

This page last changed on Jun 12, 2008 by [ghostar](#).

1. Scope

1.1 Overview

The SWING Configuration Manager Editor will provide 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 will load the configuration values for a specific namespace, will display the form, and will allow 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.

1.2 Version

1.0

1.2 Logic Requirements

1.2.1 JPanel

The main class of the component must extend from JPanel, allowing the user to place the editor inside of an existing window or JFrame. The component must support multiple instances inside the same JFrame or application, allowing for the display and editing of values from multiple different configuration namespaces.

1.2.2 Namespace Specification

The user must provide the namespace to load into the SWING control. If the namespace doesn't exist or isn't loaded, an exception should be thrown.

1.2.3 Property specification

The user must be able to provide a prompt text and property type for each property value to be edited. This must be available in the API through setter methods. For instance, in the Setting_Config.xml file for the UML tool, the "BackupDirectory" property would be specified with a prompt of "Backup Directory:" and a type of "FreeText". The following property types should be supported:

- FreeText: Displayed as a text field, allowing free text input
 - These values will be written back to the configuration file as the text input
- RadioButtons: Multiple-choice, displayed as radio buttons
- DropDown: Multiple-choice, displayed as a drop down box (combo-box)
- CheckBox: Boolean, displayed as a check box
 - These values will be written back to the configuration file as text, either "true", or "false"

The design must be extensible enough to support future property types. If a property is specified that doesn't have a matching configuration value in the given namespace, that properties prompt and corresponding edit control should not be shown.

1.2.4 Multiple-choice properties

For the RadioButtons and DropDown choices, the user will be expected to provide a list of string values indicating the various options. For instance, in the Setting_Config.xml file for the UML Tool, there is a "FontSize" property value. The user could specify this value with a prompt of "Font Size:", a type of "DropDown", and the following options: "8", "9", "10", "12", "14", "18". The options would be available in a drop down. If the user chose a type of "RadioButtons" instead, all options would be visible, but the user would only be able to choose 1. The user must be required to provide at least one option value for these properties, or an exception should be thrown.

1.2.4 Hidden properties

The user must be able to hide properties from being displayed in the form, even if they are available in the namespace. For instance, in the `Setting_Config.xml` file for the UML Tool, it contains values for "Width", "Height", "LastOpenProject", and "LastImageExportPath", which are set automatically by the tool and shouldn't be edited directly. The component should support not showing those values in the form generated. If the values are hidden, they shouldn't be modified in any way when the form data is saved. Any properties that aren't set up with prompts and property types in the main class should be considered hidden.

1.2.5 Current value loading

When the form is first loaded, the current configuration values should be loaded and used to populate the form. This component can assume that the namespace provided has already been loaded in the `ConfigManager`. If the namespace provided isn't available or contains no properties, an exception should be thrown.

1.2.6 Save method

The component must have a "save" method that is used to save the changed values to the configuration file. This method will be tied into a "Save" button in the GUI. It is expected that the user will need to restart the application to apply the configuration changes, at least in this initial version.

1.2.7 Pluggable validation

This component must offer a way to validate the values in the form before saving, to ensure the values are valid for each property. This component must offer a "validate" method and a validator interface that can be implemented by the custom validators. How the validators are assigned is up to the designer. They can either be assigned via setters, or they can be passed directly to the "validate" method.

1.2.8 Resizing

The layout of the generated form should be in two columns, one for the prompt text, and the other for the actual controls. On a resize, the controls with a width, like a drop down or text box should expand to use the newer space, or expand to use less space. The user must be able to set the size of the component, and it must show scrollbars if necessary.

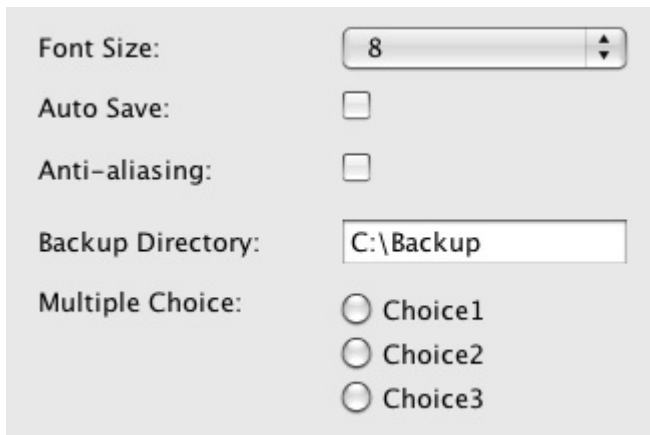
1.2.9 Look and feel

The user must be able to set the following properties for the controls:

- Vertical spacing between each property value
- Spacing between the prompt column and value control column
- Font name, size, and color for prompts and controls
 - Font specification can be for the entire control, but as an optional enhancement, can be made granular for specific properties.

1.2.10 Sample output

The output of the component should look similar to this, generated for the `Setting_Config.xml` file of the UML Tool. It would be preferable, but not required, to have the generation of the GUI be extensible, allowing for the output to be easily changed in the future.



Font Size: 8

Auto Save: ☐

Anti-aliasing: ☐

Backup Directory: C:\Backup

Multiple Choice: ☐ Choice1
☐ Choice2
☐ Choice3

1.3 Required Algorithms

- How the control is built and displayed must be coherently described

1.4 Example of the Software Usage

This component will be used in a configuration panel window for the UML Tool, allowing the user to modify the configuration values in a GUI.

1.5 Future Component Direction

- Different property types will be necessary, including file paths
- Pagination for large amounts of properties
- Internationalization support

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

- SWING will be used for the controls

2.1.2 External Interfaces

None

2.1.3 Environment Requirements

- Development language: Java 1.5
- Deployment: Java 1.5

2.1.4 Package Structure

com.topcoder.gui.configurationmanager

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- None



3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

SWING

3.2.2 TopCoder Software Component Dependencies:

- Configuration Manager 2.1.5 http://software.topcoder.com/catalog/c_component.jsp?comp=500004&ver=9

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

3.2.3 Third Party Component, Library, or Product Dependencies:

None Allowed

3.2.4 QA Environment:

- Windows XP / Vista
- Mac OS X 10.4+

3.3 Design 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.

3.4 Required Documentation

3.4.1 Design Documentation

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

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of the TC UML Tool.