Section Layout 1.0 Component Specification

1. Design

The Java Swing Section Layout component provides a Swing layout to be used in conjunction with the Java Swing Side Menu component. This component will provide sections that can be expanded, contracted, docked, and undocked from the side menu, or any other Java Swing container. Each layout will contain a number of titled sections that themselves will contain separate GUI controls.

Design Hints

To make this design flexible, I split it into three parts:

1. The section control
   A swing component with a section header can be expanded and contracted. To make its L&F pluggable as general swing component, I defined the Section Header as a custom swing component, and used MVC pattern. (Writing own data model and UIDelegate)

2. The Dock Framework
   The section control itself can't be docked or undocked directly. I designed a dock framework to archive the dock functionality, of cause, the framework is not only used for section controls, it is general to dock any swing components.

3. One custom Dock container which can dock the sections
   The dock container is the container to dock components, in this component, an implementation of it is provided to dock sections. Note, it is not a LayoutManager instance; a layout can't achieve the dock functionality. Also, the section layout described in RS can be simply achieved by using BoxLayout.

Enhancements

1. Providing a general dock framework. This is very useful and not only used for this component. See the demo to know how to use this dock framework.

1.1 Design Patterns

MVC Pattern – The SectionHeader custom component follows Swing component MVC pattern.
Listener Pattern – The Dock, DockListener and Section, SectionExpandContractListner implements this pattern.

1.2 Industry Standards

Java
Swing
Drag and Drop

1.3 Required Algorithms

1.3.1 How does the BoxLayout work for the requirements

BoxLayout can be in vertical or horizontal orientation, in each orientation, the box layout only allows having one column or row of the contained components, the layout first assign the space to each component according the preferred size, if there still has some empty space, the layout will try to assign the space according the maximum size, if one or more components' maximum size is more than the empty space size, the space will be assigned averagely.

In this component, we assume the container for sections is the side menu, and it is in vertical orientation, all the sections should expand the width to the side menu's width, and if the section is configured as expanded to fill the empty space in the vertical orientation, the height of the section should also expanded.
This can be achieved by overriding the getMaximumSize of the section which extends JPanel.

```
if (this.isExpandToFill() && this.isExpanded()) {
      // return the size fill all the screen so that this
      // section can fill the all the empty size of dock
      // container.
      return Toolkit.getDefaultToolkit().getScreenSize();
}
// this section is not expandToFill,
Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
if (this.isVertical()) {
      // set the width to the screen width, so that the width
      // of section can fill the section container's width
      return new Dimension(screenSize.width,
            this.getPreferredSize().height);
}
else {
      // set the height to the screen height, so that the width
      // of section can fill the section container's height
      return new Dimension( this.getPreferredSize().width,
            screenSize.height);
}
```

Note: in this implementation, more than one sections can be set to expanded to fill the empty space, they will share the empty space averagely.

1.3.2 *How to repaint the section container*

It's done by the swing framework automatically. When the section is expanded or contracted, it will call invalidate and repaint methods in the section, and these methods will be delivered to the section container automatically by the swing framework.
When the section is docked or undocked, the container will be repainted as the docked/undocked method will call add/remove method in the native swing container of the Dock container.

1.3.3 *How does the expansion and contraction work*

When the user click the expansion/contraction icon, the expanded property in the section model will be changed as the mouse event was listened by the section header UI, and the section itself will listen to model properties change events and set the inner component visible or invisible. It will also invalidate the section component so that the whole section container will be repainted.

1.3.4 *How does the dock function work*

This component provides a flexible dock framework to support the dock of section. In this framework, every swing component wants to be dockable should implements the Dockable interface, that interface allow a DragGestureRecognizer listening on the swing component. Also the section container should be wrapped by the DockContainer so that it can be treated as a Drop Target, when a dockable component is dragged into the dock

container, an event will be triggered and a method will be invoked, the method is to mark a gesture to indicate the component can be docked in the dock container. When the user drops the component into the container, the component will be added to the target location in the container and the container will be repainted. And if the component is dropped out of any dock container, it will be floated, a floating dock container will be created automatically and the component will be docked into that floating container and the floating container will be located at the point where the component dropped.
Also, when the user closed the floating container, it will undock the component and dock the component to its original container automatically.

1.4 Component Class Overview

Dock:

Dock represents an item can be docked into dock container. It is designed to be used as a wrapper of swing component which implements the Dockable interface. When creating an instance of it with a specific swing component, the instance should setup any necessary mechanism to enable the dock functionality.

Dockable:

This interface is designed to be implemented by any swing component supporting docking functionality. It defines method to accept a DragInitListener which used to trigger the start of dragging, it also defines method that returns the dock type, can be determined by dock container whether to dock or not.

DragInitListener:

It is a listener to listen whether a dockable component trigger an event to start drag. It can be set to a Dockable instance.

DockEvent:

Represents the event of whether a dock is docked or undocked. The Dock instance, DockContainer instance and the location/index of the dock is wrapped into this event.

DockListener:

The listener is used to listen to a Dock instance to know whether it is docked or undocked to/from a dock container. When a dock is docked, the docked method will be fired, when a dock is undocked, the undocked method will be fired.

DockContainer:

This interface defines a container which can dock or undock a Dock instance. It is defined as a wrapper of swing Container. When an instance of this is created, it will setup any necessary mechanism to enable the dock functionality in the specific swing container.

FloatContainer:

It is a floatable dock container. When a dock is dropped not in a dock container, and if the dock is floatable, it should be put in a float dock container. The float dock container can be set to arbitrary location, can be closed, when it is closed, the docks in it should be docked back to the original dock containers.

DefaultDock:

The default implementation of the Dock interface by using DnD(java.awt.dnd) framework in java.

DockHandler:

The package class is used to handle dragging event on the dockable component. It also defines the DataFlavor to transfer which is required in DnD framework.

DockDragGestureRecognizer:
This is a DragGestureRecognizer implementation for this component. When it is constructed, it will register DragInitListener to the dockable component to recognize the start of dragging. It is used by DockHandler class.

DockContainerHandler:
This package class is used to handle dragging event on the dock container. It first creates the drop target corresponding to the dock container, then listens to the drop target as it implements the DropTargetListener. If the drag enters the dock container, it will let the container mark a gesture; if the drag exits the dock container, it will let the container clear the gesture, if it is dropped into the dock container, it will let the container dock the transferred dock instance. It is used by BasicDockContainer class.

BasicDockContainer:
It is the abstract implementation of DockContainer by using DnD framework. Every implementation using DnD can extend from it.

DialogFloatContainer:
It is the default implementation of the Float Container by using the JDialog component as the native swing component. The created JDialog instance will not be resizable. If this container is closed, it will also re-dock the containing dock back to the original dock container.

DockableWrapper:
It is a convenient class to wrap a swing component which doesn't implement the Dockable yet. The typical usage is like this.

Section:
It is the section control of this component. A section is a swing component which can be expanded or contracted.

SectionExpandContractListener:
It is the listener to listen the Section instance to know when it is expanded or contracted.

SectionDockContainer:
The default implementation of dock container in this component, it supports the dock functionality for Sections.

DockContainerPanel:
The package class is used to support the gesture painting for SectionDockContainer. It overrides the paint method to achieve this.

SectionHeader:
The custom component represents the header of the Section control. It use MVC pattern, with the model SectionModel and the View SectionHeaderUI. It also listen to the Section to know if the properties is changed, if the "expanded", "floating", "title" property changed, repaint the header.

SectionHeaderUI:
The abstract class defines the super UI class for SectionHeader component. All the UI implementation of SectionHeader should extend this class.

DefaultSectionHeaderUI:

The default UI implementation of SectionHeader component. It mixed the View and Control function of the SectionHeader. For events listeners: it listens to the section header to trigger Drag Start Event, and set Expanded property of section. For VIEW : it paint the section header using the defined properties in UIManager so that the L&F can be pluggable.

SectionModel:

The model of the Section component. It contains the state of the Section.

DefaultSectionModel:

The default implementation of the SectionModel in this component.

1.5 Component Exception Definitions

IllegalArgumentException:

This exception is thrown if any argument is invalid in this component. (For example, null, empty argument sometime is not accepted.)

UnsupportedFlavorException:

This exception is thrown if the passed data flavor is not supported in DockHandler class.

1.6 Thread Safety

This component is not thread-safe. Also the thread-safe is not required for this component.

In Swing Framework, the components are not thread-safe so that this component can't be thread-safe as it used or extended the components in swing. But the listeners' management is thread-safe in this component just as the normal swing component do.

2. Environment Requirements

2.1 Environment

- Development language: Java1.5
- Compile target: Java1.5

2.2 TopCoder Software Components

None

The Look & Feel will be configured in swing framework.

The Sections in the side menu will be configured programmatically, so that the Configuration Manager 2.1.5 is not used.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.gui.sectionlayout
com.topcoder.gui.sectionlayout.dock
com.topcoder.gui.sectionlayout.dock.dndimpl
com.topcoder.gui.sectionlayout.model

com.topcoder.gui.sectionlayout.ui

3.2 Configuration Parameters

The resources used in SectionHeader component.

ResourceName	Type	Description
SectionHeader.font	Font	The font of section header
SectionHeader.height	Integer	The height of section header
SectionHeader.expandedHeaderForeGroundColor	Color	The foreground color of section header used when the section is expanded.
SectionHeader.contractedHeaderForeGroundColor	Color	The foreground color of section header used when the section is contracted.
SectionHeader.dragAreaLeftX	Integer	The x of upper-left point of the drag area
SectionHeader.dragAreaUpperY	Integer	The y of upper-left point of the drag area
SectionHeader.dragAreaWidth	Integer	The width of upper-left point of the drag area
SectionHeader.dragAreaHeight	Integer	The height of upper-left point of the drag area
SectionHeader.expandedIcon	Icon	The icon displayed when the section is expanded
SectionHeader.contractedIcon	Icon	The icon displayed when the section is contracted
SectionHeader.iconX	Integer	The x of upper-left point of the icon area
SectionHeader.iconY	Integer	The y of upper-left point of the icon area
SectionHeader.titleX	Integer	The x of upper-left point of the title
SectionHeader.itleY	Integer	The y of upper-left point of the title
SectionHeader.expandedBgLeftIcon	Icon	The image drawn on the left side of the section header when the section is expanded
SectionHeader.expandedBgRightIcon	Icon	The image drawn on the right side of the section header when the section is expanded
SectionHeader.expandedBgMidIcon	Icon	The image drawn on the middle of the section header when the section is expanded, this image is drawn repeatable in the middle.
SectionHeader.contractedBgLeftIcon	Icon	The image drawn on the left side of the section header when the section is contracted
SectionHeader.contractedBgRightIcon	Icon	The image drawn on the right side of the section header when the section is contracted
SectionHeader.contractedBgMidIcon	Icon	The image drawn on the middle of the section header when the section is contracted, this image is drawn repeatable in the middle.

3.3 Dependencies Configuration

None

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution
- Follow Dependencies Configuration
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

4.3 Demo

4.3.1 Create Section with existing component

```
Component innerComponent = new JPanel();
Section section = new Section(innerComponent); //default vertical
// configure section
section.setTitle("Tools");
section.setSize(100, 150);

// default false, if it is true, this section will expand to
// take up the empty space in the container
section.setExpandToFill(true);

// set to horizontal orientation, default to vertical
section.setVertical(false);

section.addExpandContractListener(new SectionExpandContractListener() {
    public void contracted(Section section) {
        System.out.println(section.getTitle() + " contracted");
    }
    public void expanded(Section section) {
        System.out.println(section.getTitle() + " expanded");
    }
});
```

4.3.2 How to dock the sections in the side menu

```
// assume the side menu is a JPanel
JPanel sideMenu = new JPanel();
// Create Dock Container, the orientation is vertical by default
DockContainer dockContainer = new SectionDockContainer(sideMenu);

// create dock container with horizontal orientation
dockContainer = new SectionDockContainer(sideMenu, false);

// create Dock for section, the section is created in 4.3.1
Dock dock = new DefaultDock(section);
dockContainer.dock(dock);

// set floatable of the dock to false,
// the dock will not be floatable
dock.setFloatable(false);

// Disable the drag of the dock, the dock
// can't be dragged again.
dock.setDragEnabled(false);

// adding Drag listeners to the dock and dock container
DockListener dockListener = new DockListener() {
    public void docked(DockEvent event) {
        System.out.println(event.getDock() + " docked in " + event.getContainer()
                        + ". Index is " + event.getIndex() + ", Location is " +
event.getLocation() + ".");
            }
            public void undocked(DockEvent event) {
                System.out.println(event.getDock() + " undocked in " +
event.getContainer()
                        + ". Index is " + event.getIndex() + ", Location is " +
event.getLocation() + ".");
            }
        };

        dock.addDockListener(dockListener);
                dockContainer.addDockListener(dockListener);
```

4.3.3 *How to dock swing components other than Section*

```
    /**
     * <p>
     * You can extends <code>SectionDockContainer</code> or
<code>BasicDockContainer</code>
     * to implements your own <code>DockContainer</code>.
     * This <code>PowDockContainer</code> can dock every thing.
     * </p>
     *
     * @author dmks, TCSDEVELOPER
     * @version 1.0
     */
    public class PowDockContainer extends SectionDockContainer {
        /**
         * <p>
         * Constructs a PowDockContainer instance.
         * </p>
         *
         * @param container
         *          the outer container
         */
        public PowDockContainer(Container container) {
            super(container);
        }

        /**
         * <p>
         * Return whether this container can dock the dock.
         * </p>
         *
         * @param dock
         *          the dock to be checked
         * @return whether this container can dock the dock
         */
        public boolean canDock(Dock dock) {
            // dock every thing
            return true;
        }
        }

        // First you need to get a DockContainer can dock the dock type "DockableWrapper"
        DockContainer container = new PowDockContainer(sideMenu);
        // The mouse pressed on any position on the "anyComponent" will start drag.
        Component anyComponent = new JPanel();
                       container.dock(new DefaultDock(new DockableWrapper(anyComponent)));
```

5. Future Enhancements

None