Configuration API 1.0 Requirements Specification

1. Scope

1.1 Overview

Currently the configuration object is defined by the Configuration Manager (CM) component. In practice many TopCoder components load their own configurations (based on namespace) from CM. This has created problems using customer CM implementations and homogenizing configuration practices across an application.

In a new paradigm the configuration model will be implemented by a separate component from the configuration manager component, which will be primarily concerned with file system operations and other implementation details. This component will define the API for a configuration interface that components can rely on without being coupled to the Configuration Manager component.

1.2 Logic Requirements

1.2.1 Configuration Object API

- The API should be as simple and easy to use as possible without sacrificing functionality.

1.2.2 Configuration Property API

- Mapping of property key to value(s).
- Retrieval of all property keys.
- CRUD operations on properties via keys or values.

1.2.3 Configuration Nesting API

- Mapping of nested configuration objects by keys.
- Retrieval of all nested configuration objects by keys.
- CRUD operations.
- Tree based actions.
    - Ability to find descendants through key aggregation (a list of successive child keys and/or wildcards that form a path).
    - Ability to take arbitrary actions across multiple descendants based on key aggregation.

1.2.4 Base Abstract API Implementation

- Provide a base abstract implementation of the API where every method throws an exception, to allow for easy creation of subclasses.

1.2.5 Default Configuration Object API Implementation

- Provide a default implementation for the API. This implementation should extend the base abstract API implementation and define all methods.
- The default implementation should not be thread safe, but it should provide a parallel thread safe wrapper.
- The default implementation should be serializable.
- The default implementation should be cloneable (with clone defined as a deep copy).

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

A single application might directly use 10 or 15 components. But these components in turn might use another 10 or 15 components. In order to consolidate configuration management into a single location, but without adding too much complexity, the application would load the application specific configuration parameters from the root configuration object. Additionally each of the directly accessed

components that require configuration parameters, can have it's configuration nested in the application's configuration object. The application can pass in each component specific configuration by retrieving the child configuration objects and calling the component constructors. In turn these components can configure their dependant components in the same manner.

Only the portion of the application that loads the initial configuration object will have to know how the configuration management is handled. This will allow for CM decisions to be deferred to the application layer, instead of being dependant on each individual components design and implementation.

1.5 Future Component Direction

None; the configuration

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

None.

2.1.3 Environment Requirements

- Development language: Java1.4
- Compile target: Java1.4

2.1.4 Package Structure

com.topcoder.configuration

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?

None.

3.2.2 TopCoder Software Component Dependencies:

None and should not use any unless requested

**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.

3.2.4 QA Environment:

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

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 Poseidon.