# Base Exception 2.0 Requirement Specification

## 1. Scope

### 1.1 Overview

The Base Exception component version 1.0 provides a generic exception class to facilitate the unified handling of exceptions and errors. This upgrade version is to include additional information and provide enhanced functionality in the base exception class to better facilitate problem diagnostics in production environments.

### 1.2 Logic Requirements

*1.2.1 Additional Information*

1.2.1.1 The base exception should support an error code that can be used by applications for problem diagnostics.

1.2.1.2 The base exception should support a module code indicating the location of the exception.

1.2.1.3 The base exception should support additional runtime information, such as the date and time and the running thread the exception occurred on.

1.2.1.4 The base exception should also support user-defined information, for example in key/value pairs.

*1.2.2 Logging*

1.2.2.1 The base exception should facilitate logging of the exceptions by providing a transient flag that an application can use for indicating whether the exception has been logged or not.

1.2.2.2 The actual logging is not part of this component.

*1.2.3 Exception Messages*

1.2.3.1 The base exception should support retrieving exception messages in ResourceBundles, so that exception messages can be stored in ResourceBundles to ease the localization effort.

*1.2.4 Critical and Non-critical Exceptions*

1.2.4.1 The component should provide two derivations of the base exception to represent critical and non-critical exception that the applications can further derive from.

1.2.4.2 Critical exceptions are for problems that occur outside the control of software; e.g. a database being down or otherwise unavailable in a non-recoverable manner. They would typically require an administrator to intervene or at least be notified.

1.2.4.3 Non-critical exceptions are for problems that are under control of the software; e.g. an application receives data that is within the domain, but violates some dynamic constraint, and the application wants to pass the error up the layers. They are typically handled by the software without need to log for admin.

### 1.3 Required Algorithms

None.

### 1.4 Example of the Software Usage

An enterprise application requires sophisticated error handling to ease the support and maintenance. This component is used as the base of all the exceptions in the application, so that logging of the exceptions is handled in a unified fashion.

## 1.5 Future Component Direction

Other useful information might be added.

# 2. Interface Requirements

### *2.1.1 Graphical User Interface Requirements*

None.

### *2.1.2 External Interfaces*

None.

### *2.1.3 Environment Requirements*

- Development language: Java 1.4
- Compile target: Java 1.4 and Java 1.5

### *2.1.4 Package Structure*

com.topcoder.util.errorhandling

# 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:*

**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 9
- RedHat Linux Enterprise 4
- Windows XP, 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.