Configuration Manager Redesign Requirement Specification

1. Scope

1.1 Overview

Software applications typically manage application level configuration details in “properties” or “ini” files. In the case of large applications or applications composed of distinct software components, there may be numerous configuration files each bound to a particular functional component.

The purpose of this design competition is to redesign the existing configuration manager component to resolve three limitations. The new design should not change any existing interface or break any existing functionality.

1.1.1 Version

2.2

1.2 Logic Requirements

Currently configuration manager component has some limitations. For this competition, competitors need to redesign this existing component, and conquer these limitations. You can refer to the existing design specification for new redesign work. Configuration manager can be found in http://software.topcoder.com/catalog/c_component.jsp?comp=500004. Generally, there are three limitations to resolve below. The new design should meet all these requirements in the previous requirement spec.

1.2.1 Resolve Thread Safety Limitation

Configuration Manager is to load config files for the application. On every read of a file, though, it executes a refresh of the file – that is, it reads it again from the file system (see DefaultConfigManager::refresh). If two threads are trying to access that file at the same time, one might catch the file right at the point where the other is refreshing it, and it then thinks the file doesn't exist. It then removes that file's namespace from memory, and thus the component can never read that file again until the server is restarted. There is one example illustrating this issue. See cm_threadunsafe_example.jar.

1.2.2 Refresh namespaces configured to be refreshed

Currently configuration manager component supports refreshing all namespaces or specified namespace (see **DefaultConfigManager::refresh** and **DefaultConfigManager::refreshAll**). Some applications might require namespace refreshing, and some might not. And for performance consideration, we expect namespace can be configurable (whether it really requires refreshing during namespace refreshing). When one namespace is configured to be not refreshed, refreshing the namespace does nothing. 'All on' and 'all off' options should be supported in the new configuration manager component.

1.2.3 Make implementation class of Configuration Manager configurable

Currently configuration manager component uses the implementation class **DefaultConfigManager** to generate the configuration manager instance. In the new configuration manager, make the implementation class configurable. The javadocs of the component indicate that a field called 'implementor' can be used to specify which subclass of **ConfigManager** to use (see **ConfigManager::getInstance**). It was never actually implemented though.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The Configuration Manager will be used to manage all configuration files within an application. For instance, an application that has been built from many different

components will need to manage a configuration file for each distinct component. This tool will enable developers to use one common interface to access and retrieve configuration parameters in a thread safe way.

1.5 Future Component Direction

None

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

None

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.topcoder.util.config

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- Whether one namespace requires refreshing during refreshing operation is configurable.
- Implementation class of configuration manager is configurable.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

N/A

3.2.2 TopCoder Software Component Dependencies:

Do Not Include Any TopCoder Generic components in this design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None

Please Note: Any use of Open Source packages must first be authorized.

3.2.4 QA Environment:

- Jboss 1.5
- Java 1.5

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 TCUML Tool.