Requirements Specification

1. Scope

1.1 Overview

The Logging Wrapper component provides a standard logging API with a plugable back-end logging implementation. Utilization of the Logging Wrapper insures that components are not tied to a specific logging solution. More importantly, a change to the back-end logging solution does not require a code change to existing, tested components. Support exists for log4j and java1.4 Logger as back-end logging implementations.

The Logging Wrapper 1.2 project will enhance certain areas of the component.

1.2 Logic Requirements

1.2.1 Exceptions

Catch all exceptions even runtime exceptions. Any errors, even runtime errors such as null pointers, should not effect the execution of the code attempting to do the logging. The logging wrapper should just swallow all exceptions.

1.2.2 Private Method

Change createInstance to be private.

1.2.3 Static get log helper

A static get log helper function would be nice. That way you do not have to code retrieving an instance and catching the exceptions associated with getting an instance every time. I would like a static getLog function that is exactly the same, as the getLog function that is in the LoggerFactory class now, except it would not throw any exceptions.
Example:

```
public class LogFactory ... {
  public static Log getLog() {
    try {
      return getInstance().getLog(DEFAULT_NAME);
    } catch (Exception e) {
      return null;
    }
  }
 ...
}
```

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The initial use of the Logging Wrapper component will be within other TopCoder Software components. This will allow TopCoder Software components to be plugged into an existing environment without requiring the additional configuration and implementation of a specific logging solution.

1.5 Future Component Direction

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

No Changes

2.1.3 Environment Requirements

- Development language: Java1.4
- Compile target: Java1.3, Java1.4

2.1.4 Package Structure

No Changes

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

No Changes

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

No Changes

3.2.2 TopCoder Software Component Dependencies:

No Changes

3.2.3 Third Party Component, Library, or Product Dependencies:

No Changes

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000

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
- Test Plan

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.
- Implementations of the design must clearly translate the design documentation into the corresponding language format (i.e. java -> javadoc, .NET -> XML). Source code documentation must be sufficient to enable component usage without additional documentation.