Elements Toolbar 1.0 Component Specification

1. Design

The Elements Toolbar component provides a SWING toolbar that allows the user to add different elements to the diagram. It provides a way to signal the listeners that an element was selected to be added.

The ToolBarMainPanelProducer class produces the main panel. The main panel is organized in GridBagLayout way. It retains the SectionPanel instances in every cell precisely.

The SectionPanel class represents the panel, which groups the similar elements as the “connectors” section in GUI demo. It lays out the buttons in FlowLayout way. Thus the button’s position can be adjusted dynamically. It can get current selected button instance. When one element is selected, the relative ElementSelectedListener instances will be notified. It has inner class SectionPanelListener, which implements the ActionListener interface and listens for every buttons’ event of the SectionPanel.

The ElementSelectedListener class acts as an interested listener to the specified selected elements. When a specified element is selected, the listener will receive the notification. The implementation of ElementSelectedListener can spawn a new thread if necessary.

1.1 Design Patterns

MVC pattern is used in the GUI component.

Observer pattern. ElementSelectedListener class and SectionPanelListener class act as observers.

Factory pattern. ToolBarMainPanelProducer class produces the main panel of elements toolbar, which implements factory pattern.

1.2 Industry Standards

Java AWT
Java Swing

1.3 Required Algorithms

None

1.4 Component Class Overview

ToolBarMainPanelProducer (class)

This class is used to produce the main panel of element toolbar, which contains many other SectionPanel instances in GridBagLayout way. The method "addNewSectionPanel" is used to add new SectionPanel instance into the main panel, whose location and size are set by the arguments. The method "getToolBarMainPanel" is used to retrieve the main panel of element toolbar.

Thread Safety: The members are immutable, but the method "addNewSectionPanel" isn't thread-safe. Hence, it is not thread safe.

SectionPanel (class)

This class represents the unit panel of the element toolbar. It can be constructed as "connectors" or "nodes" panel. It contains two inner panels. One is for the title; another

is for the instances of JToggleButton. The instances of JToggleButton are laid out in the FlowLayout way. It has two initialization ways. One is from programming; another is from loading configuration file.

The private method "initSectionPanel" is used to initialize the SectionPanel. The private method "createJToggleButton" is used to create new JToggleButton instance.The public method "unselectButton" is used to unselect the selected button. There are three methods to add or remove interested listeners for specified toggle buttons.

Thread Safety: The members are mutable, and the public methods can be accessed by many threads. Hence, this class is not thread safe.

SectionPanelListener (inner class of SectionPanel)

This inner class acts as the listener for the action event of all buttons in the SectionPanel. When an action even happens, it will find the button that the action event comes from. If the state of the button is unselected, notify all the interested listeners.

Thread Safety: The class has no member, and the public method is the implementation of ActionListener. Hence, this class is thread-safe.

ElementSelectedListener (interface)

This interface acts as an interested listener to the specified selected elements. When a specified element is selected, the listener will receive the notification; this is implemented by calling “elementSelected” method.

Thread safety: This interface can be implemented thread safe or not. The implementation of ElementSelectedListener interface, if required, can spawn a new thread though the notification will be given through the Event Dispatch Thread.

1.5 Component Exception Definitions

SectionPanelConfigurationException (custom):

This exception is thrown by the constructor of the SectionPanel class through loading configuration file. It primarily indicates that there was problem when initializing the instance of SectionPanel.

It is thrown when the namespace is unknown or the image location doesn't exist.

IllegalArgumentException (system):

This exception is thrown when the arguments are invalid for the methods.

In constructor of ToolBarMainPanelProducer, It is thrown if the number of rows or columns is not greater than zero. If look is null or empty, it should be thrown.

In method “addNewSectionPanel” of ToolBarMainPanelProducer, it is thrown if panel is null, or other arguments are non-negative.

In constructor of SectionPanel, it is thrown if namespace is null or empty.

In method of “initSectionPanel” of SectionPanel, it is thrown if sectionText is null or empty, backgroundColor is null, buttonNames is null, element in buttonNames is null or empty, images is null, or element in images is null. If the length of the buttonNames and the images is not equal, it should throw IllegalArgumentException.

Some methods may not refer to, but details are list in the doc tab.

1.6 Thread Safety

This component is not required to be thread-safe and is not thread-safe. In this component, the ToolBarMainPanelProducer is not thread safe, whose members are immutable, but the method "addNewSectionPanel" isn't thread safe. The SectionPanel is

not thread-safe, members are immutable, but the public methods can be accessed by threads. The inner class SectionPanelListener can not be accessed from outside of the component, and they are thread-safe at the same time. The implementation of ElementSelectedListener interface, if required, can spawn a new thread though the notification will be given through the Event Dispatch Thread.

In all, the whole component isnot thread safe.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

Configuration Manager 2.1.5

It will be used in SectionPanel to load configuration values from configuration file.

Base Exception 1.0

The exception SectionPanelConfigurationException defined by this component extends from BaseException from the BaseException 1.0 component.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.gui.toolbars.elements

3.2 Configuration Parameters

Parameter	Description	Values
look	**Required** The name of the class that implements the look and feel	A string (eg: “ javax.swing.plaf.metal.MetalLookAndFeel”)
sectionText	**Required** The section text, which is label at the head of section panel.	A string (eg: “connectors”)
backgroundColor	**Optional** The background color of section text panel. If it doesn’t exist, the component will use default Color.LIGHT_GRAY	A string to describe Color (eg: “YELLOW”). A string in form of #RRGGBB is also accepted (e.g. #FFFFFF, #12ff00);
buttonsName	**Optional** The buttons' name array. If it doesn’t exist, the component will create an empty section panel	An array of string to describe every button’s name (eg:”up”, “down”)
imageLocations	**Optional** The locations of button’s icon images. If it doesn’t exist, no image icon will be created for buttons	An array of string to describe every location of button’s icon image, which should be relative to the component. (eg: "toolbarButtonGraphics/navigation/Up16.gif”)

Here is the sample of configuration file, which is also located in the “docs” folder.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--This is a sample configuration file to illustrate how to configure SectionPanel correctly-->
<CMConfig>
   <!--The namespace of all parameters is free to be set any string-->
   <Config name="com.topcoder.gui.toolbars.elements">
   <!--The name of the class that implements the look and feel, which is required-->
   <Property name="look">
                  <Value>javax.swing.plaf.metal.MetalLookAndFeel</Value>
         </Property>
         <!--The section text, which is label at the head of section panel and is required-->
         <Property name="sectionText">
                  <Value>connectors</Value>
         </Property>
         <!--The background color of section text panel. If it doesn’t exist, the component will use
```

```
default Color.LIGHT_GRAY-->
        <Property name="backgroundColor">
                <Value>YELLOW</Value>
        </Property>
        <!--The buttons' name array. If it doesn't exist, the component will create an empty
section panel-->
        <Property name="buttonsName">
                <Value>up</Value>
                <Value>down</Value>
        </Property>
        <!--The locations of button's icon images. If it doesn't exist, no image icon will be created
for buttons-->
        <Property name="imageLocations">
                <Value>toolbarButtonGraphics/navigation/Up16.gif</Value>
                <Value>toolbarButtonGraphics/navigation/Down16.gif</Value>
        </Property>
        </Config>
</CMConfig>
```

3.3 Dependencies Configuration

Please review the components "Configuration Manager 2.1.5" to make it work.

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

- If the component will be used by loading configuration parameters, the required components are configured correctly and are available at runtime. Otherwise, no extra step to be done to use the component.

4.3 Demo

Note: The handling of the exceptions has not been shown in the demo.

1. **Create the Main Panel of Elements Toolbar**

```
//To create elements toolbar, component user creates four instances of SectionPanel
String look = "javax.swing.plaf.metal.MetalLookAndFeel";

//Initialize buttonsName and icons;
String[] buttonNames = new String[] {"buttonName", "buttonName", "buttonName",
    "buttonName"};
ImageIcon[] icons =
```

```
            new ImageIcon[] {new ImageIcon("test_files/loupe_black.gif"),
             new ImageIcon("test_files/loupe_black.gif"), new ImageIcon("test_files/loupe_black.gif"),
             new ImageIcon("test_files/loupe_black.gif")};

        //Initialize elementPanel and connectorsPanel through regular constructors.
        SectionPanel elementsPanel = new SectionPanel("elements",Color.RED, buttonsName,
        icons, look);

        SectionPanel connectorsPanel = new
        SectionPanel("connectors",Color.LIGHT_GRAY,buttonsName, icons, look);

        //"nodesPanel" is created from configuration file. Assume namespace is
        //"com.topcoder.gui.toolbars.elements.nodesPanel"
        String namespace = "com.topcoder.gui.toolbars.elements.nodesPanel";
        SectionPanel nodesPanel = new SectionPanel(namespace);

        //Create an empty panel
        SectionPanel emptyPanel = new SectionPanel("   ", Color.LIGHT_GRAY, null, null, look);

        //Create one instance of ToolBarMainPanelProducer and add these SectionPanel instances
        //into the main panel.
        Dimension preferredSize = new Dimension(300, 300);
        Dimension minimumSize = new Dimension(50,50);
          ToolBarMainPanelProducer producer = new ToolBarMainPanelProducer(2,2, look,
preferredSize, minimumSize);

        producer.addNewSectionPanel(0,0,1,1, elementsPanel);

        producer.addNewSectionPanel(0,1,1,1, connectorsPanel);

        producer.addNewSectionPanel(1,0,1,1, nodesPanel);

        producer.addNewSectionPanel(1,1,1,1, emptyPanel);

        //Get the main panel from the instance of ToolBarMainPanelProducer.
        JPanel mainPanel = producer.getToolBarMainPanel();
```

2. Manage Listener

```
        //Assume that DefaultListener is the default implementation of ElementSelectedListener,
        //which may be implemented in other components. It is implemented like this.
        public class DefaultListener implements ElementSelectedListener {

            /** Represents the current received element name. */
            private String elementName = null;

            public String getElementName() {
                return elementName;
            }

            public void elementSelected(JToggleButton button, SectionPanel sectionPanel) {
                this.elementName = button.getName();
            }
        }

        //Add new listener, which is interested in the "Up" button selection event.
        DefaultListener listener = new DefaultListener();
```

```
nodesPanel.addElementSelectedListener("buttonName", listener);

//If the listener is interested in all buttons of nodesPanel, it will be registered as following
nodesPanel.addListenerForAllButtons(listener);

//Remove listener, which is interested in the "Up" button selection event.
nodesPanel.removeElementSelectedListener("buttonName", listener);
```

3. **Get Information of Generating Event**

```
//Assume that the button "Up" is clicked in nodesPanel. Then the name of the clicked button,
//which is also the selected element name, will be notified to its interested listeners and the
//information of the selected element can be retrieved by this way.
String selectedElement = listener.getElementName();
```

4. **Unselect Toggle Button**

```
//When user gets the selectedElement, he can unselect the toggle button.
//Assume that the button "Up" is selected in nodesPanel. The value of selectedElement is
//equal to "Up". To unselect the button, just do the following.
nodesPanel.unselectButton(selectedElement);
```

5. Future Enhancements

None