# Review Management Component Specification

## 1. Design

Reviews are produced based on scorecards. A review holds a collection of items which address each of the questions on the scorecard. It also consists of the author that produced the review, the submission it addresses and the scorecard template it is based on. Various types of comments can be attached to the review or to each review item. A committed review must address all questions on the corresponding scorecard, and will have its overall score available.

The component provides the management functionalities to create, update or search reviews. The review persistence logic is pluggable.

### 1.1 Design Patterns

DefaultReviewManager and ReviewPersistence implement the strategy pattern.

### 1.2 Industry Standards

JDBC

Informix

### 1.3 Required Algorithms

#### *1.3.1 Transaction of InformixReviewPersistence*

All the methods of InformixReviewPersistence that involve in the database operations, should put the database queries in one transaction. Take the createReview(Review,String) method for instance, in order to create Review, this method will also be responsible for creating the associated Item and Comment. All these database operations should be put in one transaction. The connection is retrieved from DBConnectionFactory, and it should be closed after operation to release to connection pool.

Note that searchReview method will fall into two transactions. One transaction is used to retrieve an array of review ids, and the other to initialize the review array.

#### *1.3.2 Create the Review in the persistence*

This algorithm is also responsible for creating the associated Comment, and Item, but not for CommentType. Please note that the comment_type_lu table is simply a look-up table.

This method is responsible for setting the creation user and modification user for the review instance. The creation user and modification user is identical to the passed in operation.

Before creating the review instance in the database, all the associated review properties should be validated.

Please see the Review Data Structure component about which properties should be validated, and the corresponding valid values. Review class in the Review Data Structure component might add validate() method to perform the data validation. If that, the

validate() method will be called before inserting the instance into the database.

```
Impl note:

1. Get the review id and check that if it exists in the database. If
exist, throw DuplicateReviewEntityException.

2. get id from reviewIDGenerator, set it on review, and insert the review
properties into the review table

3. get all comments from review, generate id for each comment(using
reviewCommentIDGenerator), and insert the comment into the table.

4. get all the items from review, generate id for each item(using
reviewItemIDGenerator), and insert the item into the table.

5. for each item in the step 4, get all the item comments, and generate
the id for them(using reviewItemCommentIDGenerator), and insert them into
the item_comment table.

6. if any exception occurred, reset the review properties to  their
initial values, and roll back.

Note that: the source_id field represents the author id
```

### *1.3.3 Update Review in the persistence*

This algorithm is also responsible for creating, deleting, updating the associated Items, Comments, and Item Comments. The algorithm is also responsible for set operator on Review#modifictionUser.

Before updating the review instance in the database, all the associated review properties should be validated.

Please see the Review Data Structure component about which properties should be validated, and the corresponding valid values. Review class in the Review Data Structure component might add validate() method to perform the data validation. If that, the validate() method will be called before inserting the instance into the database.

```
Impl note:

1. get the review id, and check that it exists in the database.
   otherwise, throw ReviewEntityNotFoundException.

2. get the Review bean properties, and update them on "review" table.

3. select all the comment id from review_comment table which belong to
   the given review. Assume that idSet keeps the comment ids retrieved in
   this step.

4. get the review comments, for each comment, get the comment id,

if the id exists in idSet, update the comment, and remove the id from
   idSet.

if not exists, get the id(using reviewCommentIDGenerator), set the id on
   comment, and create the comment.

5. If idSet is not empty, iterate through the set, and remove the comment
   from review_comment table for each comment id

7. select all the review item ids from review_item table which belong to
   the given review.. Assume that itemIdSet keeps the items ids retrieved
   in this step.

8. get the review items, for each item id:

if the id exists in the itemIdSet, update the item, and remove the id
   from itemIdSet. Note that update the item involve creating, updating
   and deleting the item comments. t

If the id not exists, get the id from reviewItemIDGenerator, and create
   the item. Note that creating item will involve creating all the
   associated item comments.
```

```
9. if itemIdSet is not empty, iterate through the set, for each id,
   remove it from review_item table.
10. If any error occurred, roll back, and reset the review properties to
   their initial statuses..
```

### *1.3.3 Get Review from the persistence*

Get the Review instance from the persistence with given id

```
Impl note:
1. if id is not positive, throw IllegalArgumentException
2. check that id exists in the database, if not exists, throw
ReviewEntityNotFoundException
3. get all comment types from comment_type_lu table.
4. create a Review isntance, select all the Review properties  and
Comments from the review, review_comment  and set them on the instance.
Notice that the source_id represents Review#author.
The step 4 must be done in one SQL query
5. select all the review item and item comments from review_item and
review_item_comment tables where review_id is equal to given id,
and create the Item instance for each row,
Note that the Item#getDocument() is identical review_item.upload_id
The step 5 must also be done in one SQL query.
```

### *1.3.4 Search for the review in the persistence*

Search the persistence with given filter. If complete is false, the associated items and comments of the matching review will not be retrieved.

In the version 1.0, the filter supports at most five fields:

scorecardType --- the score card type, must be java.Long type

submission --- the review submission id, must be java.Long type

reviewer --- the author of the review, must be java.Long type

project --- the project id of the review, must be java.Long type

committed --- indicate if the review has been committed, must be java.lang.Integer type.

Either new Integer(1), representing committed, or new Integer(0), represent not committed

```
Impl note:
1. get an array of review ids by calling
this.searchBundle.search(filter). If the returned CustomResultSet is
empty, simply return empty array.
2. if complete is false, select all the review properties from review
table:
```

```
select * from review where review_id IN (,,,,,) // the (,,,) is an array
of review ids retrieved in step 1

Initialize an array of review with data retrieved in the above sql and
return it.

Review#getAhtor() is identical to review.resource_id

2. if complete is true get all comment types  from comment_type_lu table

2.1 retrieve review properties and review comments from review ,
review_comment tables

where review.review_id=review_comment.review_id and

review_id IN (....) order by review_id, review_comments.review_comment_id

Initialize the array of review with the data retrieved in the above sql
and the comment types retrieved in step 2

2.2 retrieve review items and item comments from review_item:

select review_id, item properties(document, question, etc), item comment
properties from review_item, reiview_item_comment where

review_item.review_id IN (.....) and

review_item.review_item_id=review_item_comment.review_item_id

order by review_id, reivew_item_id, review_item_comment_id

Initiliaze the reviews with items retrieved in the above sql and the
comment types retrieved in step 2.
```

### 1.4 Component Class Overview

**ReviewManagement**:

This interface defines the contract of managing the review entities the datastore. It provides the management functionalities to create, update and search reviews. Additionally, application users can also add comment for review and item, and get all the comment types from the manager.

The implementation of this interface is not required to be thread safe.

**DefaultReviewManagement**:

This class is the default implementation of ReviewManager. It actually delegates the work to the pluggable persistence to create, update and search reviews. Additionally, application users can also add comment for review and item, and get all the comment types from the manager.

This class is not thread safe.

**ReviewPersistence**:

This interface defines the contract of managing the review entities in the persistence. It provides the persistence functionalities to create, update and search reviews. Additionally, application users can also add comment for review and item, and get all the comment types from the persistence.

This class is not thread safe since it’s mutable.

**InformixReviewPersistence**:

This class is responsible for creating, updating, searching the review entities stored in the Informix database. Additionally, application users can also add comment for review and item, and get all the comment types from the persistence.

This class is not thread safe.

**ChainFilter**:

The ChainFilter class provides the convenient methods to constructor composite filter with "chain" operation.

For instance, if the search condition is that " search for the reviews which have been committed, reviewer id is 10000, or project id is greater than 10001",

we can construct the filter in this way

ChainFilter cf = new ChainFilter(new EqualToFilter("committed", 't'));

cf.and(new EqualToFilter("reviewer", new Long(10000))).or(new GreaterThanFilter("project", new Long(10001)));

This class does not implement the Filter interface from Search Builder since DatabaseSearchStringBuilder hardcodes all the filter types, which

make it impossible to add new Filter without modifying the source code of DatabaseSearchStringBuilder.

This class is not thread safe since it's mutable.

## 1.5 Component Exception Definitions

**IllegalArgumentException**

This exception is thrown in various methods if null object is not allowed, or the given string argument is empty. Refer to the documentation in Poseidon for more details.

**NOTE: Empty string means string of zero length or string full of whitespaces.**

**ConfigurationException**

This exception is thrown from the constructor taking a namespace argument, if failed to load configuration values from the ConfigManager or the configuration values are invalid.

**ReviewManagementException**

This exception will be thrown by ReviewManager, ReviewPersistence and their implementations if any error occurs during the management of review entities

**ReviewPersistenceException**

This exception will be thrown by ReviewPersistence and its implementation if any error occurred during persisting the review entity.

**ReviewEntitiyNotFoundException**

This exception will be thrown by InformixReviewPersistence if application tries to get or update a non-existing review entity

**DuplicatedReviewEntityException**

This exception will be thrown by InformixReviewPersistence if application tries to create a

duplicated review entity.

### 1.6 Thread Safety

Thread safety is not required, and this component is not thread safe.

The ReviewPersistence and its implementation are not thread safe. InformixReviewPersistence are not thread safe since its inner states like DBConnectionFactory makes no guarantee of thread safety. ReviewManager and its implementation also not thread safe. DefaultReviewManger is simply a wrapper of ReviewPersistence. ChainFilter is not thread safe since it's mutable.

## 2. Environment Requirements

### 2.1 Environment

Java 1.4 or higher.

### 2.2 TopCoder Software Components

**Base Exception 1.0**

The custom exception extends BaseException in this component.

**Configuration Manager 2.1.4**

It will be used by InformixReviewPersistence to load the configuration values.

**DB Connection Factory 1.0**

It will be used by InformixReviewPersistence to get named connection.

**Search Builder 1.3**

It's used by InformixReviewPersistence to generate the complicated search conditions..

**ID Generator 3.0**

It is used by InformixReviewPersistence to get the long id for review entities

### 2.3 Third Party Components

None.

## 3. Installation and Configuration

### 3.1 Package Name

com.topcoder.management.review

com.topcoder.management.review.persistence

### 3.2 Configuration Parameters

Configuration values for InformixReviewPersistence

| Parameter | Description | Values |
|---|---|---|
| **connection.name** | The connection name to retrieve connection from DB Connection Factory . **Required**. | review_manager_connection. |
| **connection.factory_namespace** | The namespace used in DB Connection Factory component, **Required**. | com.topcoder.db.connectionfactory.DBConnectionFactoryImpl |
| **connection.factory_class** | The DBConnectionFactory class name used to create DB Connection Factory , **Required**. | com.topcoder.db.connectionfactory.DBConnectionFactoryImpl |
| **search_bundle_manager_namespace** | The namespace of SearchBundleManager | com.topcoder.management.review.persistence.InformixReviewPersistence.SearchBundleManager |
| **Search_bundle_name** | The name of the SearchBundle. **Required** | databaseName |

Configureation values for DefaultReviewManager

| Parameter | Description | Values |
|---|---|---|
| **persistence.class** | The persistence class name. **Required**. | com.topcoder.management.review.persistence.InformixReviewPersistence |
| **persistence.namespace** | The namespace used to create the persistence via reflection, **Required**. | com.topcoder.management.review.persistence.InformixReviewPersistence |

Here is an example of the configuration file.

```
<?xml version="1.0" ?>
<?xml version="1.0" encoding="UTF-8"?>
<CMConfig>
   <Config name="com.topcoder.management.review">

      <Property name="connection">
         <!-- the connection name to retrieve connection in DB Connection Factory, required -->
         <Property name="name">
            <Value>asset_manager_connection</Value>
         </Property>
```

```
            <!-- The DBConnectionFactory class name used to create DB Connection Factory, required -->
            <Property name="factory_namespace">
               <Value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</Value>
            </Property>
            <Property name="factory_class">
               <Value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</Value>
            </Property>
         </Property>

         <Property name="search_bundle_manager_namespace">
            <Value>
com.topcoder.management.review.persistence.InformixReviewPersistence.SearchBundleManager </Value>
         </Property>
         <Property name="search_bundle_name">
            <Value>databaseName</Value>
         </Property>

      </Config>
      <Config name="com.topcoder.management.review.DefaultReviewManager">
         <Property name="persistence">
            <Property name="persistence_namespace">
               <Value>com.topcoder.management.review.persistence.InformixReviewPersistence</Value>
            </Property>
           <Property name="persistence_class">
               <Value> com.topcoder.management.review.persistence.InformixReviewPersistence</Value>
            </Property>
         </Property>
      </Config>

</CMConfig>
```

### 3.3 Dependencies Configuration

ConfigurationManager,DBConnectionFactory and IDGenerator should be properly configured to make this component work.

## 4. Usage Notes

### 4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

### 4.2 Required steps to use the component

Preload the configuration file into Configuration Manager. Follow demo.

### 4.3 Demo

The configuration file listed in the section 3.2 is used in the demo.

#### *4.3.1 Create instance of ReviewManagement*

```
// create the instance with given namespace,
ReviewPersistence persist = new InformixReviewPersistence(namespace);
ReviewManager manager = new DefaultReviewManager(persist);
```

#### *4.3.2 Manage Review*

```
// Get all comment types
CommentType[] commentTypes = manager.getAllCommentTypes();

// create the review in the manager
// assume that the review is initialized
Review review = …// initialized the review
manager.createReview(review, “someReviewer”);

// add comment for review
Comment reviewComment = …// assume that comment is initialized.
Manager.addReviewComment(review.getId(), comment, “admin”);

// add comment for item
Comment itemComment = ..// assume that comment is initialized
// assume that the itemId is retrieved from review
manager.addItemComment(itemId, itemComment, “admin”);

// update the review
review.setCommitted(true);
manager.updateReview(review, “someReviewer”);

// search reviews

// search for the reviews which have been committed
Filter filter = new EqualToFilter(“committed”,new Integer(1));
Review[] reviews = manager.searchReviews(filter,true);
// iterate through the array and check

// search for the reviews which have been committed and reviewer id is
   10000
ChainFilter cf = new ChainFilter(filter);
cf.and(new EqualToFilter(“reviewer”, new Long(10000));
reviews = manager.searchReviews(cf.getFilter(), true);
// check the array

// search for the reviews which have been committed, reviewer id is
   10000,
// or project id is greater than 10001
cf.or(new GreaterThanFilter(“project”, 10001);
reviews = manager.searchReviews(cf.getFilter(), true);
// check the array
```

## 5. Future Enhancements

More ReviewPersistence implementations could be added.