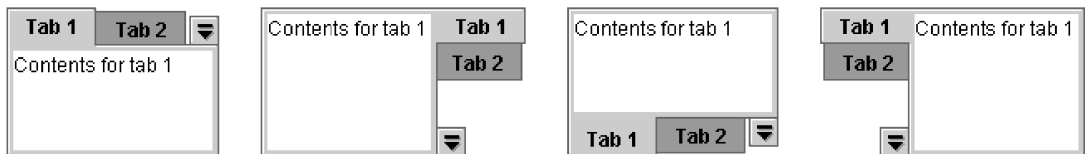
**Figure 2.2**

Tab area orientation: up, right, down and left of the content area

2.3 Tab Area Components

The tab area contains a rectangular area where it's possible to add custom Swing components called tab area components as shown in *Figure 2.1*. This makes it possible to add buttons (or other Swing components) to the tabbed panel.

The tab area components are shown to the right of the tabs if the tab area is placed above (up) or below (down) of the content area and below the tabs if the tab area is placed to the left or to the right of the content area, see *Figure 2.3*.

**Figure 2.3**

Tab area component placement depending on tab area orientation (up, right, down and left of content area)

2.4 Tab Area Layout

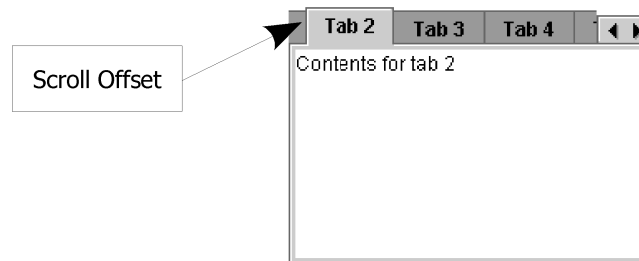
The tabs are always laid out in a single line. If the tabbed panel contains more tabs than what will fit in the tab area, then there are two layout policies to choose from, scrolling or compression.

If the tab area only contains one tab and that tab is wider or higher (depending on tab area placement) than the tab area then that tab will be compressed (if possible) to fit into the tab area.

2.4.1 Layout Policy Scrolling

This is the default layout policy. When not all tabs fit in the tab area it's possible to scroll the line of tabs, see *Figure 2.4*, and scroll buttons will be shown as tab area components. It's also possible to use a mouse with a mouse wheel to scroll the tabs when the mouse pointer is located over a tab or the scroll buttons.

Each scroll step will scroll an entire tab. You will see a bit of the scrolled out tab to the left or above (depending of tab area placement). This so called scroll offset can be changed using properties.

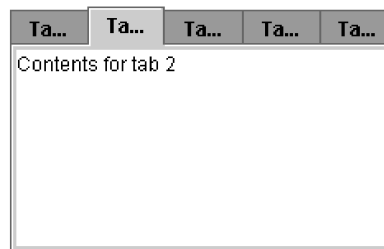
**Figure 2.4**

Tab layout policy: scrolling

When a tab is selected that is not entirely visible you can configure the tabbed panel (using properties) to ensure that the tab becomes visible, i.e. scrolls into the visible area. This is however not the default behaviour.

2.4.2 Layout Policy Compression

This is an alternate layout policy. If not all tabs fit in the tab area the tabs will be downsized (if possible) to fit into the tab area, see *Figure 2.5*. A tab can never be smaller than its minimum size.

**Figure 2.5**

Tab layout policy: compression

2.4.3 Tab Drop Down List

A tab drop down list is a drop down list that will show a text and an icon for each tab. It's useful for making it easier to select a tab when there are many tabs in a tabbed panel, see *Figure 2.6*.



Figure 2.6

Tab drop down list visible

2.5 Tab in a Tabbed Panel

A tab in a tabbed panel can be enabled, disabled, selected and highlighted, see *Figure 2.7*.

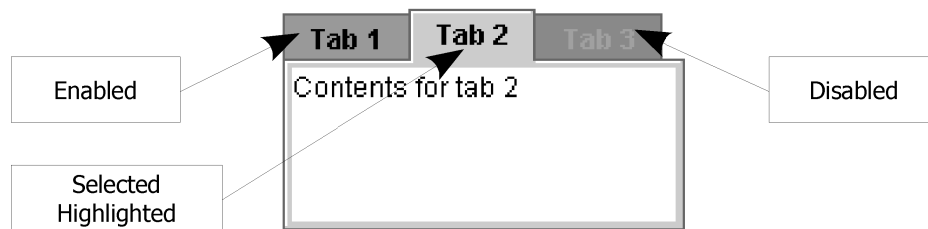


Figure 2.7

Enabled, selected, highlighted and disabled tab

2.5.1 Enabled or Disabled

An enabled tab can be selected. The tab's content will be hidden until the tab is selected. Several tabs can be enabled at the same time in a tabbed panel.

When the tab is disabled it cannot be selected. It's the same as disabled component in Swing. Several tabs can be disabled at the same time in a tabbed panel.

2.5.2 Selected

When a tab is selected, its content will be shown in the tabbed panel's content area. Only one tab in a tabbed panel can be selected at the same time.

2.5.3 Highlighted

A tab can be highlighted to indicate that there's something "special" about a tab. The tab could choose to render its looks in a different way. For example, when a tab is pressed or selected the tabbed panel will also tell the tab to be highlighted i.e. the tab

is selected and highlighted at the same time. It's also possible to tell the tabbed panel to highlight a specific tab.

Only one tab in a tabbed panel can be highlighted at the same time.

2.6 Tabs

A tab is a Swing component, much like a JPanel in Swing, with some additional features. It has an event mechanism that makes it possible to receive events when a tab is selected, highlighted, added etc.. A tab has a reference to the content i.e. the Swing component to be displayed in the content area when the tab is selected. This component is called the tab's content component.

It's possible to create user defined tabs, see chapter 10 *Creating Custom Tabs*, and it's also possible to mix different types of tabs in the same tabbed panel. The tabbed panel comes with a ready to use tab type called titled tab that most likely will fulfill most application needs.

When a tab is added to a tabbed panel it's considered to be a member of that tabbed panel. **A tab can only be a member of one tabbed panel at the same time.**

2.7 Titled Tab

A titled tab is a tab that supports a text and an icon, just like a tab in JTabbedPane, and a custom Swing component called title component that can be used for adding buttons etc. to the tab, see *Figure 2.8*.

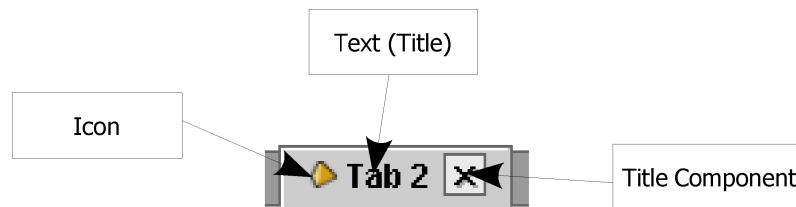


Figure 2.8

A titled tab

2.7.1 Layout and Direction

The text, icon and title component are always laid out in a line. That line can be rotated in four directions, right, down, left and up, see *Figure 2.9*. The text and icon are rotated too but the title component will only be moved. The line can be rotated regardless of the tab area orientation.

If compression is the tab area's layout policy in the tabbed panel that the tab is a member of, then the text and title component part of the tab will be compressed.

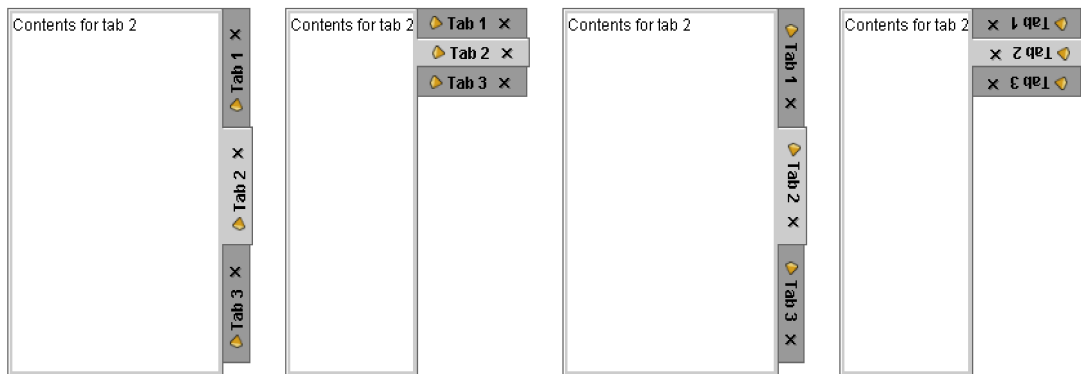


Figure 2.9

Titled tab directions: up, right, down and left. The tab area is oriented to the right of the content area.

2.7.2 Rendering States

A titled tab has three rendering states (i.e. looks), see *Figure 2.10*. These states are:

- **Normal State**
This is the rendering state when the tab is enabled and not selected in a tabbed panel.
- **Highlighted State**
This is the rendering state when the tab is highlighted in a tabbed panel.
- **Disabled State**
This is the rendering state when the tab is disabled in a tabbed panel.

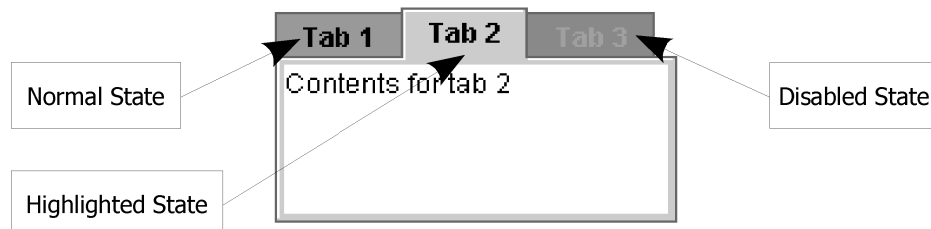


Figure 2.10

Titled tab's rendering states

As you can see the titled tab doesn't have a rendering state when the tab is selected. A selected tab in a tabbed panel will be highlighted when it's selected. This means that the titled tab will render the highlighted state when it's selected and no other tab has been highlighted in the tabbed panel.

3 Creating Tabbed Panels and Titled Tabs

This chapter describes how to use and create `TabbedPanels` and `TitledTabs`. Most of the configuration is performed using properties, see chapter 4 *Configuring a Tabbed Panel* and chapter 5 *Configuring a Titled Tab*.

3.1 Threading

ITP is not thread safe and all constructor and method calls should be made in the AWT event dispatching thread. Use `SwingUtilities.invokeLater()` and `SwingUtilities.invokeAndWait()` when calling ITP from other threads.

3.2 Creating and Using. A Comparison with `JTabbedPane`

The ITP API supports adding, removing and inserting tabs just as in a `JTabbedPane`.

The following examples will demonstrate how to create and use `TitledTab` and `TabbedPanel`. Each example will first show how it can be done with a `JTabbedPane` and then how it can be done with an ITP.

Creating a `JTabbedPane` with three tabs:

```
JTabbedPane tabbedPane = new JTabbedPane();
tabbedPane.addTab("Tab 1", new JTextArea());
tabbedPane.addTab("Tab 2", new JTextArea());
tabbedPane.addTab("Tab 3", new JTextArea());
```

Creating an ITP with three tabs:

```
TabbedPanel tabbedPanel = new TabbedPanel();
TitledTab tab1 = new TitledTab("Tab 1", null, new JTextArea(), null);
TitledTab tab2 = new TitledTab("Tab 2", null, new JTextArea(), null);
TitledTab tab3 = new TitledTab("Tab 3", null, new JTextArea(), null);
tabbedPanel.addTab(tab1);
tabbedPanel.addTab(tab2);
tabbedPanel.addTab(tab3);
```

Let's extend the examples and select "Tab 2".

Selecting "Tab 2" in `JTabbedPane`:

```
tabbedPane.setSelectedIndex(tabbedPane.indexOf("Tab 2"));
```

Selecting "Tab 2" in ITP (there are two ways to do this):

```
// Alternative 1: Selection via the tabbed panel:
tabbedPanel.setSelectedTab(tab2);

// Alternative 2: Selection via the tab
tab2.setSelected(true);
```

Now let's retrieve the selected tab's text and content component (the `JTextArea`).

Retrieving selected tab's title and content component in `JTabbedPane`:

```
int selectedIndex = tabbedPane.getSelectedIndex();
String title = tabbedPane.getTitleAt(selectedIndex);
JComponent contentComponent = (JComponent)tabbedPane.getComponentAt(selectedIndex);
```

Retrieving selected tab's title and content component in ITP:

```
TitledTab selectedTab = (TitledTab)tabbedPanel.getSelectedTab();
String title = selectedTab.getText();
JComponent contentComponent = selectedTab.getContentComponent();
```

Each tab has an index internally in a `TabbedPanel` (just as in a `JTabbedPane`). The first tab has index 0, the next tab has index 1 and so on. In most cases you don't have to worry about a tab's index but in some cases it's necessary. Let's say we want to insert a new tab called "Tab 4" at "Tab 2's" position.

Inserting "Tab 4" at "Tab 2's" position in JTabbedPane:

```
tabbedPane.insertTab("Tab 4", null, new JTextArea(),
    null, tabbedPane.indexOfTab("Tab 2"));
```

Inserting "Tab 4" at "Tab 2's" position in ITP:

```
TitledTab tab4 = new TitledTab("Tab 4", null, new JTextArea(), null);
tabbedPanel.insertTab(tab4, tabbedPanel.getTabIndex(tab2));
```

In the final comparison we will check if "Tab 3" is selected.

Checking if "Tab 3" is selected in JTabbedPane:

```
boolean selected = tabbedPane.getSelectedIndex() == tabbedPane.indexOfTab("Tab 3");
```

Checking if "Tab 3" is selected in ITP (there are at least two ways to do this):

```
// Alternative 1: Via the tabbed panel
boolean selected = tabbedPanel.getSelectedTab() == tab3;

// Alternative 2: Checking the tab itself
boolean selected = tab3.isSelected();
```

3.3 The Tabbed Panel and Tab Event Mechanism

A `TabbedPanel` and a `Tab` share the same event mechanism. It's possible to add a `TabListener` to either a `TabbedPanel` or a `Tab` or to both.

The event mechanism will notify when a `Tab` has been added/inserted, removed, selected, deselected, highlighted, dehighlighted, moved (in the tab area), dragged, dropped or an ongoing drag is aborted.

Creating a tab listener that prints out every event and adding it to a tab and a tabbed panel:

```
TabListener listener = new TabListener() {
    public void tabAdded(TabEvent event) {
        System.out.println("Tab " + event.getTab() +
            " was added to tabbed panel " +
            event.getTab().getTabbedPanel());
    }

    public void tabRemoved(TabRemovedEvent event) {
        System.out.println("Tab " + event.getTab() +
            " was removed from tabbed panel " +
            event.getTabbedPanel());
    }

    public void tabDragged(TabDragEvent event) {
        System.out.println("Tab " + event.getTab() +
            " in tabbed panel " + event.getTab().getTabbedPanel() +
            " was dragged to point " + event.getPoint());
    }
}
```

```
public void tabDropped(TabDragEvent event) {
    System.out.println("Tab " + event.getTab() +
        " in tabbed panel " + event.getTab().getTabbedPanel() +
        " was dropped at point " + event.getPoint());
}

public void tabDragAborted(TabEvent event) {
    System.out.println("Tab " + event.getTab() +
        " in tabbed panel " + event.getTab().getTabbedPanel() +
        " was not dropped");
}

public void tabSelected(TabStateChangedEvent event) {
    System.out.println((event.getTab() == null ?
        "No Tab" :
        "Tab " + event.getTab() +
        " in tabbed panel " + event.getTabbedPanel() +
        " was selected");
    System.out.println((event.getPreviousTab() == null ?
        "No Tab" :
        "Tab " + event.getPreviousTab() +
        " was selected before");
    System.out.println((event.getCurrentTab() == null ?
        "No Tab" :
        "Tab " + event.getCurrentTab() +
        " is now selected");
}

public void tabDeselected(TabStateChangedEvent event) {
    // Deselect tells you which tab that was deselected. The event also contains
    // information about the selected tab in the tabbed panel.

    System.out.println("Tab " + event.getTab() +
        " in tabbed panel " + event.getTabbedPanel() +
        " was deselected");
    System.out.println((event.getPreviousTab() == null ?
        "No Tab" :
        "Tab " + event.getPreviousTab() +
        " was selected before");
    System.out.println((event.getCurrentTab() == null ?
        "No Tab" :
        "Tab " + event.getCurrentTab() +
        " is now selected");
}

public void tabHighlighted(TabStateChangedEvent event) {
    System.out.println((event.getTab() == null ?
        "No Tab" :
        "Tab " + event.getTab() +
        " in tabbed panel " + event.getTabbedPanel() +
        " was highlighted");
    System.out.println((event.getPreviousTab() == null ?
        "No Tab" :
        "Tab " + event.getPreviousTab() +
        " was highlighted before");
    System.out.println((event.getCurrentTab() == null ?
        "No Tab" :
        "Tab " + event.getCurrentTab() +
        " is now highlighted");
}

public void tabDehighlighted(TabStateChangedEvent event) {
    // Dehighlight tells you which tab that was dehighlighted. The event also
    // contains information about the highlighted tab in the tabbed panel.

    System.out.println("Tab " + event.getTab() +
        " in tabbed panel " + event.getTabbedPanel() +
        " was deselected");
    System.out.println((event.getPreviousTab() == null ?
        "No Tab" :
        "Tab " + event.getPreviousTab() +
        " was selected before");
}
```

```
        System.out.println((event.getCurrentTab() == null ?
            "No Tab" :
            "Tab " + event.getCurrentTab() +
            " is now selected");
    }

    public void tabMoved(TabEvent event) {
        System.out.println("Tab " + event.getTab() +
            " in tabbed panel " + event.getTab().getTabbedPanel() +
            " was moved in the tab area");
    }
};

// Add the listener to a tabbed panel
TabbedPanel tabbedPanel = new TabbedPanel();
tabbedPanel.addTabListener(listener);

// Add the listener to a tab
TitledTab tab = new TitledTab("Tab", null, new JTextArea(), null);
tab.addTabListener(listener);
```

4 Configuring a Tabbed Panel

This chapter describes how to configure a `TabbedPanel` and ends with a more complex configuration example. `TabbedPanel` and its properties objects are included in the `net.infonode.tabbedpanel` package.

4.1 Property Basics

A `TabbedPanel` is configured using properties. These properties defines the functional and visual behaviour of the `TabbedPanel`. All properties are fully documented in the Javadoc for the ITP API and only some properties will be described in this document.

All code examples assume that a `TabbedPanel` has already been created.

Creating the tabbed panel used in the chapter examples:

```
TabbedPanel tabbedPanel = new TabbedPanel();
```

ITP uses an advanced properties framework with many flexible features that are not found in Swing. The properties framework is described in chapter *11 Properties*. However, in most cases you only need to use a regular getter and setter.

A `Property` is an object itself. All properties are defined as public static fields. To make it easier, all `TabbedPanel` properties can be set and retrieved by calling the corresponding set and get methods.

When a `Property` is changed, the `TabbedPanel` will automatically update itself. The set method for a `Property` in a properties object will return a reference to the same properties object.

4.2 Default Configuration

A `TabbedPanel` will always have a default configuration. This means that all properties have default values. These values are based on the current look and feel. Some values such as colors, content insets etc are taken from the default values for `JTabbedPane` for the current look and feel.

4.3 Properties Composition

A `TabbedPanel` contains a reference to its properties object that can be retrieved by calling `getProperties()`. This will return a properties object of type `TabbedPanelProperties`.

Retrieving the tabbed panel properties:

```
TabbedPanelProperties properties = tabbedPanel.getProperties();
```

The object, see *Figure 4.1*, is a composition of all the functional properties for a

TabbedPanel and properties objects that controls the visual look of the TabbedPanel areas.

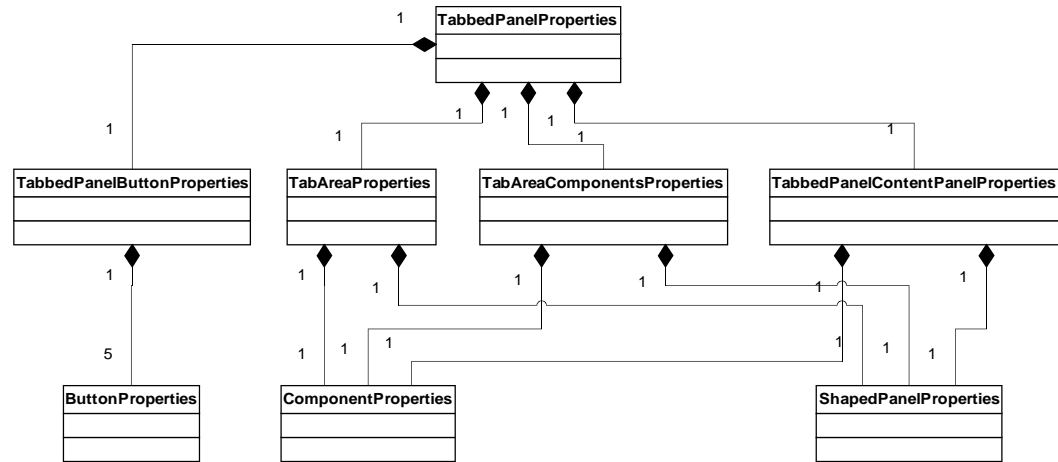


Figure 4.1

TabbedPanelProperties composition

4.3.1 Button Properties

The `ButtonProperties` class contains properties for a button such as icon, disabled icon, tool tip text and a `ButtonFactory` used when creating the button. `ButtonProperties` is part of the `net.infonode.properties.gui.util` package.

4.3.2 Component Properties

The `ComponentProperties` class contains the visual properties that are normally found in a `JComponent`. It contains properties for border, font, background color, foreground color and insets. `ComponentProperties` is part of the `net.infonode.properties.gui.util` package.

4.3.3 Shaped Panel Properties

The `ShapedPanelProperties` contains properties for assigning a component painter, setting a direction and flip horizontal and/or vertical that can be used by the component painter. A component painter is responsible for painting the background of the component. It's called `ShapedPanelProperties` because internally it's used by a so called `ShapedPanel` object. See also chapter 6 *Custom Shapes and Component Painters*.

The `ShapedPanelProperties` contains a property called `DIRECTION` that is **not** used by the `TabbedPanel` or the `TitledTab`.

4.4 Tabbed Panel Properties

The `TabbedPanelProperties` class contains the functional properties of a

TabbedPanel and references to properties objects for each of the TabbedPanel's areas.

4.4.1 Example: Changing the Tab Area Orientation

Changing the tab area orientation i.e. on what side of the content area the tab area should be placed can be accomplished by changing one property.

Configuring tab area to be positioned below (down) the content area:

```
tabbedPanel.getProperties().setTabAreaOrientation(Direction.DOWN);
```

4.4.2 Example: Changing Tab Area Layout Policy

Changing from scrolling (the default tab area layout policy) to compression layout policy is accomplished by changing one property.

Configuring compression as tab area layout policy:

```
tabbedPanel.getProperties().setTabLayoutPolicy(TabLayoutPolicy.COMPRESSION);
```

4.4.3 Example: Changing Tab Selection Trigger

A Tab is by default selected on mouse press. It's possible to change this to mouse release. The advantage of using mouse release is that as long as the mouse hasn't been released, it's possible to abort the selection by either pressing the escape key or the right mouse button.

Configuring a tab to be selected on mouse release:

```
tabbedPanel.getProperties().setTabSelectTrigger(TabSelectTrigger.MOUSE_RELEASE);
```

4.5 Tab Area Properties

The TabAreaProperties class contains all visual properties for the tab area. The property values are stored in a ComponentProperties object.

4.5.1 Example: Changing Background Color

Changing the background color for the tab area is accomplished by changing one property.

Setting red color as background color for tab area:

```
tabbedPanel.getProperties().getTabAreaProperties().getComponentProperties().setBackgroundColor(Color.RED);
```

4.6 Tab Area Components Properties

The TabAreaComponentsProperties class contains properties for the visual behaviour for the rectangular area in the tab area where the tab area components (scroll buttons etc) are shown.

4.6.1 Example: Setting Border around Tab Area Components

Setting a `Border` around the rectangular area containing all the tab area components is accomplished by changing one property.

Setting black line border around the area with the tab area components:

```
tabbedPanel.getProperties().getTabAreaComponentsProperties().  
getComponentProperties().setBorder(new LineBorder(Color.BLACK));
```

4.6.2 Example: Stretching the Tab Area Components

It's possible to stretch the area with the tab area components to be as high as the tab area if the tab area is placed above (up) or below (down) the content area) or as wide as the tab area if the tab area is placed left or right of the content area.

This functionality is not found in a `JComponent` and therefore that property is located in the `TabAreaComponentsProperties` and not in the `ComponentProperties`.

Enabling stretching of the area with the tab area components:

```
tabbedPanel.getProperties().getTabAreaComponentsProperties().  
setStretchEnabled(true);
```

4.7 Tabbed Panel Button Properties

The `TabbedPanelButtonProperties` class contains settings for all the buttons in a tabbed panel, i.e. scroll up/down/left/right and tab drop down list. It's possible to change the `ButtonFactory`, icon, disabled icon and tool tip text for a button. This let you use your own buttons and button icons in the tabbed panel instead of the default buttons and icons.

The default button factories can be found in `TabbedPanelDefaultButtonFactories`.

4.7.1 Example: Changing Button Tool Tip and Button Factory

In this example we will first change the tool tip text for the default scroll up button and then also change the button factory to a custom factory.

Changing tool tip text for scroll up button:

```
tabbedPanel.getProperties().getButtonProperties().getScrollUpButtonProperties().  
setToolTipText("Scroll Up");
```

Changing the scroll up button factory:

```
ButtonFactory scrollUpButtonFactory = new ButtonFactory() {  
    public AbstractButton createButton(Object object) {  
        JButton button = new JButton("Up");  
  
        // You can also add action listeners  
        button.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                System.out.println("Scrolled Up");  
            }  
        });  
  
        return button;  
    }  
}
```

```
tabbedPanel.getProperties().getButtonProperties().getScrollUpButtonProperties().  
    setFactory(scrollUpButtonFactory);
```

4.8 Tabbed Panel Content Panel Properties

The `TabbedPanelContentPanelProperties` class contains all visual properties for the content area. A `TabbedPanel` uses a `TabbedPanelContentPanel` (acts as container for the tabs' content components) as the default content area and the properties are applied to that content panel.

4.8.1 Example: Setting Content Area Indentation

Changing the indentation between the border around the content area and the selected tab's content component is accomplished by changing one property.

Setting insets to 5 pixels between the border around content area and the content component:

```
tabbedPanel.getProperties().getContentPanelProperties().getComponentProperties().  
    setInsets(new Insets(5, 5, 5, 5));
```

4.9 Configuring Several Properties

Now we're ready to create a more complex configuration.

Let's say we want a tabbed panel with the following characteristics:

- Tabs can be reordered using the mouse.
- Tabs are selected on mouse release.
- Possible to deselect a selected tab by clicking on it again.
- The tab area is placed at the bottom (down) of the content area.
- The tab drop down list button should be showing when there are more than one tab in the tabbed panel.
- Black line border around a blue tab area.
- There's no border around the area with the tab area components and that area is stretched and is transparent i.e. same background color as the tab area.
- Black line border around a blue content area.

Calling the `set` method on a property in a properties object will return a reference to the same properties object. This makes it a bit more convenient to set several properties.

Setting the properties:

```
LineBorder blackBorder = new LineBorder(Color.BLACK);  
  
// Changing the properties that are located in the properties root object  
tabbedPanel.getProperties().setTabReorderEnabled(true).  
    setTabSelectTrigger(TabSelectTrigger.MOUSE_RELEASE).  
    setTabDeselectable(true).  
    setTabAreaOrientation(Direction.DOWN).  
    setTabDropDownListVisiblePolicy(TabDropDownListVisiblePolicy.MORE_THAN_ONE_TAB);
```

```
// Changing the properties for the tab area
tabbedPanel.getProperties().getTabAreaProperties().getComponentProperties().
    setBackgroundColor(Color.BLUE).
    setBorder(blackBorder);

// Changing the properties for the area with the tab area components
tabbedPanel.getProperties().getTabAreaComponentsProperties().
    getComponentProperties().
    setBackgroundColor(null). // using null as color means transparent
    setBorder(null);

// Changing the properties for the content area
tabbedPanel.getProperties().getContentPanelProperties().getComponentProperties().
    setBackgroundColor(Color.BLUE).
    setBorder(blackBorder);
```



Figure 4.2

The configured TabbedPanel

Note that the `TitledTabs` have not been changed. They are configured separately, see chapter 5 *Configuring a Titled Tab*.

Each property modification will trigger an update of the `TabbedPanel`. There's a way to modify several properties and then only trigger one update, see section 12.2.6 *Batch Processing*.

4.10 Removing a Property Value

If you have set a `Property` and want to reset to the default value for that `Property`, then you'll have to remove the property's value from the properties storage map.

Setting tab spacing and removing then remove the set value:

```
// Let's say that the default tab spacing is 1
int tabSpacing = tabbedPanel.getProperties().getTabSpacing();

// Now change tab spacing
tabbedPanel.getProperties().setTabSpacing(20);

// The tab spacing is 20
tabSpacing = tabbedPanel.getProperties().getTabSpacing();

// Remove the property value in the storage map
TabbedPanelProperties.TAB_SPACING.removeValue(tabbedPanel.getProperties().getMap());

// Now tab spacing will be 1 again.
tabbedPanel.getProperties().getTabSpacing();
```

4.11 Setting Tab Area Components

It's possible to add custom tab area components to a `TabbedPane`. This makes it possible to for example add additional buttons to a `TabbedPane`. Setting tab area components is not accomplished using properties but by calling `setTabAreaComponents()` in the `TabbedPane`.

These components will show up next to the scroll buttons and the tab drop down list button.

4.11.1 Example: Setting Two Buttons as Tab Area Components

Let's say we want to add two buttons as tab area components. Then we create an array of `JComponents` and call `setTabAreaComponents()` in the `TabbedPane` and they will show up in the tab area, see *Figure 4.3*.

Creating two example buttons and adding them as tab area components:

```
// Note CloseButton and MaximizeButton are just extended JButtons
// used in this example
tabbedPane.setTabAreaComponents(new JComponent[]{new CloseButton(),
                                                  new MaximizeButton()});
```

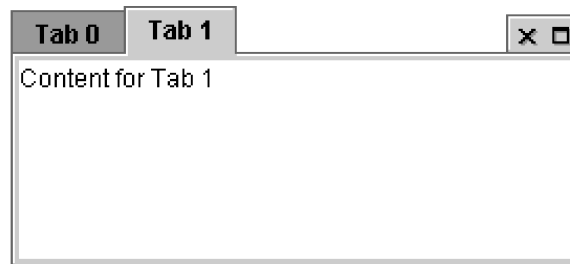


Figure 4.3

TabbedPane with two buttons as tab area components

If you want to remove the tab area components, call `setTabAreaComponents()` with `null` as parameter.

5 Configuring a Titled Tab

This chapter describes how to configure a `TitledTab` and ends with a more complex configuration example. `TitledTab` and its properties objects are included in the `net.infonode.tabbedpanel.titledtab` package.

It's also possible to configure the shape of the tab and how the background is painted, see chapter 6 *Custom Shapes and Component Painters*.

5.1 Property Basics

A `TitledTab` is configured using properties. These properties defines the functional and visual behaviour of the `TitledTab`. All properties are fully documented in the Javadoc for the ITP API and only some properties will be described in this document.

All code examples assume that a `TitledTab` has already been created.

Creating the titled used in the chapter examples:

```
TitledTab tab = new TitledTab("My Titled Tab",
    new PyramidIcon(), // An example icon
    new JTextArea("Content for My Titled Tab",
    new CloseButton()); // An example title component
```

ITP uses an advanced properties framework with many flexible features that are not found in Swing. The properties framework is described in chapter 11 *Properties*. However, in most cases you only need to use a regular getter and setter.

A `Property` is an object itself. All properties are defined as public static fields. To make it easier, all `TitledTab` property values can be set and retrieved by calling the corresponding set and get methods.

When a `Property` is changed, the `TitledTab` will be automatically updated. The set method for a `Property` in a properties object will return a reference to the same properties object.

5.2 Default Configuration

A `TitledTab` will always have a default configuration. This means that all properties have default values. These values are based on the current look and feel. Some values such as colors, fonts etc are taken from the default values for `JTabbedPane` for the current look and feel.

5.3 Properties Composition

A `TitledTab` contains a reference to its properties object that can be retrieved by calling `getProperties()`. This will return a properties object of the type `TitledTabProperties`.

Retrieving the titled tab's properties:

```
TitledTabProperties properties = titledTab.getProperties();
```

The root object, see *Figure 5.1*, is a composition of properties that are common for each rendering state and three separate properties objects, `TitledTabStateProperties`, that controls the visual look for the different titled tab rendering states.

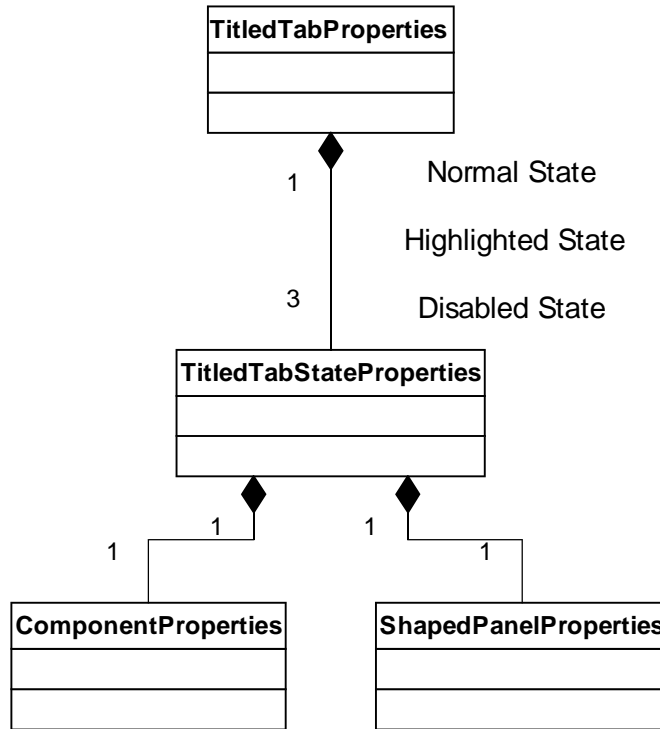


Figure 5.1

TitledTabProperties composition

5.3.1 Component Properties

The `ComponentProperties` class contains the visual properties that are normally found in a `JComponent`. It contains properties for border, font, background color, foreground color and insets. `ComponentProperties` is part of the `net.infonode.properties.gui.util` package.

5.3.2 Shaped Panel Properties

The `ShapedPanelProperties` contains properties for assigning a component painter, setting a direction and flip horizontal and/or vertical that can be used by the component painter. A component painter is responsible for painting the background of the component. It's called `ShapedPanelProperties` because internally it's used by a so called `ShapedPanel` object. See also chapter 6 *Custom Shapes and Component Painters*.

The `ShapedPanelProperties` contains a property called `DIRECTION` that is **not**

used by the `TabbedPanel` or the `TitledTab`.

5.4 Titled Tab Properties

`TitledTabProperties` is the root properties class that contains references to a `TitledTabStateProperties` object for each of the titled tab's rendering state, normal, highlighted and disabled state. It also contains properties that affect all the states.

5.4.1 Example: Changing Size Policy

Size policy is a property that defines what size the tab should have for its different states. If for example the normal state would render the tab smaller than the highlighted state, then as default the tab will still use the maximum size calculated for all its rendering states, see *Figure 5.2*. This can be changed so that each tab state will be rendered with an individual preferred size, see *Figure 5.3*.

Creating a titled tab with an icon only in the highlighted state:

```
TitledTab tab = new TitledTab("My Titled Tab", null, new JTextArea(), null);
tab.getProperties().getHighlightedProperties().setIcon(new PyramidIcon());
```



Figure 5.2

TitledTab with normal, highlighted and disabled state of equal size

Setting individual size policy:

```
tab.getProperties().setSizePolicy(TitledTabSizePolicy.INDIVIDUAL_SIZE);
```



Figure 5.3

TitledTab with normal, highlighted and disabled state of individual size. Note that the tab is smaller in the normal and disabled state than the tab in highlighted state.

5.5 Titled Tab State Properties

`TitledTabStateProperties` is a property class containing all the properties that can be changed for a rendering state. Almost everything can be changed for each state:

- Text
- Icon
- Title component
- Order of the icon, text and title component. For example the icon after the text

- Direction (rotation)
- Component properties such as font, colors, borders etc
- ... and many more

5.5.1 State Property Reference

The highlighted and disabled state properties are linked to the properties from the normal state so that it's unnecessary to set all properties for every state. This means that if the tab's text is set in the normal state, then both the highlighted and disabled state will have the same text if the text is not explicitly set for the highlighted or disabled state, see *Figure 5.4* and *Figure 5.5*.

Creating a titled tab:

```
TitledTab tab = new TitledTab("My Titled Tab", null, new JTextArea(), null);
```



Figure 5.4

The created TitledTab in its normal, highlighted and disabled state

Changing the text for the highlighted state:

```
tab.getProperties().getHighlightedProperties().  
    setText("My Titled Tab Highlighted Text");
```



Figure 5.5

The TitledTab with a different text for the highlighted state

5.6 Removing a Property Value

If you have set a `Property` and want to reset to the default value for that `Property`, then you'll have to remove the property's value from the properties storage map. This is accomplished in the same way as for a `TabbedPanel`, see section 4.10 *Removing a Property Value*.

5.7 Setting Title Component

In Swing, a `JComponent` should only be added to one container. A `TitledTab` supports a title component for each of the rendering states and the same title component can be used for more than one state. The `TitledTab` will handle this internally.

Setting a title component is accomplished by calling the `set` function for the title component for the rendering state in `TitledTab`. The title components for the states are not linked as the `TitledTabStateProperties` are. There's however a

convenience method that sets the same title component for all the states at once.

5.7.1 Example: Set a Button as Title Component for every State

In this example we will set a button as title component for every rendering state, see *Figure 5.6*.

Setting a title component for all states either via the constructor or the `setTitleComponent()` method :

```
// Via constructor
TitledTab tab = new TitledTab("My Titled Tab",
                             null,
                             new JTextArea(),
                             new CloseButton());

// Via set method
tab.setTitleComponent(new CloseButton());
```



Figure 5.6

The same close button as title component for all the TitledTab's rendering state

5.7.2 Example: Set Different Title Components for the States

In this example we will set a minimize button as title component for the normal state, a close button as title component for the highlighted state and no title component for the disabled state, see *Figure 5.7*.

Setting different title components for the states:

```
TitledTab tab = new TitledTab("My Titled Tab", null, new JTextArea(), null);
tab.setNormalStateTitleComponent(new MinimizeButton());
tab.setHighlightedStateTitleComponent(new CloseButton());
tab.setDisabledStateTitleComponent(null);
```



Figure 5.7

Minimize button as title component for the normal state, close button as title component for the highlighted state and no title component for the disabled state

6 Custom Shapes and Component Painters

This chapter describes how to change the look and visual shape of a tabbed panel and a titled tab using shaped borders and component painters.

6.1 Shapeable Areas

You can change the visual shape of the tabbed panels's tab area, tab area components area and content area. You can also change the shape for a titled tab. Each of the areas can have individual shapes and custom backgrounds. Changing a shape is accomplished by the use of a shaped border.

A `Tab` has a `getShape()` method. `TitledTab` will return the active rendering state's `Shape` when that method is called. The method can for example be called by content borders that wish to calculate where a shaped tab tangents the content area.

6.1.1 Shaped Border

Normally, all the shapeable areas are rectangular. Using a `ShapedBorder` makes it possible to assign a user defined visual shape to the areas. The area will still be rectangular but look like it has another shape.

A `ShapedBorder` (found in `net.infonode.gui.shaped.border` package) is an interface that is an extension of the `Swing Border` interface. It adds support for a `java.awt.Shape` via the `getShape()` method.

`ShapedBorders` are used in the same way as regular `Borders`. You can assign a `ShapedBorder` to the border property in the `ComponentProperties` for the area you wish to shape.

The `Shape` returned by `getShape()` will be used for clipping the area i.e. the area will only show a background within its `Shape` and it will be transparent on the outside of the `Shape`. All `TitledTab` event handling will only be performed when the mouse is on the inside of the shape.

Note that if you put a `ShapedBorder` inside a `CompoundBorder`, the outer most `ShapedBorder`'s `Shape` will be used.

6.1.2 Component Painters

It's possible to use component painters (found in the `net.infonode.gui.componentpainter` package) for painting the shapeable areas. This makes it possible to have custom backgrounds such as gradients etc the tabbed panel's tab area, tab area components area, content area and the titled tab.

The painted background area is always clipped to the `Shape` so the `ComponentPainter` does not have to worry about irregular shapes, it can paint the component as if it was rectangular.

The `ComponentPainter` is set in the `ShapedPanelProperties` for the area you

want to paint. If no painter is specified, the area will be painted with the background color specified in the `ComponentProperties`. The area will be transparent if both background color and component painter are set to null.

6.2 Example: Creating and using a Shaped Border and a Component Painter

In this example we will create a `TabbedPanel` where the `TitledTabs` and the tab area components area look like “houses”.

First we need to create a `ShapedBorder` that looks like a “house”.

Creating a Shaped Border:

```
public class HouseShapedBorder implements ShapedBorder {
    // Height of the roof
    private static final int TOP_INSET = 10;

    private Color color;

    public HouseShapedBorder(Color color) {
        this.color = color;
    }

    public Shape getShape(Component c, int x, int y, int width, int height) {
        int xpoints[] = new int[5];
        int ypoints[] = new int[5];

        int index = 0;
        xpoints[index] = x;
        ypoints[index++] = y + height;

        xpoints[index] = x;
        ypoints[index++] = y + TOP_INSET;

        xpoints[index] = (x + width) / 2;
        ypoints[index++] = y;

        xpoints[index] = x + width - 1;
        ypoints[index++] = y + TOP_INSET;

        xpoints[index] = x + width - 1;
        ypoints[index++] = y + height;

        return new Polygon(xpoints, ypoints, 5);
    }

    public boolean isBorderOpaque() {
        return false;
    }

    public void paintBorder(Component c, Graphics g,
        int x, int y, int width, int height) {
        g.setColor(color);

        // Get the shape we are going to paint
        Polygon p = (Polygon) getShape(c, x, y, width, height);

        // We use drawPolyline so that no line is drawn between
        // the first and the last coordinate
        g.drawPolyline(p.xpoints, p.ypoints, p.npoints);
    }

    public Insets getBorderInsets(Component c) {
        return new Insets(TOP_INSET, 1, 0, 1);
    }
}
```

We want to make the roof of the house in one color and the rest of the house in white color. This is accomplished by creating a `ComponentPainter`.

Creating a Component Painter:

```
public class HouseComponentPainter implements ComponentPainter {
    private static final int TOP_INSET = 10;

    private Color roofColor;

    public HouseComponentPainter(Color roofColor) {
        this.roofColor = roofColor;
    }

    public void paint(Component component, Graphics g, int x, int y,
        int width, int height) {
        g.setColor(roofColor);
        g.fillRect(x, y, width, TOP_INSET + 1);

        g.setColor(Color.WHITE);
        g.fillRect(x, y + TOP_INSET + 1, width, height - TOP_INSET - 1);
    }

    public void paint(Component component, Graphics g,
        int x, int y, int width, int height,
        Direction direction,
        boolean horizontalFlip, boolean verticalFlip) {
        paint(component, g, x, y, width, height);
    }

    public Color getColor(Component component) {
        return null;
    }

    public boolean isOpaque(Component component) {
        return false;
    }
};
```

Now we create a `TabbedPanel` and add a few `TitledTabs` with the `HouseShapedBorder` and the roof color of `HouseComponentPainter` in black color for the normal state and red color for the highlighted state. The tab area components area will also have a `HouseShapedBorder` and a `HouseComponentPainter` but in blue color, see final result in *Figure 6.1*.

Creating a Tabbed Panel with Titled Tabs and using House Shaped Border:

```
TabbedPanel tabbedPanel = new TabbedPanel();

HouseShapedBorder normalBorder = new HouseShapedBorder(Color.BLACK);
HouseShapedBorder highlightBorder = new HouseShapedBorder(Color.RED);
HouseShapedBorder tabAreaComponentsBorder = new HouseShapedBorder(Color.BLUE);

HouseComponentPainter normalPainter = new HouseComponentPainter(Color.BLACK);
HouseComponentPainter highlightPainter = new HouseComponentPainter(Color.RED);
HouseComponentPainter tabAreaComponentsPainter = new HouseComponentPainter
(Color.BLUE);

for (int i = 0; i < 8; i++) {
    TitledTab tab = new TitledTab("Tab " + i, null,
        new JTextArea("Contents for Tab " + i), null);

    // Set the borders
    tab.getProperties().getNormalProperties().getComponentProperties().
        setBorder(normalBorder);
    tab.getProperties().getHighlightedProperties().getComponentProperties().
        setBorder(highlightBorder);

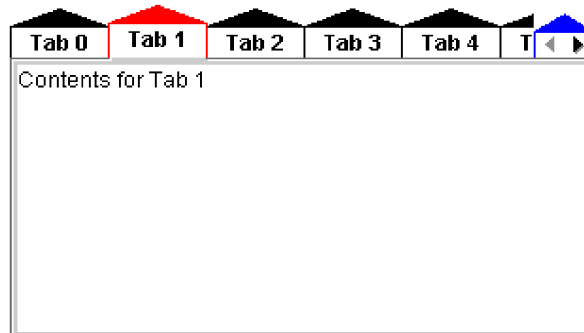
    // Set the painters
    tab.getProperties().getNormalProperties().getShapedPanelProperties().
        setComponentPainter(normalPainter);
```

```
tab.getProperties().getHighlightedProperties().getShapedPanelProperties().
    setComponentPainter(highlightPainter);

tabbedPanel.addTab(tab);
}

// Set tab area components area border
tabbedPanel.getProperties().getTabAreaComponentsProperties().
    getComponentProperties().setBorder(tabAreaComponentsBorder);

// Set tab area components area component painter
tabbedPanel.getProperties().getTabAreaComponentsProperties().
    getShapedPanelProperties().setComponentPainter(tabAreaComponentsPainter);
```

**Figure 6.1**

Tabbed Panel with Shaped Borders and Component Painters

7 Themes and Inheriting Properties

This chapter describes how to share properties object and how to create and use themes. The themes are part of the `net.infonode.tabbedpanel.theme` package.

7.1 Inheriting Properties

It's possible to “share” properties between several `TabbedPanels` or `TitledTabs`. This is accomplished by property inheritance. If you want several `TabbedPanels` with the same configuration it's easier to create a new properties object that is added as a so called super object to the `TabbedPanels`' properties objects, see chapter 12 *Property Maps* and section 12.2.1 *Super Maps*. Changing a property in the shared super object automatically changes all the `TabbedPanels`.

7.2 Creating and Using Themes

A theme in ITP is based on property inheritance and is a composition of a `TabbedPanelProperties` object, a `TitledTabProperties` object and a theme name.

Themes are created by extending the abstract `TabbedPanelTitledTabTheme` class.

Using a theme is just a matter of adding its super object to the existing `TabbedPanelProperties` object and its super object to the `TitledTabProperties` object for every titled tab.

ITP comes with a few themes and we will demonstrate how to apply the `GradientTheme` to a `TabbedPanel` with a few `TitledTabs`.

7.2.1 Applying a Theme

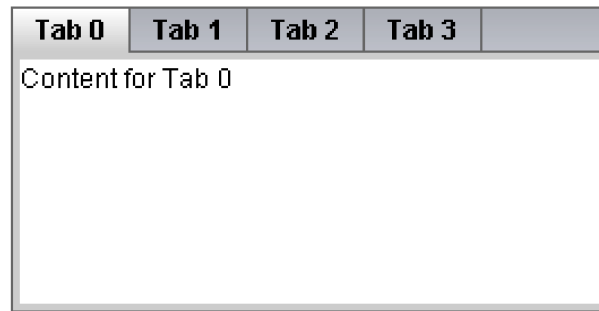
We will now apply the `GradientTheme` to a `TabbedPanel` with `TitledTabs`, see *Figure 7.1*.

Creating and applying a Gradient Theme:

```
// Creating a Gradient Theme with opaque tab area and no shadow
GradientTheme gradientTheme = new GradientTheme(true, false);
TabbedPanel tabbedPanel = new TabbedPanel();

// Apply the theme properties to the tabbed panel
tabbedPanel.getProperties().
    addSuperObject(gradientTheme.getTabbedPanelProperties());

for (int i = 0; i < 4; i++) {
    TitledTab tab = new TitledTab("Tab " + i, null,
        new JTextArea("Content for Tab " + i), null);
    // Apply the theme properties to the titled tab
    tab.getProperties().addSuperObject(gradientTheme.getTitledTabProperties());
    tabbedPanel.addTab(tab);
}
```

**Figure 7.1**

GradientTheme applied to TabbedPanel and TitledTabs

You will have to apply the theme to every `TitledTab` you add to a `TabbedPanel` if you want the same theme for all the `TitledTabs`. It's possible to mix different themes for the `TitledTabs` in the same `TabbedPanel`.

If you want to apply a new theme you should first remove the previously applied themes.

7.2.2 Removing a Theme

A `TabbedPanel` and a `TitledTab` always have a default configuration based on the current look and feel. The default configuration is however not a theme so removing a theme from a themeless `TabbedPanel` or `TitledTab` is an undefined behaviour. So either we need to use a flag or store a reference to the current theme so that we only remove a theme if a theme has been added.

Instead of “knowing” that a theme has been added it's possible to always add the `DefaultTheme` that does nothing but contain empty `TabbedPanelProperties` object and `TitledTabProperties` object so that they keep their default look.

Now we remove the `GradientTheme` we added in the previous example.

Removing a theme from tabbed panel with titled tabs:

```
// Remove theme properties from the tabbed panel
tabbedPanel.getProperties()
    .removeSuperObject(gradientTheme.getTabbedPanelProperties());

// Iterate over every titled tab and remove theme properties
for (int i = 0; i < tabbedPanel.getTabCount(); i++)
    ((TitledTab)tabbedPanel.getTabAt(i)).getProperties()
        .removeSuperObject(gradientTheme.getTitledTabProperties());
```

7.3 Look and Feel Theme

The Look and Feel Theme is an **experimental** theme that tries to replicate the Swing `JTabbedPane`'s look under the active look and feel. This may or may not work depending on the look and feel that is being used. It is wise to test if the theme works well together with the look and feel that is going to be used in the application.

The theme tries to replicate the look for the content area and the tabs, including focus frame and hover effects. The buttons such as scroll buttons are not replicated.

The theme uses heavyweight components internally and therefore the theme must be disposed when it is no longer needed.

Disposing the Look and Feel Theme:

```
// First remove look and feel theme properties from the tabbed panel
tabbedPanel.getProperties()
    .removeSuperObject(lookAndFeelTheme.getTabbedPanelProperties());

// Iterate over every titled tab and remove theme properties
for (int i = 0; i < tabbedPanel.getTabCount(); i++)
    ((TitledTab)tabbedPanel.getTabAt(i)).getProperties()
        .removeSuperObject(lookAndFeelTheme.getTitledTabProperties());

// Dispose the look and feel theme
lookAndFeelTheme.dispose();
```

The theme uses the hover mechanism for tab mouse hover effects, see chapter 8 *Mouse Hover*. If the security permission isn't granted the hover effects for the tabs will be disabled but the theme can still be used.

There's no support given for this theme. The theme may change, be removed etc in future versions of InfoNode Tabbed Panel. It's important that you test that the theme is working with the look and feel you intend to use. If the look and feel for example doesn't render its `JTabbedPane` tabs correctly the same problem will be seen using this theme.

8 Mouse Hover

Both tabbed panel and titled tab support mouse hovering. This makes it possible to create visual effects when the user hovers the mouse over the tabbed panel or the titled tab.

8.1 Security

The hover mechanism listens to `AWTEvents` and therefore it needs the permission `AWTPermission("listenToAllAWTEvents")`. This means that if there's a security manager (for example when running web start), you must grant permission otherwise the hover mechanism will not work. If permission is not granted, hovering will be disabled (without signalling any error messages).

There is a convenience method in `TabbedUtils` called `isHoverEnabled()` that can be used to check if hover is enabled or not.

8.2 Tabbed Panel and Titled Tab Hovering

A hoverable area is an area that can be hovered by the mouse. When the mouse enters the area, a `net.infonode.gui.hover HoverListener` is called and when the mouse exits the area the `HoverListener` is called again. This is different from just adding a `MouseListener` to a `Component` because when a `Component` has a child with a `MouseListener`, the component will trigger `mouseExited(...)` in the `MouseListener` when the mouse hovers the child even if the mouse is still inside the `Component`.

The tabbed panel supports nested hovering. This means that several of the tabbed panel's areas can have a hover listener and be hovered. You can assign a `HoverListener` for the entire tabbed panel, for the tab area, for the tab area components area and for the content area. The `HoverListeners` are called outside in when the mouse enters and inside out when the mouse exits.

You can change the tabbed panel's hover policy by changing `TabbedPanelHoverPolicy` (see the API documentaion) in the properties for the tabbed panel. This gives you the ability to decide if for example a tabbed panel with a child tabbed panel should be considered hovered or not if the child tabbed panel is also hovered.

Titled tab supports hovering. A titled tab is hovered when the mouse is inside it's visual shape.

The hovering order when the mouse enters a tabbed panel:

1. The tabbed panel itself
2. The tab area or the content area depending on where the mouse is located
3. The tab area components area if the mouse is over that area

4. Titled tab if mouse is over a titled tab

The `HoverListener` contains two methods, `mouseEntered(HoverEvent event)` and `mouseExited(HoverEvent event)`. The `HoverEvent` contains information about the hovered component. When a tabbed panel or any of its areas are hovered, the `HoverEvent`'s source will always be the hovered `TabbedPanel`. When a titled tab is hovered, the `HoverEvent`'s source will be the `TitledTab`.

8.2.1 Example: Setting Hover Listeners

The hover listeners are configured in the properties objects.

Setting all hover listeners for a tabbed panel:

```
// Hover listener for entire tabbed panel
tabbedPanel.getProperties().setHoverListener(
    new HoverListener() {
        public void mouseEntered(HoverEvent event) {
            System.out.println("Enter Tabbed Panel");
        }

        public void mouseExited(HoverEvent event) {
            System.out.println("Exit Tabbed Panel");
        }
    });

// Hover listener for tab area
tabbedPanel.getProperties().getTabAreaProperties().setHoverListener(
    new HoverListener() {
        public void mouseEntered(HoverEvent event) {
            System.out.println("Enter Tab Area");
        }

        public void mouseExited(HoverEvent event) {
            System.out.println("Exit Tab Area");
        }
    });

// Hover listener for tab area components area
tabbedPanel.getProperties().getTabAreaComponentsProperties().setHoverListener(
    new HoverListener() {
        public void mouseEntered(HoverEvent event) {
            System.out.println("Enter Tab Area Components Area");
        }

        public void mouseExited(HoverEvent event) {
            System.out.println("Exit Tab Area Components Area");
        }
    });

// Hover listener for content area
tabbedPanel.getProperties().getContentPanelProperties().setHoverListener(
    new HoverListener() {
        public void mouseEntered(HoverEvent event) {
            System.out.println("Enter Content Area");
        }

        public void mouseExited(HoverEvent event) {
            System.out.println("Exit Content Area");
        }
    });
```

Setting hover listener for a titled tab:

```
tab.getProperties().setHoverListener(
    new HoverListener() {
        public void mouseEntered(HoverEvent event) {
            System.out.println("Enter Titled Tab " +
                ((TitledTab)event.getSource()).getText());
        }
    });
```

```
}  
    public void mouseExited(HoverEvent event) {  
        System.out.println("Exit Titled Tab " +  
            ((TitledTab)event.getSource()).getText());  
    }  
};
```

8.3 Hover Actions and Changing Properties

In order to simplify the creation of visual hover effects, the `net.infonode.tabbedpanel.hover` package contains hover actions that make it easy to change properties when a tabbed panel or a titled tab is hovered.

An action is set as a `HoverListener`. When the mouse hovers the area, the action's properties object is added as super object to the hovered area's properties and then removed when the area is no longer hovered.

The following actions are included:

- `TabbedPanelHoverAction` – To be set as a `HoverListener` for a `TabbedPanel` or any of its areas. Contains a `TabbedPanelProperties` object that is added as super object to the `TabbedPanelProperties` for the `TabbedPanel`.
- `TabbedPanelTitledTabHoverAction` – Same as `TabbedPanelHoverAction` but it also contains a `TitledTabProperties` object that will be added as super object to the `TitledTabProperties` for all the `TitledTabs` in the `TabbedPanel`.
- `TitledTabHoverAction` - To be set as a `HoverListener` for a `TitledTab`. Contains a `TitledTabProperties` object that is added as super object to the `TitledTabProperties` for the `TitledTab`.
- `TitledTabTabbedPanelHoverAction` – This action is similar to the `TabbedPanelTitledTabHoverAction`, i.e. it contains both a `TabbedPanelProperties` object and a `TitledTabProperties` object. It is to be set as a `HoverListener` for a `TitledTab`. It will then add the `TabbedPanelProperties` object as super object to the `TabbedPanel` that the `TitledTab` is a member of and the `TitledTabProperties` object as super object to the `TitledTabProperties` for the `TitledTab`.
- `TitledTabDelayedMouseExitHoverAction` – This action wraps another action and the mouse exit will be delayed a specified delay after the mouse has exited the `TitledTab`.

It's possible to combine more than one action using a `net.infonode.gui.hover.CompoundHoverListener` that combines two `HoverListeners`.

Note: The above actions will automatically handle adding/removing super objects when titled tabs are added to or removed from a tabbed panel i.e. there's no need for any special code to deal with this.

8.4 Example: Creating a Combined Color/Folding Titled Tab Hover Effect

In this example we will create a hover effect that changes the titled tab's background to blue when it's hovered. The tabs should be folded i.e. not display their text. When a tab is hovered, the text should be displayed and then shown for another 3 seconds when the mouse has exited the tab.

Creating a combined titled tab hover effect:

```
TabbedPanel tabbedPanel = new TabbedPanel();

// Create a shared properties object for all the titled tabs
TitledTabProperties sharedProperties = new TitledTabProperties();

// Hide the tab text
sharedProperties.getNormalProperties().setTextVisible(false);

// Create a hover action that sets the background to blue when tab is hovered
TitledTabHoverAction colorAction = new TitledTabHoverAction();
colorAction.getTitledTabProperties().getNormalProperties()
    .getComponentProperties().setBackgroundColor(Color.BLUE);

// Create a hover action that shows the text when tab is hovered
TitledTabHoverAction foldingAction = new TitledTabHoverAction();
foldingAction.getTitledTabProperties().getNormalProperties().setTextVisible(true);

// Create a 3-second delayed exit action for the folding action
TitledTabDelayedMouseExitHoverAction delayedExitAction =
    new TitledTabDelayedMouseExitHoverAction(3000, foldingAction);

// Set the compound hover action as hover listener
sharedProperties.setHoverListener(new CompoundHoverListener(
    colorAction, delayedExitAction));

// Create a few tabs and add them to the tabbed panel
for (int i = 0; i < 4; i++) {
    TitledTab tab = new TitledTab("Tab " + i, null, new JLabel("Tab " + i), null);

    // Add the shared properties as super object
    tab.getProperties().addSuperObject(sharedProperties);

    tabbedPanel.addTab(tab);
}
```


9 Custom Tabbed Panel Content Area

This chapter will show how a `TabbedPanel` can be used with a custom content area or without any content area.

9.1 Using a Custom Content Area

A `TabbedPanel` has a default content area. The content area is responsible for being a container for the tabs' content components, showing the selected component and for rendering the looks for the content area such as border, color etc.

The default content area, `TabbedPanelContentPanel`, acts as a container that shows the selected tab's content component.

9.1.1 Custom Content Area Inside a Tabbed Panel

It's possible to substitute the default content area of a `TabbedPanel` with a custom content area. This makes it possible to use shadow and to change the tab area orientation around the content area.

Using a `JPanel` as custom content area inside a tabbed panel:

```
JPanel customContentArea = new JPanel();  
TabbedPanel tabbedPanel = new TabbedPanel(customContentArea);
```

Note that the `TabbedPanelContentPanelProperties` in `TabbedPanelProperties`, see section 4.8 *Tabbed Panel Content Panel Properties*, will not automatically apply to a custom content area.

9.1.2 No Content Area or Custom Content Area Outside a Tabbed Panel

It's possible to use a custom `JComponent` as content area. The component is then by itself responsible for handling the Tabs' content components. This can be achieved by listening to tab events from the `TabbedPanel` such as tab added, tab removed and tab selected.

A `TabbedPanel` can exist without a content area. In this case the `TabbedPanel` will only have a tab area and will look like a bar. The only way to know what's going on in the `TabbedPanel` is to register a `TabListener`.

Creating a tabbed panel without content area:

```
TabbedPanel tabbedPanel = new TabbedPanel(null);
```

Using a content area outside the `TabbedPanel` is accomplished by telling the `TabbedPanel` not to use a content area and then place a custom component somewhere that listens to events from the `TabbedPanel` and shows the Tabs' content components.

When the `TabbedPanel` doesn't have a content area both the tab orientation up and down will display the `TabbedPanel` as a horizontal bar and orientation left and right

will show the `TabbedPanel` as a vertical bar. The tabs and tab area components will still be positioned as if there was an invisible content area.

9.2 Example: Selection Bar and Content Area Outside a Tabbed Panel

Let's say we want a `TabbedPanel` with the following characteristics:

- Act as a vertical selection bar with titled tabs.
- The bar color should be gray.
- Titled tabs should have direction up.
- Black line border around tabs.
- Black line border around tab area.
- Tab should be transparent and the selected tab should be white.
- 2 pixel spacing between tab area edges and the tabs and a 2 pixel spacing between the tabs.
- 2 pixel spacing between tab area components and tab area edges. Tab area components should be stretched to be as wide as the tab area (minus the spacing).
- Tab drop down list button should be visible.
- Content area that can be placed anywhere outside the tabbed panel and shows selected tab's content component.

First we create the custom content area class.

Creating a custom content area:

```
public class MyContentArea extends JPanel {
    public MyContentArea(TabbedPanel tabbedPanel) {
        super(new BorderLayout());

        // Adding a tab listener so we know when a tab is selected
        tabbedPanel.addTabListener(new TabAdapter() {
            public void tabSelected(TabStateChangedEvent event) {
                // If there was a previously selected tab then we need to remove its
                // content component
                if (event.getPreviousTab() != null &&
                    event.getPreviousTab().getContentComponent() != null)
                    remove(event.getPreviousTab().getContentComponent());

                // If there's a currently selected tab we add its content component
                if (event.getCurrentTab() != null &&
                    event.getCurrentTab().getContentComponent() != null)
                    add(event.getCurrentTab().getContentComponent(), BorderLayout.CENTER);

                repaint();
            }
        });
    }
}
```

Now we create and configure a `TabbedPanel` containing 5 example `TitledTabs`.

Creating and configuring the tabbed panel and titled tabs:

```
LineBorder blackBorder = new LineBorder(Color.BLACK);
// Tabbed panel without content area
TabbedPanel tabbedPanel = new TabbedPanel(null);

// Configuring tabbed panel
tabbedPanel.getProperties().
    setTabAreaOrientation(Direction.LEFT).
    setTabSpacing(2).
    setTabDropDownListVisiblePolicy
(TabDropDownListVisiblePolicy.MORE_THAN_ONE_TAB).
    getTabAreaProperties().getComponentProperties().
        setBorder(blackBorder).
        setInsets(new Insets(2, 2, 2, 2)).
        setBackgroundColor(Color.LIGHT_GRAY);

tabbedPanel.getProperties().getTabAreaComponentsProperties().
    setStretchEnabled(true).
    getComponentProperties().
        setBorder(blackBorder).
        setBackgroundColor(null);

// Configuring titled tab
TitledTabProperties tabProperties = new TitledTabProperties();
tabProperties.setHighlightedRaised(0).
    getNormalProperties().
        setDirection(Direction.UP).
        getComponentProperties().
            setBorder(blackBorder).
            setBackgroundColor(null);
tabProperties.getHighlightedProperties().getComponentProperties().
    setBorder(blackBorder).
    setBackgroundColor(Color.WHITE);

// Creating the custom content area
MyContentArea contentArea = new MyContentArea(tabbedPanel);
contentArea.setBorder(blackBorder);

// Creating 5 titled tabs with JTextArea as content component
for (int i = 0; i < 5; i++) {
    TitledTab tab = new TitledTab("Tab " + i, null,
        new JTextArea("Content for Tab " + i), null);

    // Adding the configured titled tab properties as super object to each titled
    // tab's properties
    tab.getProperties().addSuperObject(tabProperties);
    tabbedPanel.addTab(tab);
}
```

We add the `TabbedPanel` and the content area to another container and we end up with a `TabbedPanel` with an outside content area, see *Figure 9.1*.

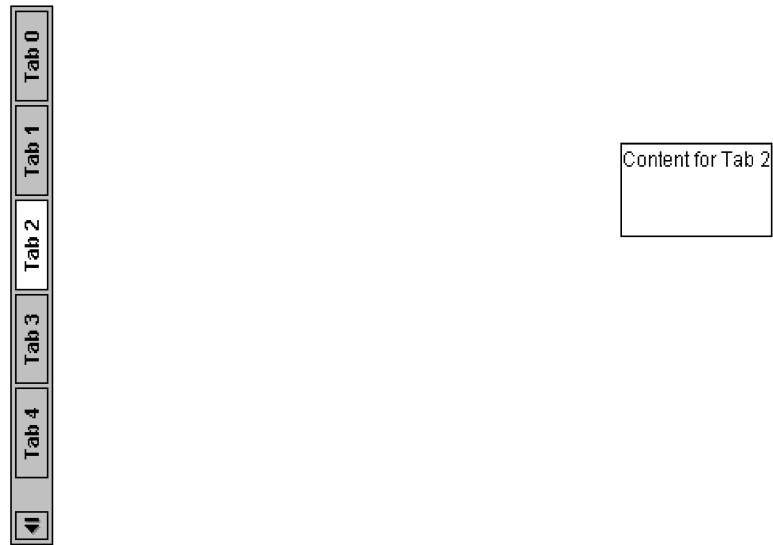


Figure 9.1

TabbedPanel as selection bar with a custom content area

10 Creating Custom Tabs

This chapter describes how to create custom Tabs. Tab is found in the `net.infonode.tabbedpanel` package.

10.1 Tab Container

A Tab is a container for other Swing components. It's basically an extended `JPanel` with a `BorderLayout` as layout manager, reference to a content component, an event mechanism and `opaque` set to `false` (transparent).

10.1.1 Adding Components

It's possible to extend a Tab or directly add components to a created Tab. The default layout manager can be changed using `setLayout()` as for any other Swing containers.

10.1.2 Event Components

A `TabbedPanel` listens to mouse events on the Tab so that it knows when the Tab is selected, dragged etc. The Tab is called an event component.

It's possible to tell what components in a Tab that the `TabbedPanel` should listen to. This is accomplished by either calling `setEventComponent()` (only one component) or `setEventComponents()` (with an array of components). These components can only be the Tab and/or any components added to the Tab.

10.2 Example: Creating a Custom Tab

We will now finish with an example that creates a custom Tab with the following characteristics, see *Figure 10.1*:

- A text.
- A button that closes the tab.
- Tab should have gray background and white background when selected.
- 5 pixel spacing between the tab border and the tab content on the left and right side and 2 pixel spacing on the top and bottom side.
- Tab should only be selected when clicking on the text.
- The button should be positioned to the left of the text.

The custom Tab added to a `TabbedPanel` can be seen in *Figure 10.2*.

**Figure 10.1**

A scetch of what the tab should look like

The code example for creating the custom tab:

```
public class MyCustomTab extends Tab {
    private JLabel textLabel = new JLabel();
    private JButton closeButton = new JButton("X");

    public MyCustomTab(String text, JComponent contentComponent) {
        super(contentComponent);

        textLabel.setText(text);

        // Make solid
        setOpaque(true);

        // Set background
        setBackground(Color.GRAY);

        // Add components, BorderLayout is default layout manager
        add(closeButton, BorderLayout.WEST);
        add(textLabel, BorderLayout.CENTER);

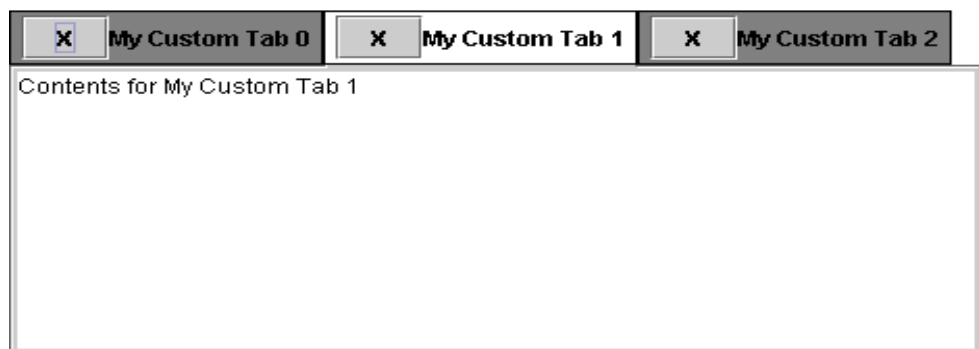
        closeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Close tab by removing it from the tabbed panel it is a member of
                getTabbedPanel().removeTab(MyCustomTab.this);
            }
        });

        // Set border and spacing
        setBorder(new CompoundBorder(new LineBorder(Color.BLACK),
                                     new EmptyBorder(2, 5, 2, 5)));

        // Set the textLabel as the only event component
        setEventComponent(textLabel);

        // Add a tab listener so we know when tab is selected or not
        addTabListener(new TabAdapter() {
            public void tabSelected(TabStateChangedEvent event) {
                if (event.getCurrentTab() == MyCustomTab.this)
                    setBackground(Color.WHITE);
            }

            public void tabDeselected(TabStateChangedEvent event) {
                if (event.getPreviousTab() == MyCustomTab.this)
                    setBackground(Color.GRAY);
            }
        });
    }
}
```

**Figure 10.2**

Three tabs of the type MyCustomTab (with JTextArea as content components) in a TabbedPane

11 Properties

This chapter is included in both the *“InfoNode Docking Windows Developer’s Guide”* and *“InfoNode Tabbed Panel Developer’s Guide”*.

The InfoNode properties module is located in the `net.infonode.properties` package. The properties module is NOT thread safe, so only one thread at a time should call methods inside the module. IDW and ITP always use the AWT event dispatching thread when calling methods in the properties module. InfoNode properties are similar to Java bean properties, but they are not connected to Java class and method names.

11.1 Property

A `Property` is similar to a Java class field. It has a name, a type and a description. It can belong to a `PropertyGroup`. The value of a `Property` can be any type of object and the value can be stored in any type of object, for example a `PropertyMap` (see chapter 12 *Property Maps*). To set the value of a `Property` in an object use the `Property.setValue()` method. To get the value of a `Property` from an object use the `Property.getValue()` method.

Some property value containers support removing of a property value with the `Property.removeValue()` method. To check if a property value can be removed use the `Property.valueIsRemovable()`.

A `Property` is type checked, so it can only be assigned values that are compatible with the type of the `Property`.

11.2 PropertyGroup

A `PropertyGroup` is similar to a Java class. It is a collection of `Property`s and has a name, a description and an optional parent group.

11.3 Typed Properties

The `net.infonode.properties.types` package contains a number of `Property` classes which have values that are instances of common Java classes. For example, the `IntegerProperty` class stores values that are instances of the `Integer` class.

11.4 PropertyValueHandler

The typed property classes will take a `PropertyValueHandler` object as constructor parameter. The `PropertyValueHandler` is responsible for storing property values in an object. As mentioned before property values can be stored in any type of object.

12 Property Maps

This chapter is included in both the “*InfoNode Docking Windows Developer’s Guide*” and “*InfoNode Tabbed Panel Developer’s Guide*”.

The property map module is a part of the InfoNode properties module. A property map is a powerful container for Property values.

12.1 Property Map Classes

The property map classes are located in the `net.infonode.properties.propertymap` package. *Figure 12.1* shows the property map classes and their relationships.

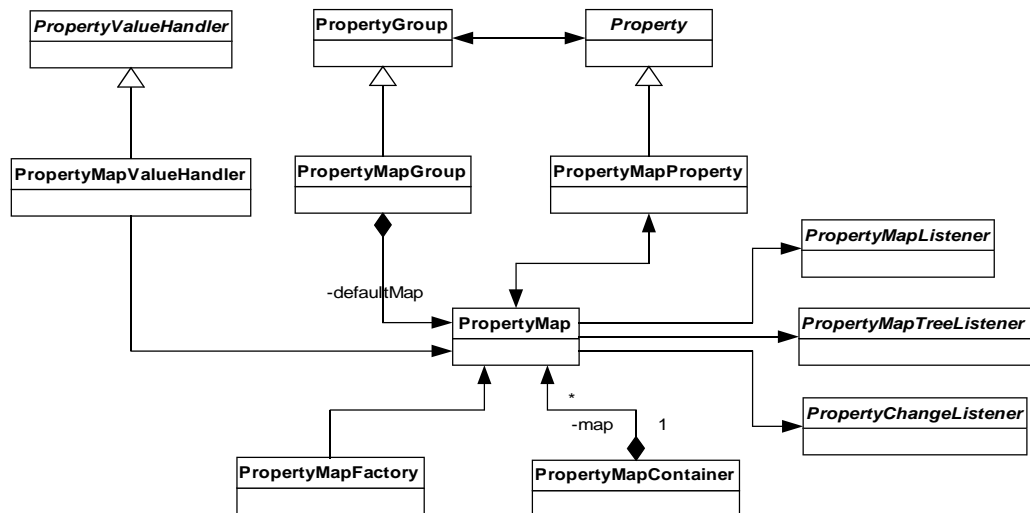


Figure 12.1
Property map classes.

The classes are described in the following chapters.

12.1.1 PropertyMap

A `PropertyMap` is a map for `Property` values. `PropertyMap` objects can be created using the methods in `PropertyMapFactory`. You can either create a `PropertyMap` for a `PropertyMapGroup` or a `PropertyMap` that has another `PropertyMap` as super map and the same `PropertyMapGroup` as the super map.

Use the `Property.setValue()` and `Property.getValue()` methods to store and retrieve values in a `PropertyMap`. `PropertyMap` supports removal of property values with the `Property.removeValue()` method.

12.1.2 PropertyMapGroup

A `PropertyMapGroup` is a special `PropertyGroup` which uses `PropertyMaps` as property value storage. Each `PropertyMapGroup` has a `PropertyMap` which contains default values for the `Property` in the group. These values are returned when a value is not found in a `PropertyMap`. You can access and modify the default values with `PropertyMapGroup.getDefaultMap()`. Note that modifying the default values will NOT trigger listener notifications in other `PropertyMaps`, so the default values should be set before any other `PropertyMaps` are created, for example in a static initializer.

12.1.3 PropertyMapValueHandler

`PropertyMapValueHandler` is used for handling property values stored in a `PropertyMap`.

12.1.4 PropertyMapProperty

A `PropertyMapProperty` is a property that has `PropertyMap` as values. These properties are automatically assigned a value which cannot be modified.

12.1.5 PropertyMapFactory

Creates `PropertyMap` objects.

12.1.6 PropertyMapContainer

A utility class that is used as base class for properties classes that uses a `PropertyMap` for property value storage.

12.2 Advanced Features

In this chapter some of the more advanced features of the `PropertyMap` class is described. These features are most easily described with examples using pseudo code. Here are the definitions of the `PropertyMapGroup` used in the examples:

- `Point` is a `PropertyMapGroup` which contains two `IntegerProperty`s, `X` and `Y`.
- `Line` is a `PropertyMapGroup` which contains two `PropertyMapProperty`s, `Start` and `End`, of type `Point`.

The figures in the examples uses normal arrows with a number to indicate super maps, the number is the search order, and arrows with a diamond to indicate map composition.

12.2.1 Super Maps

A `PropertyMap` can have a number of super maps. If a property value isn't found in a `PropertyMap`, the last added super map is searched for the property value. If it isn't found there, the next super map is searched and so on.

A super map is added using `PropertyMap.addSuperMap()`, and removed using `PropertyMap.removeSuperMap()`. A super map can be replaced using the `PropertyMap.replaceSuperMap()` method.

An example:

1. `P1 = new Point`
2. `P1.X = 1, P1.Y = 2`
3. `P2 = new Point`
4. `P2.Y = 3`
5. `P1` is added as super map to `P2`. Now `P2.X == 1`, the value is found in `P1`, and `P2.Y == 3`, the value is found in `P2`.

12.2.2 Property Map Composition

When a `PropertyMap` is created for a `PropertyMapGroup` that contains `PropertyMapProperty`s, each `PropertyMapProperty` is assigned a new `PropertyMap` as value. Getting the property value will return this `PropertyMap`. The property value is constant and can't be modified.

When a super map, `A`, is added to a `PropertyMap`, `B`, which contains `PropertyMap` values, the corresponding `PropertyMap` value in the `A` is added as super map to each `PropertyMap` value in `B`. The super map is added after, and thus its values are overridden by, any super maps explicitly added to a `PropertyMap` value in `B`. The process is recursively repeated. The reverse operation is performed when a super map is removed.

For example:

1. `L1 = new Line`
2. `P1 = new Point`
3. `P1.X = 1`
4. `P1` is added as super map to `L1.Start`, which means that `L1.Start.X == 1`

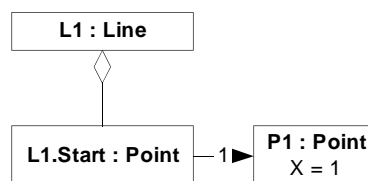


Figure 12.2

The PropertyMap hierarchy after step 4

5. `L2 = new Line`
6. `L2.Start.X == 2, L2.Start.Y == 3`

7. L2 is added as super map to L1. This causes L2.Start to be added as super map to L1.Start, but the values in P1 overrides the values in L2.Start. So now L1.Start.X == 1 and L1.Start.Y == 3.

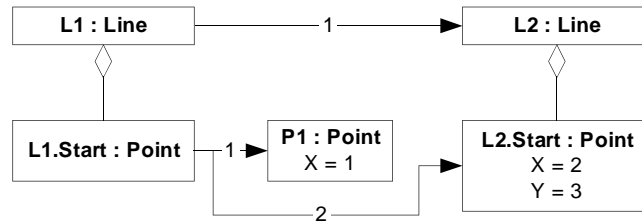


Figure 12.3

The PropertyMap hierarchy after step 7

12.2.3 Property Value References

Using `PropertyMap.createRelativeRef()` you can create a property value that is a reference to another property value. The reference is inherited just like normal property values, but its actual value may differ depending in which `PropertyMap` the property value is read. The reference is dereferenced relative to the `PropertyMap` where the property value is read.

Here's an example on how it works:

1. `L1 = new Line`
2. `L1.End.Y = 1`
3. A relative reference is created from `L1.Start.X` to `L1.End.Y`. So, `L1.Start.X == 1`

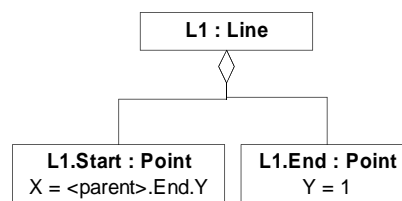


Figure 12.4

The PropertyMap hierarchy after step 3

4. `L2 = new Line()`
5. L1 is added as super map to L2. Now `L2.Start.X == 1`, because:
 1. No value for X is found in `L2.Start`, so `L1.Start` is searched and the reference is found.

2. The reference is dereferenced relative to L2 which causes a lookup for the value of L2 . End . Y.
3. No value for Y is found in L2 . End, so the value in L1 . End is returned.

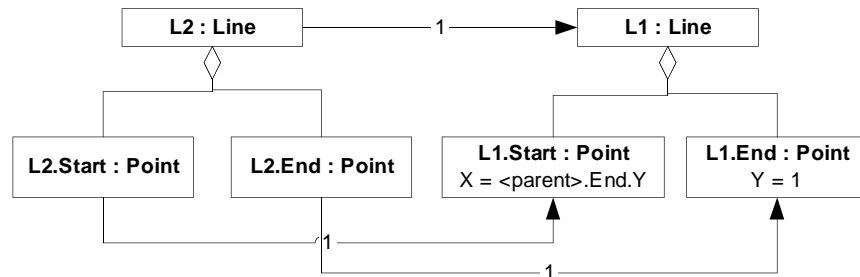


Figure 12.5
The PropertyMap hierarchy after step 5

6. L2 . End . Y = 2. Now L2 . Start . X == 2 because the reference is inherited to L2 and dereferenced relative to L2. Still L1 . Start . X == 1 though.

12.2.4 Listeners

Three types of listeners can be added to a PropertyMap:

PropertyChangeListener	Listens to value changes in the PropertyMap, but not changes in child maps. Value changes for multiple properties can be bundled together in a single listener notification.
PropertyMapTreeListener	Listens to value changes in the PropertyMap and all its child maps recursively. Value changes for multiple properties can be bundled together in a single listener notification.
PropertyChangeListener	Listens for a value change for a specific property in a PropertyMap.

The listeners are notified when the Property.getValue() with the PropertyMap would return a different value than before, not only when Property.setValue() is called for the PropertyMap. This means that when a property value that is referenced is modified listener notifications will be triggered in all maps that reference the value. For example, the listeners of a PropertyMap will be notified if a the map contains no value for a property and the value of that property is changed in a super map, or if a new super map where this value is set is added to the PropertyMap.

12.2.5 Weak Listeners

A weak listener is a listener that are garbage collected and removed from the PropertyMap it listens to when there are no strong or soft references to the listener. Weak listeners are useful for example when you want to listen to a PropertyMap, but you don't want the listener to prevent the map from being garbage collected.

The PropertyMapWeakListenerManager class handles weak listeners and

contains methods for adding the three listener types as weak listeners to a `PropertyMap`. You must use the remove methods in `PropertyMapWeakListenerManager` when removing a previously added weak listener.

12.2.6 Batch Processing

The `PropertyMapManager.runBatch()` method can be used minimize the number of listener notifications when performing multiple property value changes. All property value modifications done inside `run()` of the `Runnable` passed to `PropertyMapManager.runBatch()` will be bundled together and sent to listeners when the `run()` method returns. No listener notifications are performed before that. Each listener is guaranteed to be called at most once for each batch operation.

Below is an example that sets two `Property` values:

```
PropertyMapManager.runBatch(new Runnable() {
    public void run() {
        X.set(p1, 4);
        Y.set(p1, 5);
    }
});
```

12.2.7 Serialization

A `PropertyMap` can be written to an `ObjectOutputStream` using the `PropertyMap.write()` methods. If the recursive flag is set, all the `PropertyMaps` of all `PropertyMapProperty`s will be written recursively. In the stream the `Property`s are identified by their name. The property values are written using normal serialization.

A previously written `PropertyMap` is read using the `PropertyMap.read()` method. If the map was written recursively, it is read recursively as well. If a name for a non-existing `Property` is found in the stream the value is skipped. Not all `Property`s must have a value in the stream.

12.3 ComponentProperties

The `net.infonode.properties.gui.util.ComponentProperties` class contains properties common to all `JComponent`. The property values can be applied to a `JComponent` using the `applyTo()` methods.