.NET Configuration Manager 2.0 Requirements Specification

1. Scope

1.1 Overview

.NET libraries do not currently support automatic loading of configuration data. In the instance of a library dll used in numerous applications, adding additional configuration details to all of the applications' configuration files is not optimal. The purpose of the Configuration Manager is to centralize the management of, and access to, this data.

1.2 Logic Requirements

1.2.1 Remove the System-Wide Singleton

Security changes to remoting in the .NET Framework version 1.1 have broken the current system-wide singleton implementation. The system-wide singleton should be replaced with a normal singleton.

1.2.2 Extend the concept of "Config Files" to include other data sources.

The current plug-in architecture for this component only supports configuration sources which are files. It is desirable to allow plug-ins for other data sources, such as a database (like the Java Configuration Manager DB component.)

- It should be possible to load data from any combination of data sources into a common configuration namespace.
- Existing configuration files should continue to work unmodified with the new Configuration Manager, but the formats may be extended to support additional data sources.
- Preloading must continue to be handled as it is in the currently version: An application must define a path to a preload configuration file under the "TopCoder.Util.ConfigurationManager.Preload" key in its app.config file.

1.2.3 Improve the component interface.

Since this component is used in so many other components, it is important that the interface be as simple and powerful as possible. The following are some common usage patterns for this component which could be improved:

- Allow the developer to set parsing preferences such as if values should be trimmed before processing, if null/empty values should be allowed, and if an exception should be thrown if a property is requested that does not exist and/or is empty, etc. Different components within the same application should be able to maintain their own preferences.
- Allow developers to easily retrieve properties as types other than string (such as bool, int, double, Enum, etc.) An application should be able to request a parameter as a specific datatype and the Configuration Manager should ensure that the property's value is parsable as that type before returning it (and then return it in a typesafe way, if possible.)
- A common usage of the Configuration Manager is to determine a runtime type of a configurable object. To simplify this procedure, the component should be able to instantiate objects based on the typename found in a configuration property. An application should be able to specify a required base type or interface and the Configuration Manager will enforce this type dependency when determining if the requested property is valid.

1.2.4 Retain API compatibility.

The new API must be backwards compatible with the 1.1 interface. Since API changes may be useful in order to remain consistent with the new features in this version, compatibility may be achieved by simply providing deprecated wrapper methods/classes for any parts of the API which

are provided simply for the sake of compatibility.

- Any classes or methods which remain in the design only for compatibility reasons should be clearly marked as such in the design and annotated with the ObsoleteAttribute in the code.

1.2.5 Improve the Documentation.

Since this component is used in so many other components, it is critical that the "Installation and Configuration" and "Usage Notes" sections of the component specification be clear, concise, and correct. Reviewers must ensure that this documentation is sufficient to easily get the Configuration Manager setup and usable for other components.

In addition, the documentation should STRONGLY SUGGEST that components which use the Configuration Manager NOT hard-code their configuration namespaces. A default namespace may be used, but it should be possible to change this default at runtime by passing a different configuration namespace to that component at runtime. This should allow a component, such as an updated version of the Logging Wrapper which may be used by several other components in the same application, to use different configuration parameters each time it is used. (For example, the database portion of an application may wish to log message directly to its database, while the front-end portion of the application must log messages to a file. If both use the Logging Wrapper, there must be a way for different configuration parameters to be specified for each usage.)

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

A library for sending emails requires configuring the SMTP server address and port. A web application using this library does not want to configure these details in the web.xml.

1.5 Future Component Direction

In the future, additional plug-ins will be developed to support other sources of configuration data such as a database or the .NET app.config file.

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: C#

2.1.4 Namespace

TopCoder.Util.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?

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:

- Windows 2000
- Windows Server 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

XML documentation must provide sufficient information regarding component design and usage.