Review Feedback Management 2.0 Component Specification

1. Design

This component defines a new entity called "Review Feedback" and provides basic CRUD operations for it.

In version 2.0 the review feedback entity is refined and separated into two entity classes. Now, the component also persists audit data for all modifications to the review feedback records.

1.1 Design Patterns

1.1.1 DAO

ReviewFeedbackManager is a DAO for ReviewFeedback.

1.1.2 DTO

ReviewFeedback and ReviewFeedbackDetail are DTO-s.

They are managed by ReviewFeedbackManager DAO. Please note that operations of this DAO work with the ReviewFeedback class, while the data for ReviewFeedbackDetail is accessed through ReviewFeedback entity.

1.1.3 Strategy

ReviewFeedbackManager interface (declared in this component) is expected to be used in a strategy context (external) by application. This component provides implementation (JDBCReviewFeedbackManager) of the mentioned interface.

JDBCReviewFeedbackManager uses Log, ConfigurationObject and DBConnectionFactory in a strategy context.

1.1.4 Factory

Factory methods like DBConnectionFactory.createConnection() and LogFactory.getLog() are used by this component.

JDBCReviewFeedbackManager.loadConfiguration() method, declared in this component, can also be viewed as a factory method. However, please note that this method actually delegates to ConfigurationFileManager.getConfiguration().

1.2 Industry Standards

JDBC, SQL, XML

1.3 Required Algorithms

1.3.1 Logging

The component will log all the errors, exceptions, warnings and debug information during its execution.

Logging is performed by CRUD methods of JDBCReviewFeedbackManager, via Logging Wrapper component (via Log instance stored in “log” class variable (or will not be performed if that variable is null).

Any error during logging will be silently ignored.

The following data will be logged:

- Date and timestamp.
- Method/action name.
- Description of a problem.
- Input arguments.

- Output results.

Format of log message is like below:

```
<YYYY-MM-DD hh:mm:ss> <Type> <Method/action name> <Message text>
```

Please note that message format is configurable via Logging Wrapper.

1.3.2 DDL

DDL scripts for tcs_catalog database are provided. It includes new tables: "informix".review_feedback, "informix".review_feedback_detail, "informix".review_feedback_audit, "informix".review_feedback_detail_audit.

Please see “tcs_catalog” folder inside the design submission.

Developers may not change the DDL without approval.

1.4 Component Class Overview

1.4.1 com.topcoder.management.reviewfeedback

AuditableEntity

This is a base class for auditable entities. It's a simply POJO, containing several properties (each with public getter and setter with no validation) and default empty constructor.

This class is mutable and not thread-safe.

IdentifiableEntity

This is a base class for identifiable entities. It's a simply POJO, containing an Id property (with public getter and setter with no validation) and default empty constructor.

This class is mutable and not thread-safe.

ReviewFeedback

Review feedback entity. It's a simply POJO, containing several properties (each with public getter and setter with no validation) and default empty constructor.

This class hold the general data about the review feedback: project ID, general comment, creation and modification authors and timestamps. This class will also contain a (possibly empty) list of associated ReviewFeedbackDetail instances.

It's used as DTO for ReviewFeedbackManager.

This class is mutable and not thread-safe.

#Changes in 2.0:

1) Poperties are updated according to the new data model.

ReviewFeedbackDetail

Review feedback detail entity. It's a simply POJO, containing several properties (each with public getter and setter with no validation) and default empty constructor.

This class defines review feedback for a specific reviewer: reviewer user ID, score and textual feedback.

This class is mutable and not thread-safe.

#Changes in 2.0:

1) Poperties are updated according to the new data model.

ReviewFeedbackManager

This is a DAO for ReviewFeedback. It provides CRUD operations for managing the ReviewFeedback entities in persistence.

Please note that ReviewFeedbackDetail entities, aggregated by ReviewFeedback entity, are also managed by this DAO. However, the interface of all CRUD operations of this DAO works with the ReviewFeedback class (the data for ReviewFeedbackDetail class will be accessed through ReviewFeedback entity).

Implementations are required to be thread-safe with assumption that caller uses method arguments thread safely.

#Changes in 2.0:

1) Documentation is updated to reflect the data model changes.

2) New "operator:String" argument is added to create() and update() methods in order to support auditing.

For details about changes in specific members, please refer to the documentaions of the members.

1.4.2 com.topcoder.management.reviewfeedback.impl

JDBCReviewFeedbackManager

This is a JDBC based implementation of ReviewFeedbackManager. It provides CRUD operations for managing the ReviewFeedback entities in persistence and performs them using SQL queries executed with aid of JDBC.

Please note that ReviewFeedbackDetail entities, aggregated by ReviewFeedback entity, are also managed by this DAO. However, the interface of all CRUD operations of this DAO works with the ReviewFeedback class (the data for ReviewFeedbackDetail class will be accessed through ReviewFeedback entity).

This class uses TC configuration components for configuration and owns a DBConnectionFactory used for obtaining DB connections.

Also it performs logging via TC Logging Wrapper component as per CS 1.3.1.

This class is immutable and thread-safe with assumption that caller uses method arguments thread safely.

#Changes in 2.0:

1) Documentation is updated to reflect the data model changes.

2) New "operator:String" argument is added to create() and update() methods in order to support auditing.

For details about changes in specific members, please refer to the documentaions of the members.

1.5 Component Exception Definitions

1.5.1 com.topcoder.management.reviewfeedback

ReviewFeedbackManagementConfigurationException

This runtime exception indicates error with component configuration.

This class inherits from not thread-safe class, so it's not thread-safe.

ReviewFeedbackManagementException

This exception indicates error with review feedback management. It's a top-level checked custom exception of the component.

This class inherits from not thread-safe class, so it's not thread-safe.

ReviewFeedbackManagementPersistenceException

This exception indicates error with persistence operation.

This class inherits from not thread-safe class, so it's not thread-safe.

ReviewFeedbackManagementEntityNotFoundException

This exception indicates error caused by fact that specified entity is not found in persistence.

This class inherits from not thread-safe class, so it's not thread-safe.

1.6 Thread Safety

The DAO, exposed by the component, is thread-safe with assumption that caller uses method arguments thread safely. The DTO, exposed be the component, is not thread-safe.

So, the component is thread-safe as long as caller uses ReviewFeedback, ReviewFeedbackDetail and ReviewFeedbackManager methods' arguments thread safely.

In version 2.0 thread safety is not affected. However, transaction handling is added to create(), update() and delete() operations.

2. Environment Requirements

2.1 Environment

Development language: Java 1.5

Compile target: J2SE 1.5, J2SE 1.6

2.2 TopCoder Software Components

Base Exception 2.0.0 – Used to support common TC exceptions inheritance.

Configuration API 1.1.0 – Used to configure the component.

Configuration Persistence 1.0.2 – Used to load configuration from files.

DB Connection Factory 1.1.0 – Used to obtain DB connection in a configurable manner.

Logging Wrapper 2.0.0 – Used to perform logging.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None.

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.topcoder.management.reviewfeedback

com.topcoder.management.reviewfeedback.impl

3.2 Configuration Parameters

3.2.1 JDBCReviewFeedbackManager

JDBCReviewFeedbackManager loads configuration from ConfigurationObject instance (either loaded from file or passed directly by caller). See JDBCReviewFeedbackManager constructors for more details about passing configuration to this class.

The table below described the configuration parameters, which will be looked for in the provided / loaded ConfigurationObject:

Parameter	Description	Value
logName	Name of the logger, used to obtain logger via LogFactory. That logger will be used to perform logging as per 1.3.1.	String. Optional. If not provided, then logging will not be performed.

dbConnectionName	Database connection name used to obtain a database connection via configured DBConnectionFactory. That connection will be used to perform operations in persistence.	String. **Required.** Not empty.
dbConnectionFactoryConfiguration	Configuration for database connection factory (DBConnectionFactoryImpl), which will be used to create database connections to access persistence.	ConfigurationObject. **Required.** Must be a valid configuration for DBConnectionFactoryImpl.
createAuditActionTypeId	Audit action type ID for "create" action, used for auditing.	long. Optional. Default value is 1.
updateAuditActionTypeId	Audit action type ID for "update" action, used for auditing.	long. Optional. Default value is 3.
deleteAuditActionTypeId	Audit action type ID for "delete" action, used for auditing.	long. Optional. Default value is 2.

3.3 Dependencies Configuration

Refer documentation of the dependency components to configure them.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Setup Informix database:

  a. Run test_files/sqls/ddl.sql to create the tables.

  test_files/sqls/drop.sql can be used to drop tables.

  b. Update connection properties in test_files/config.xml if needed.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Please see the demo.

4.3 Demo

4.3.1 Configuration

Content of configuration file “config.xml”:

```
<?xml version="1.0"?>
<CMConfig>
  <Config name="com.topcoder.management.reviewfeedback">
    <property name="logName">
      <value>com.topcoder.management.reviewfeedback</value>
    </property>
    <property name="dbConnectionName">
      <value>TCSCatalog</value>
    </property>
   </Config>

    <Config name="dbConnectionFactoryConfiguration">
      <!-- Configuration for DBConnectionFactoryImpl should be placed here. It must have "TCSCatalog" connection configured.
-->
                <Property name="com.topcoder.db.connectionfactory.DBConnectionFactoryImpl">
                    <Property name="connections">
```

```
                        <Property name="default">
                            <Value>TCSCatalog</Value>
                        </Property>
                        <Property name="TCSCatalog">
                            <Property name="producer">
                                <Value>com.topcoder.db.connectionfactory.producers.JDBCConnectionProducer</Value>
                            </Property>
                            <Property name="parameters">
                                <Property name="jdbc_driver">
                                    <Value>com.informix.jdbc.IfxDriver</Value>
                                </Property>
                                <Property name="jdbc_url">
                                    <Value>jdbc:informix-sqli://localhost:1526/test:informixserver=ol_topcoder</Value>
                                </Property>
                                <Property name="user">
                                    <Value>informix</Value>
                                </Property>
                                <Property name="password">
                                    <Value>123456</Value>
                                </Property>
                            </Property>
                        </Property>
                    </Property>
                </Property>
    </Config>

</CMConfig>
```

4.3.2 Input

```
// Create sample input.
ReviewFeedback entity = new ReviewFeedback();
entity.setProjectId(1);
entity.setComment("comment text");
ReviewFeedbackDetail detail = new ReviewFeedbackDetail();
detail.setReviewerUserId(126);
detail.setScore(2);
detail.setFeedbackText("feedback text");
List<ReviewFeedbackDetail> details = new ArrayList<ReviewFeedbackDetail>();
details.add(detail);
entity.setDetails(details);
```

4.3.3 Usage

```
// Create DAO.
ReviewFeedbackManager dao = new JDBCReviewFeedbackManager();

// Perform CRUD operations
// Create.
entity = dao.create(entity, "operator");
System.out.println("entity.getId() = " + entity.getId());
System.out.println("entity.getCreateUser() = " + entity.getCreateUser());
// Update.
entity.getDetails().get(0).setScore(1);
dao.update(entity, "anotherOperator");
// Get.
long id = entity.getId();
entity = null;
entity = dao.get(id);
System.out.println("entity.getScore() = " + entity.getDetails().get(0).getScore());
System.out.println("entity.getModifyUser() = " + entity.getModifyUser());

// Retrieves entities with given project ID from persistence.
List<ReviewFeedback> list = dao.getForProject(1);
System.out.println(list.size() + " ReviewFeedback entities return.");

// Delete.
dao.delete(entity.getId());
entity = dao.get(entity.getId());
System.out.println("'entity == null' = " + (entity == null));
```

4.3.4 *Output*

Text printed to console (generated ID is just a sample; actual value depends on database state):

```
entity.getId() = 1
entity.getCreateUser() = operator
entity.getScore() = 1
entity.getModifyUser() = anotherOperator
1 ReviewFeedback entities return.
'entity == null' = true
```

5. Future Enhancements

None.