SWING Configuration Manager Editor 1.0 Component Specification

1. Design

The SWING Configuration Manager Editor provides a simple SWING-based editor for modifying configuration manager properties. This first version of the component will support drop-downs, radio buttons, free text entry, and checkboxes for boolean values. The component loads the configuration values for a specific namespace, displays the form, and allows the user to edit the values and commit them back to the configuration file. This component will be used in the UML Tool to allow the user to modify configuration values in the tool, as opposed to forcing them to edit a file.

The design provides the main panel class. This panel retrieves the properties from configuration using the PropertyDefinition instances. The specific JComponents, tied to the properties, are generated using the related generators, based on the type. This is a flexible mode to create other component generators in the future (RS requires a flexible mode for the properties types in 1.2.3). I didn't use the ObjectFactory because this component will be used inside TC Tool and if a property type will be created then the construction of the new concrete implementation will be added directly in the constructor , for also an easy development of this component.
The component defines the validators that are used also to save the properties. The properties , only the modified properties, are saved when the save method is called, saved with Configuration Manager using the specific namespace.
If I want to hide (if I want to not use) a property of the namespace then it's sufficient to not include it in the properties definitions.
The font and resize configuration parameters are extended from the JPanel class.

1.1 Design Patterns

1.1.1 Strategy

The editor panel uses strategically the validators and the component generators.

1.2 Industry Standards

None

1.3 Required Algorithms

1.3.1 Initialize the Editor Panel: initialize() method

This is the algorithm which the initialize() method will implement to initialize the editor panel:

```
// create the grid bag layout
GridBagLayout gridBagLayout = new GridBagLayout();
this.setLayout(gridBagLayout);

// create the index for the cycle
```

```
int index = 0;

for (final PropertyDefinition propertyDefinition :
propertyDefinitions.keySet()) {
    // create the label related to the prompt text
    JLabel label = new JLabel(propertyDefinition.getPromptText());

    // create the constraints for the label
    GridBagConstraints c = new GridBagConstraints();
    c.gridx = 0;
    c.gridy = index;
    c.fill = GridBagConstraints.BOTH;
    c.anchor = GridBagConstraints.LINE_START;
    // also set the c.insets,c.ipadx and c.ipady constraints: use the
    // values of rowsSpacing and columnsSpacing
    c.ipadx = columnsSpacing;
    c.ipady = rowsSpacing;
    c.insets.bottom = c.insets.top = rowsSpacing;
    c.insets.left = c.insets.right = columnsSpacing;

    // add the component to the grid
    gridBagLayout.addLayoutComponent(label, c);
    add(label);

    // now the column is the second
    c.gridx = 1;

    // create the component from the component generator, and add it to
    // the property definitions
    JComponent component =
this.componentGenerators.get(propertyDefinition.getPropertyType())
            .generateComponent(propertyDefinition);
    this.propertyDefinitions.put(propertyDefinition, new JComponent[]
{ label, component });
    add(component);
    // add the component to the layout manager
    gridBagLayout.addLayoutComponent(component, c);
    index++;
}
```

1.3.2 *CheckBoxPropertyComponentGenerator#generateComponent*

This method generates the checkbox from the property definition:

```
// create the check box
final JCheckBox checkBox = new JCheckBox();

// set the property value of property to CheckBox
checkBox.setSelected(Boolean.parseBoolean(propertyDefinition.getProperty
Value()));

// add the listener to CheckBox
checkBox.addItemListener(new ItemListener() {
    /**
     * if the CheckBox is selected then the property definition will be
     * "true" else will be "false".
     *
     * @param e event.
     */
    public void itemStateChanged(ItemEvent e) {
       if (e.getStateChange() == ItemEvent.SELECTED) {
          propertyDefinition.setPropertyValue(Boolean.toString(true));
       } else if (e.getStateChange() == ItemEvent.DESELECTED) {
          propertyDefinition.setPropertyValue(Boolean.toString(false));
       }
    }
});

return checkBox;
```

1.3.3 *DropDownPropertyComponentGenerator#generateComponent*

This method generates the combobox (multiple choices) from the property definition:

```
// get the value choices
Set<String> valueChoices = propertyDefinition.getValueChoices() == null ?
new HashSet<String>()
       : propertyDefinition.getValueChoices();
String propertyValue = propertyDefinition.getPropertyValue();
if (propertyValue != null && !valueChoices.contains(propertyValue)) {
   throw new IllegalArgumentException("value choices does not contain
property value:" + propertyValue);
}

String[] items = valueChoices.toArray(new String[valueChoices.size()]);

// construct the ComboBox with the choices
final JComboBox comboBox = new JComboBox(items);
// must be not edit-able
comboBox.setEditable(false);
// set the property value as selected
comboBox.setSelectedItem(propertyValue);

// add the listener
comboBox.addActionListener(new ActionListener() {
    /**
     * change the property definition with the selected item.
     *
     * @param e event.
     */
    public void actionPerformed(ActionEvent e) {
       String selectedItem = (String) comboBox.getSelectedItem();
       propertyDefinition.setPropertyValue(selectedItem);
    }
});

return comboBox;
```

1.3.4 *FreeTextPropertyComponentGenerator#generateComponent*

This method generates the text field from the property definition:

```
// create the text field
final JTextField textField = new
```

```
JTextField(propertyDefinition.getPropertyValue());
// add the listener
textField.getDocument().addDocumentListener(new DocumentListener() {
   // change the property definition with the text
   public void changedUpdate(DocumentEvent e) {
      propertyDefinition.setPropertyValue(textField.getText());
   }

   // change the property definition with the text
   public void insertUpdate(DocumentEvent e) {
      propertyDefinition.setPropertyValue(textField.getText());

   }

   // change the property definition with the text
   public void removeUpdate(DocumentEvent e) {
      propertyDefinition.setPropertyValue(textField.getText());

   }
});

return textField;
```

1.3.5 *RadioButtonsPropertyComponentGenerator#generateComponent*

This method generates the radio buttons (multiple choices) from the property definition:

```
// Check propertyDefinition is not null.
if (propertyDefinition == null) {
   throw new IllegalArgumentException("Parameter propertyDefinition cannot
be null.");
}
// Check propertyDefinition.type equals to PropertyType.RADIO_BUTTONS
if
(!propertyDefinition.getPropertyType().equals(PropertyType.RADIO_BUTTONS
)) {
   throw new IllegalArgumentException(
         "propertyDefinition.propertyType should be
PropertyType.FREE_TEXT. But it is "
                + propertyDefinition.getPropertyType());
}

// get the value choices
Set<String> valueChoices = propertyDefinition.getValueChoices() == null ?
new HashSet<String>()
        : propertyDefinition.getValueChoices();
String propertyValue = propertyDefinition.getPropertyValue();
if (propertyValue != null && !valueChoices.contains(propertyValue)) {
   throw new IllegalArgumentException("Value choices does not contain
property value: " + propertyValue);
}

// create the panel which contains the radio buttons
JPanel radioButtonsPanel = new JPanel();
// set the layout
radioButtonsPanel.setLayout(new BoxLayout(radioButtonsPanel,
BoxLayout.PAGE_AXIS));

// create the group of buttons
ButtonGroup buttonGroup = new ButtonGroup();

// create a button for each value choice
for (final String choice : valueChoices) {
   JRadioButton radioButton = new JRadioButton(choice);

   // if the property value is equal to the choice value then the
   // button will be selected
   if (choice.equals(propertyValue)) {
      radioButton.setSelected(true);
   }

   // add the action listener
   radioButton.addActionListener(new ActionListener() {
      /**
       * if the radio button is selected then the property definition
       * will have the value of the radio button.
       *
       * @param e event.
       */
      public void actionPerformed(ActionEvent e) {
         propertyDefinition.setPropertyValue(choice);
      }
   });

   buttonGroup.add(radioButton);
   radioButtonsPanel.add(radioButton);
}

return radioButtonsPanel;
```

1.4 Component Class Overview

PropertyValidator
It's the validator used to validate the properties when the properties are saved or simply when the properties are validated. Used in the editor panel, there are no implementations.
Thread safety: the thread safety is required

ValidationResult
Represent the validation result of a property validation: it contains the information of the fail or success.
Thread safety: it's thread safe, its' immutable

ConfigurationManagerEditorPanel
This is the main class of this component. It uses the PropertyDefinition instances to define the properties and the related jcomponents. These jcomponents are constructed with the related generators using the type. Many other graphic values are configurable. The component is initialized in the initialize method called in the constructor. The

component uses the validators to validate the properties against the old properties. The new properties values are saved using the Configuration Manager, (the old properties values are retrieved using ConFigurationManager)
Thread safety: this class is not thread safe but it's not required because only a single thread will use this class, this TC Tool application and then the
user who uses the application.

PropertyComponentGenerator
this interface defined the contract to generate the jcomponent based on the type. The JComponent will be tied to the propertyDefinition: when the component will change its state, the property value will change (for example in a text field, when the user changes the text, the property definition must reflect this change). It is used inside the {ConfigurationManagerEditorPanel.
Thread safety: the implementations are required to be thread safe

PropertyType
Represent the type of a property definition. Each type is tied to a specific jcomponent. It is used to define the related jcomponents in the editor panel. Ignore the type of the fields: this is a Java 5 enum, I only shot it for the TC Tool.
Thread safety: it's an enum it's thread safe.

PropertyDefinition
Represents the property definition. This class defines a property with the name in the configuration manager (this is like an attribute in ConfigurationManager). The value will be loaded from Configuration Manager in the editor panel.
Thread safety: it's not thread safe, it's mutable

RadioButtonsPropertyComponentGenerator
This generator will generate a JPanel. this JPanel contains the radio buttons grouped using a ButtonGroup. When the user will click on the radio, the property value will be modified with the value of the radio.
Thread safety: it's stateless, then it's thread safe

CheckBoxPropertyComponentGenerator
This generator will generate a JCheckBox related to the property definition. When the user will check or uncheck the checkbox, the property's value will be "true" or "false".
Thread safety: it's stateless, then it's thread safe

FreeTextPropertyComponentGenerator
This generator will generate a JTextField related to the property definition. When the user will edit the text in the textFile, the propertyDefinition's
property value will be modified with this text
Thread safety: it's stateless, then it's thread safe
DropDownPropertyComponentGenerator
This generator will generate a JComboBox related to the property definition. When the user will select the item of combobox, the property's value

will be modified with the value of the item.
Thread safety: it's stateless, then it's thread safe

1.5 Component Exception Definitions

InsufficientDataException
This exception is thrown if there is no sufficient data to perform the validation

ValidationException
This exception is thrown if some errors occur in the validation of properties definitions

SavingException
This exception is thrown if some errors occur during the save method, when the properties are saved.

ConfigurationManagerEditorPanelException
This exception is the parent exception of all exceptions in this class

ConfigurationException
This exception is thrown if the configuration is bad: missing required properties or in the bad format

NotValidNamespaceException
This exception is thrown if the namespace used to load the properties is not valid: it doesn't contain properties or it is missed in the configuration.

1.6 Thread Safety

The component is not thread safe, but the thread safety is not required, The main class is not thread safe but it's not required because only a single thread will use this class, this TC Tool application and then the user who uses the application. The user can't use the same instance in different frames in parallel mode, so the thread safety is not applicable in this case of an user interface.
The component generators are required to be thread safe because it's possible that the instance will be used also in other places, even tough the thread safety in this component is not required.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

- Configuration Manager 2.1.5
  - Used to configure the main class and also to load and save the properties
- Base Exception 2.0
  - TopCoder standard for all custom exceptions.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Names

com.topcoder.gui.configurationmanager
com.topcoder.gui.configurationmanager.componentgenerators

3.2 Configuration Parameters

3.2.1 ConfigurationManagerEditorPanel

Field name	**Description**	**Details**
propertiesNamespace	The namespace used to load the properties Required	A valid namespace String not empty and must not be empty (must contain properties)
propertyDefinitions	This is a property that contain other sub properties, these sub-properties are multi-values properties. The sub-properties are defined: The key is name of property, the first value is the property type, the second value is the prompt text. If it's a multiple choices type then the following values are the possible values	This property must contain sub-properties. The keys must be String not empty. The property type must be the name of an enum values in the PropertyType enum (for example “FREE_TEXT”). The prompt Text must be a String not empty. The following values are the possible values in the case of a multiple choices type.

	Required, it's used to instantiate the PropertyDefinitions	
rowsSpacing	The spacing among the rows in the editor panel Optional	Int >=0
columnsSpacing	The spacing among the columns in the editor panel Optional	Int >=0
fontName	The name of Font, to construct the Font instance Optional	A String not empty, valid Font name
fontStyle	The style of font, o construct the Font instance Optional	Constant in the Font class, String not empty
fontSize	The size of font, o construct the Font instance Optional	int>=0
fontColor	The name of constant in the Color class (case insensitive) Optional	Constant in the Color class, String not empty

3.3 Dependencies Configuration

3.3.1 TopCoder dependencies

All the dependencies are to be configured according to their component specifications.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow the dependencies.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

None

4.3 Demo

4.3.1 Setup

This is the configuration of the class:

```
<Config name="ConfigurationManagerEditorPanel">
```

```
	<!-- Define the namespace used to load the old values of properties
and to save the properties -->
	<Property name="propertiesNamespace">
		<Value>MyClass</Value>
	</Property>

	<!-- Define the properties definitions -->
	<Property name="propertyDefinitions">

		<!-- Defines a text field -->
		<Property name="name">
			<Value>Name :</Value>
			<Value>FREE_TEXT</Value>

		</Property>

		<!-- Defines a combobox with multiple values -->
		<Property name="size">
			<Value>Size :</Value>
			<Value>DROP_DOWN</Value>
			<Value>1</Value>
			<Value>10</Value>
			<Value>20</Value>
		</Property>

		<Property name="checked">
			<Value>checked :</Value>
			<Value>CHECK_BOX</Value>
		</Property>
		<Property name="radios">
			<Value>radios :</Value>
			<Value>RADIO_BUTTONS</Value>
			<Value>1</Value>
			<Value>10</Value>
			<Value>20</Value>
		</Property>
	</Property>

	<Property name="rowsSpacing">
		<Value>10</Value>
	</Property>

	<Property name="columnsSpacing">
		<Value>20</Value>
	</Property>

	<Property name="fontName">
		<Value>Times New Roman</Value>
	</Property>

	<Property name="fontStyle">
		<Value>0</Value><!-- Plain style -->
	</Property>

	<Property name="fontSize">
		<Value>10</Value>
	</Property>

	<Property name="fontColor">
		<Value>black</Value>
	</Property>

</Config>
```

This is the namespace which contains the properties to modify:

```
<?xml version="1.0"?>
<CMConfig>
	<Property name="name">
		<Value>John</Value>
	</Property>
	<Property name="size">
		<Value>10</Value>
	</Property>
	<Property name="checked">
		<Value>true</Value>
	</Property>
	<Property name="radios">
		<Value>10</Value>
	</Property>
</CMConfig>
```

4.3.2 Usage

```
final ConfigurationManagerEditorPanel panel = new
ConfigurationManagerEditorPanel(
       "ConfigurationManagerEditorPanel");

JButton ok = new JButton("OK");
panel.add(ok);
ok.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {
        // the properties are validated
        try {
            // validate the properties
            ValidationResultSet resultSet = panel.validateProperties();
            // check the validation
            if (resultSet.isSuccess()) {
                // save the properties
                // the validation is already performed since I can call
                // directly the
                // save method
                panel.saveProperties();
                System.out.println("saving");
                // the properties now are saved to the related namespace
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
});
panel.setSize(400, 400);
panel.setVisible(true);
```

```
// add it to the frame
frame.getContentPane().add(panel);
frame.pack();
frame.setSize(400, 400);
frame.setVisible(true);
```

The result:

```
<?xml version="1.0"?>
<CMConfig>
      <Property name="name">
            <Value>Michael </Value>
      </Property>
      <Property name="size">
            <Value>20</Value>
      </Property>
      <Property name="checked">
            <Value>true</Value>
      </Property>
      <Property name="radios">
            <Value>10</Value>
      </Property>
</CMConfig>
```

5. Future Enhancements

Other component generators can be added