DB Connection Factory Requirements Specification

1. Scope

1.1 Overview

The DB Connection Factory component allows applications to obtain JDBC Connections without knowing the underlying way that these connections are getting created. In Java, connections are most often obtained either via a DataSource or via the DriverManager class. This component abstracts that logic so that the application does not have to know about the details of creating a connection, and so that those details can be changed without modifying application code.

Version 1.1 adds the ability to obtain connections having user-provided authentication credentials, and uses the TopCoder Configuration API component instead of relying directly on Configuration Manager. Changes and additions between versions 1.0 and 1.1 of these requirements are colored red.

1.2 Logic Requirements

1.2.1 Interface Definition

This component must implement the interface found in section 2.1.2.1. You may add functionality to this interface, you just can't remove any.

1.2.2 Name Mapping / Connection Configuration

The configuration of the DB Connection Factory will allow a user to define multiple connections and give them names.

A default connection will be configured so that the client does not need to know the name of a connection.

1.2.3 Exception handling

Any sort of exception that occurs creating a connection should be wrapped into a DBConnectionException. This will allow for consistent error handling in the client application no matter how the underlying implementation creates the connection.

The DBConnectionException should be extended from the TCS Base Exception component.

1.2.4 Pluggable Connection Sources

This component will support a pluggable connection source with initial plug-ins supporting the following two connection creation mechanisms:

- javax.sql.DataSource
- JDBC URL (java.sql.DriverManager)

1.2.5 User-Provided Credentials

The component will allow clients to obtain connections that are authenticated with user-provided credentials rather than credentials recorded in component configuration.

1.2.6 Configuration API

The component will provide means for its clients to configure it via a `ConfigurationObject` (from the Configuration API component) prepared by the client instead of via a Configuration Manager config file loaded by the component itself.

- The component will rely on the structure of the `ConfigurationObject` to be consistent with that of the root configuration object produced by using the Configuration Persistence component to load a Configuration Manager config file.
- The component will support both use of a component-defined default namespace and use of a client-specified namespace to select the appropriate configuration properties from the `ConfigurationObject`.

All component elements that depend on Configuration Manager will be marked deprecated.

1.2.7 Compatibility

Version 1.1 will be backwards-compatible with version 1.0.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

TopCoder components that use a database connection will now be able to use the DB Connection Factory component to create their SQL Connections. This will enable the components to change their connection types (Data Source or JDBC URL) in the future via a configuration change, rather than having to change and recompile code.

1.5 Future Component Direction

The integration of custom connection pools as a pluggable connection source is a possible future enhancement.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 Internal Interfaces

2.1.2.1 DBConnectionFactory

```
package com.topcoder.db.connectionfactory;

public interface DBConnectionFactory {

        /**
         * <p>Creates a default SQL Connection from the underlying implementation,
         * authenticated with the configured username and password, if any.</p>
         *
         * @return java.sql.Connection the SQL connection that is created by an
         * underlying implementation.
         * @throws DBConnectionException is thrown when there is a problem creating a
         * SQL Connection.
         */
        public java.sql.Connection createConnection() throws DBConnectionException;

        /**
         * <p>Creates an SQL Connection from an underlying implementation with
```

```
         * the specified name, authenticated with the configured username and
         * password, if any</p>
         *
         * @param name the name of the connection in the configuration file.
         * @return java.sql.Connection the SQL connection that is created by an
         * underlying implementation.
         * @throws DBConnectionException is thrown when there is a problem creating a
         * SQL Connection.
         */
        public java.sql.Connection createConnection(String name) throws DBConnectionException;

        /**
         * <p>Creates a default SQL Connection from an underlying implementation,
         * authenticated with the specified username and password.</p>
         *
         * @param username the username with which to connect to the database; may be null
         * @param password the name of the connection in the configuration file; may be null
         * @return java.sql.Connection the SQL connection that is created by an
         * underlying implementation.
         * @throws DBConnectionException is thrown when there is a problem creating a
         * SQL Connection.
         */
        public java.sql.Connection createConnection(String username, String password)
                throws DBConnectionException;

        /**
         * <p>Creates an SQL Connection from an underlying implementation with the specified
         * name, authenticated with the specified username and password.</p>
         *
         * @param name the name of the connection in the configuration file.
         * @param username the username with which to connect to the database; may be null
         * @param password the name of the connection in the configuration file; may be null
         * @return java.sql.Connection the SQL connection that is created by an
         * underlying implementation.
         * @throws DBConnectionException is thrown when there is a problem creating a
         * SQL Connection.
         */
        public java.sql.Connection createConnection(String name, String username,
                String password) throws DBConnectionException;
}
```

2.1.3 Environment Requirements

- Development language: Java 1.4
- Compile target: Java 1.4, Java 5.0

2.1.4 Package Structure

com.topcoder.db.connectionfactory

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- Connections – One or more connections will be configured with the following properties

    - name
    - pluggable connection source to use
        - Connection Parameters
- Default Connection – one of the connections needs to be the default connection for the component.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

JDBC

3.2.2 TopCoder Software Component Dependencies:

Base Exception 1.0

Configuration Manager 2.1.5 (deprecated)

Configuration API 1.0

**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, 10
- RedHat Enterprise Linux 4
- Windows XP, 2003, Vista

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.