Java Swing Section Layout 1.0 Requirements Specification

1. Scope

1.1 Overview

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.

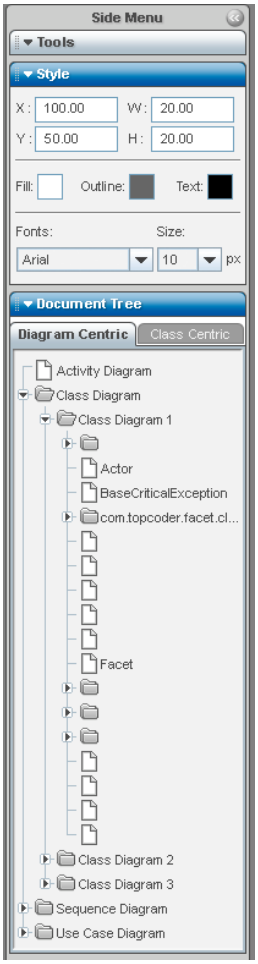
1.2 Logic Requirements

1.2.1 Control layout

Prototype images are given that explain the overall layout and functionality of the side menu. Note that the custom look and feel **is not** part of the competition, as that will be handled externally. Although the image shown shows the layout in a vertical manner, the layout should support both vertical and horizontal layout grids.

1.2.2 Sections

Each item in the layout should be defined as a title section. In the example below, the sections are “Tools”, “Style”, and “Document tree”. Each section should contain getters and setters for the following items: Height, Width, and Title. There should also be ways for the user to set the internal control contents of each section.



1.2.3 Section layout

The layout will support both a vertical and horizontal orientation. In the example above, a vertical orientation is shown, but this component must also support a horizontal orientation. In a vertical orientation, the component only needs to support a single column, and in a horizontal orientation, it only needs to support a single row.

1.2.4 Section spacing

The sections also need a property that allows the user to specify that a specific section fills in any additional space in the side menu. In the prototype image above, it can be seen that the "Tools" and "Style" side menu panels are fixed height, but the "Document Tree" side menu section is allowed to expand to take up any remaining space in the side menu.

1.2.5 Section header

Each section has a header that includes the title, an arrow for expanding and contracting, and a grab area where the section can be docked and undocked. For this component, placeholders can be used for the expand and contract images.

1.2.6 Section expanding and contracting

Each section can be expanded or contracted by clicking on the image to the left of the title. When the section is contracted, only the header area is visible, and when it is expanded the entire internal contents are visible. When a section is contracted, the other sections should resize accordingly. In the example image above, if the "Style" section is contracted, "Document Tree" would expand to take up the empty space created.

1.2.7 Side menu panel docking

This component needs to support docking, undocking, and reordering of the sections in the side menu. The undocking should be done by grabbing the area to the left of the expand / contract arrow and dragging the section out of the layout. Docking is done the same way, where the user drags the window back onto the panel. There needs to be a visual indicator when docking to show the user where the section is going to be placed in the layout.

When a panel is "undocked", it should be contained in a floating tool window, with the header title visible, but without the expand and contract arrow. Both expanded and contracted sections will be visible when undocked. If a user undocks a contracted section, the section's contents should be visible in the resultant tool window.

A user should be able to undock an item and dock it in a different location in the layout. In the example image above, the user can undock the "Document Tree" section and then dock it back into the side menu between the "Style" and "Tools" sections. The reordering should not affect the side menu panel spacing mentioned in requirement 1.2.3.

1.2.8 Events

There should be an event handler interface defined to allow the developer to define custom functionality when a section is expanded, contracted, undocked, or docked. When docking, the index or location of the section's new placement in the layout should be available.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

This component will be used in the TopCoder UML Tool to house functionality for modifying elements, and will allow the user to customize how they want their environment to function.

1.5 Future Component Direction

None

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

The tool should match the prototype given, using placeholders for images. The individual section header styles do not have to match the prototype, as the exact look and feel will be updated.

2.1.2 External Interfaces

None

2.1.3 Environment Requirements

- Development language: Java 1.5
- Compile target: Java 1.5

2.1.4 Namespace

com.topcoder.gui.sectionlayout

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The sections to be displayed in the layout

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None.

3.2.2 TopCoder Software Component Dependencies:

- Configuration Manager 2.1.5

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3Third Party Component, Library, or Product Dependencies:

None

3.2.4QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003

3.3Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4Required Documentation

3.4.1Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2Help / User Documentation

XML documentation must provide sufficient information regarding component design and usage.