Requirements 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 the Configuration Manager is to centralize the management of and access to these files. The benefits of the Configuration Manager include:

- A centralized code base for reading and loading configuration files.
- A system wide caching strategy of configuration details to limit performance bottlenecks.
- A common interface for accessing and updating application properties.

Potential uses of the Configuration Manager include:

- Retrieve configuration properties for TopCoder software components
- Configure other Java components (including beans). As an example, consider using Configuration Manager for configuring log4j, itself a component with its own extensive configuration management.

1.2 Existing Functionality

1.2.1 File Loading

- Ability to preload a list of configuration files.
- Ability to add additional configuration files once the configuration manager has started.
- Ability to reload configuration files into memory when properties are modified.
- Ability to refresh the Configuration Manager if configuration files are modified. The Configuration Manager does not poll the source file for changes.

1.2.2 File Types

- Support for plain text property files
- Support for xml-based configuration files based on the Configuration Manager DTD.
- Support for an xml file configuration file that supports multiple namespaces.

1.2.3 Configuration Manager Access

- Only one instance of the configuration manager should exist in the JVM.
- The Configuration Manager will be accessed from numerous clients simultaneously.

1.2.4 Properties

- Many configuration files may contain the same property names.
- Properties may have multiple values.

1.2.5 Administration API

- Addition and deletion of namespaces.
- Addition of property names and values.

1.3 Enhancements

1.3.1 Additional Functionality

- Provide a pluggable data source. The current version only supports file-based configuration in the form of .properties or .xml. An example of an additional data source is a RDBMS.
- Maintain file integrity. When modifying a configuration file through the configuration manager, the order of the underlying file is not currently maintained. This also includes maintaining the order of comments.
- Elaborate upon the existing javadocs, particularly the ConfigManager class. Component usage and configuration should be fully explained in the javadocs. (See java.util.Date The details of how to use this class are fully detailed in the javadoc.)
- Migrate to the latest directory structure as defined in the distribution file. This includes moving test cases to the new package structure.
- Update the design documents to .zargo. The design submission should provide all documentation as defined in the deliverables document, even if it does not exist.
- The API should support the addition and deletion of namespaces, property names and property values.
- Properties currently have one or more values. The component should be extended to support nested properties.
  For example: A country property may have many nested countries. Each country may have multiple properties, each with multiple values

```
+Country
  +US
    +NAME
      +USA
      +United States of America
    +Currency
      +NAME
        +dollar
      +Symbol
        +$
  +France
  +Germany
```

- The existing Configuration Manager defines an interface to obtain namespace and property details from components that use the Configuration Manger. Define an additional interface for the core Configuration Manager itself. This interface should define all of the methods required to implement a custom Configuration Manager.
- Modify the current design of the Configuration Manager to return the newly defined Interface when an instance of the Configuration Manager is requested. This will allow the implementation of the Configuration Manager to be changed without modifying client-side code. More importantly, it will enable customers to use Configuration Manager-dependent TopCoder components without requiring TopCoder's implementation of the Configuration Manager.
- The implementation class to be instantiated should be read from the Configuration Manager configuration file at startup. The implementation cannot be changed during runtime.

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.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

None specified.

2.1.3 Environment Requirements

- Development language: Java 1.4
- Compile target: Java 1.4

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?

- Namespaces to load on startup.
- Value separator character for property files with properties that have multiple values.

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.
**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

3.3 Design Constraints

3.3.1 Development Standards:

- It is required that all code be clearly commented.
- All code must adhere to javadoc standards, and, at minimum, include the @author, @param, @return, @throws and @version tags.
    - When populating the author tag, use your TopCoder member handle. In addition, please do not put in your email addresses.
    - Copyright tag: Copyright © 2002, TopCoder, Inc. All rights reserved

- For standardization purposes, code must use a 4-space (not tab) indentation.
- Do not use the ConfigurationManager inside of EJB's. The usage of the java.io.* package to read/write configuration files can potentially conflict with a restrictive security scheme inside the EJB container.
- All code must adhere to the Code Conventions outlined in the following: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

3.3.2 Database Standards:

- Table and column names should be as follows: table_name, not tableName.
- Every table must have a Primary Key.
- Always use Long (or database equivalent) for the Primary Key.
  For maximum portability, database objects (table names, indexes, etc) should be kept to 18 characters or less. Informix and DB2 have an 18-character limit.

3.4 Required Documentation

3.4.1 Design Documentation

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

3.4.2 Help / User Documentation

- Javadocs must provide sufficient information regarding component design and usage.