Review Data Structure 1.0 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. This component defines the object model, hierarchy, and data structures for a review.

All of the classes that make up this model (Review, Item, Comment, and CommentType) possess a Java Bean-like interface. That is, they all have a no-argument constructor, and all properties have both getter and setter methods. Convenience constructors for setting some fields are also provided. Although all the classes in this component share the id field and getter/setter, no abstract base class is used. This is partly because the code savings would be minimal, but mostly because inheriting from an abstract base class implies that various subclasses are just specializations of some common base. In this component, the various classes do not have a "common ancestry" to represent, so an abstract base class is not justified.

1.1 Design Patterns

The ReviewEditor class is similar to a **Proxy**. However, it does not proxy the get methods, but instead allows direct access to the review. Nor is this a **Decorator** pattern, as no common interface/base class exists.

1.2 Industry Standards

The Review/Item/Comment/CommentType classes have a Java Bean-like interface. (Although they do not broadcast property changes, which is an integral part of normal Java Beans.)

1.3 Required Algorithms

There are no complicated algorithms in this class, so this section will simply review a few features in the design which may be useful for developers to see condensed here.

1.3.1 ReviewEditor

All set methods in the ReviewEditor class need to make the following sequence of calls:

- Call the corresponding set/reset/add/remove/clear method on the review
- Call setModificationUser on the review
- Call setModificationTimestamp on the review with the current date/time

1.3.2 set and reset methods

These methods provide all of the functionality to manipulate the main classes of this component. There are essentially two types of pairs of set and reset methods provided:

- When the underlying field is a primitive long value, the set method will not permit the user to set the field to its "unassigned" value (which is -1). The reset method must be used in this case.
- When the underlying field is a reference type, the set method can be used both for a normal set and to set the field to its "unassigned value" (which is null). The reset method is provided as a convenience in this case.

The difference in the set/reset behavior is based on the reasoning that when the field is a primitive long type, it is expected to always be in a valid state once initialization of the object (by loading data from a database or through another method) is complete. Resetting it is not considered a normal action and therefore warrants a special method to enact it. On the other hand, for fields for which an "unassigned" (i.e. null) value is expected to occur in normal use, the set method is allowed to set the field to the "unassigned" value, as this is an expected normal state for these fields. The reset method is provided in these cases as a convenience and to maintain API consistency.

1.4 Component Class Overview

Review:

The Review class is the class at the root of the review modeling hierarchy. It represents an entire review, which is composed primarily of a list of items. Comments can also be associated with a review, and each review also possesses a few simple data fields. The review class is simply a container for these basic data fields (along with the comment and item lists). All data fields are mutable, and each data field has a 3 method get/set/reset combination for that data field. Each of the comments and item lists is manipulated through 9 methods (2 adds, 3 removes, a clear and 3 getters).

The only thing to take note of when developing this class is the difference between the set/reset pairs for longs and the set/reset pairs for non-primitive fields. The differences are well documented in the method doc, and the reasoning for this is explained in the component specification.

This class is highly mutable. All fields can be changed.

Item:

The Item class is the second level in the review model hierarchy. A review consists of responses to individual items, each of which is represented by an instance of this class. There are two types of data fields for this class: the simple data fields and the comments list. All data fields are mutable, and each data field has a 3 method get/set/reset combination for that data field. The

comments list is manipulated through 9 methods (2 adds, 3 removes, a clear, and 2 getters).

The only thing to take note of when developing this class is the difference between the set/reset pairs for longs and the set/reset pairs for non-primitive fields. The differences are well documented in the method doc, and the reasoning for this is explained in the component specification.

This class is highly mutable. All fields can be changed.

Comment:

The Comment class can be considered the third level in the review model hierarchy. Comments can be associated with both reviews/items. There is no limit on the number of comments that can be associated with a review/item. Since a review/item is not required to have any associated comments, the Comment class can be seen as (slightly) auxiliary to the main Review/Item part of the model. As in the Review and Item classes, each data field has a 3 method get/set/reset combination for manipulating that data field.

The only thing to take note of when developing this class is the difference between the set/reset pairs for longs and the set/reset pairs for non-primitive fields. The differences are well documented in the method doc, and the reasoning for this is explained in the component specification.

This class is highly mutable. All fields can be changed.

CommentType:

The CommentType class supports the Comment class, and allows a comment to be tagged as being of a certain type/style. Unlike the other classes in this component, which are likely to be created dynamically and frequently, only a few CommentType instances are likely to be used in any application. For this component, this consideration really has no impact on the design or development of this class. Like the other classes in this component, it consists of simple data fields, each of which has a 3 method get/set/reset combination for manipulating that field.

The only thing to take note of when developing this class is the difference between the set/reset pairs for longs and the set/reset pairs for non-primitive fields. The differences are well documented in the method doc, and the reasoning for this is explained in the component specification.

This class is highly mutable. All fields can be changed.

ReviewEditor:

The ReviewEditor class provides the ability to edit the properties of a review while automatically updating the modification user and modification date

whenever a property is edited. This class allows the client using this component to associate a user with an editing session and then avoid the hassles of having to manually call the setModificationUser and setModificationTimestamp whenever changes are made. This class simply parallels the set/reset/add/remove/clear methods of the Review class. Each set/reset/add/remove/clear call simply calls the same method on the Review instance and then invokes the setModificationUser and setModificationTimestamp methods.

This class is immutable.

1.5 Component Exception Definitions

No custom exceptions are defined in this component. The Java standard exception IllegalArgumentException and IndexOutOfBoundsException are sufficient to cover all exceptional situations that occur.

1.6 Thread Safety

This component is not thread safe. This decision was made because the majority of the classes in this component are mutable. Combined with there being no business oriented requirement to make the component thread safe, making this component thread safe would only increase development work and needed testing while decreasing runtime performance (because synchronization would be needed for get/set methods). These tradeoffs can not be justified given that there is no current business need for this component to be thread-safe.

This does not mean that making this component thread-safe would be particularly hard. Making the Review/Item/Comment/CommentType classes thread safe can be done by simply adding the synchronized keyword to the various set and access methods. This would leave only the ReviewEditor class as possibly non thread-safe. The only thread safety issue that would remain with this class is that multiple ReviewEditor instances that are editing the same underlying Review from multiple threads could conflict in the following way:

- ReviewEditor #1 sets attribute
- ReviewEditor #2 sets attribute
- ReviewEditor #2 sets modification date/time
- ReviewEditor #1 sets modification date/time

Notice that this causes the modification date/time (last set by #1) to not reflect the last modification (made by #2). Note that even in this case, the synchronization in the Review class would prevent any internal data structure corruption. This race condition could be eliminated by having ReviewEditor lock on the review field in each set/reset/add/remove/clear method.

2. Environment Requirements

2.1 Environment

Java 1.4 or greater is required for compilation, testing, or use of this component.

2.2 TopCoder Software Components

None.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.management.review.data

3.2 Configuration Parameters

None.

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

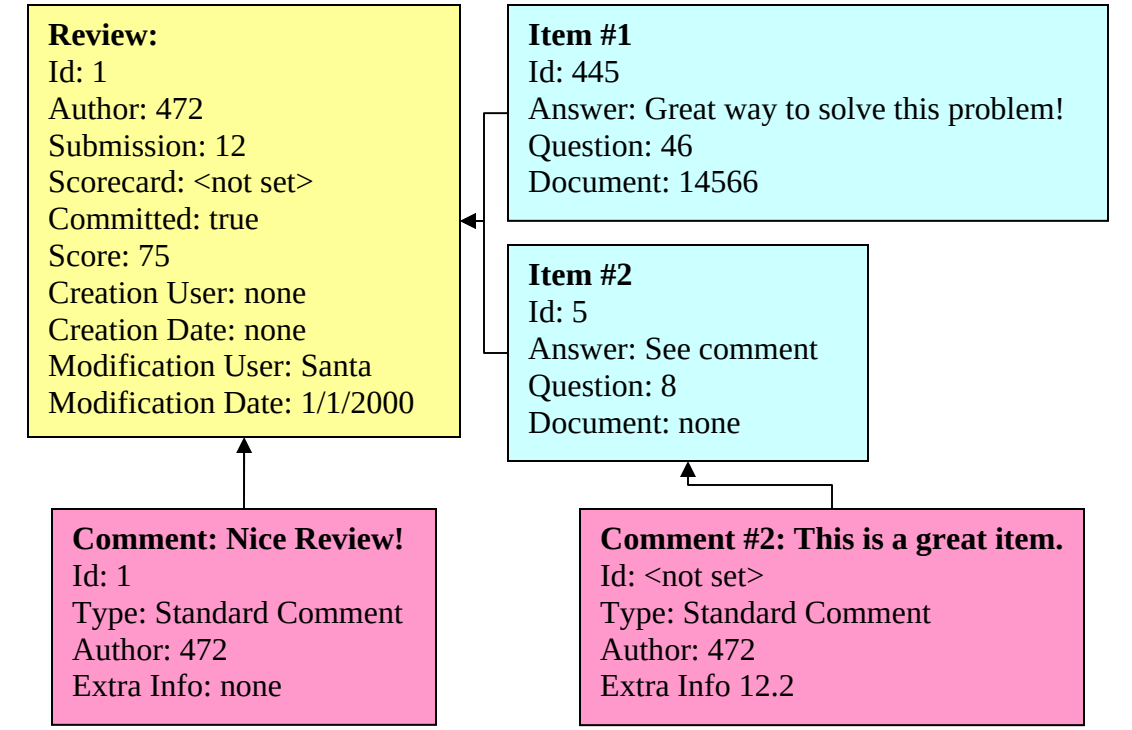
- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Install as per section 3 and follow demo below.

4.3 Demo

As there is only really one thing that can be done to with this component - build up a review hierarchy (and query into an existing review hierarchy) - this demo will simply show how to set up a review object hierarchy that a customer might use. The hierarchy will be the one in the following diagram. (The CommentType, class has been condensed into a single line in the classes that use it. This is done simply to make the diagram fit.)



4.3.1 Creating a Review hierarchy

```
// Create the review
Review review = new Review();

// Set review data values
review.setId(1);
review.setAuthor(45);
review.setSubmission(12);
review.setCommitted(true);
review.setScore(new Float(75.0));
review.setModificationUser("Santa");
review.setModificationTimestamp(
   DateFormat.getDateInstance().parse("2000-1-1"));

// Set the scorecard and then demonstrate how to reset it
// to the unassigned value
review.setScorecard(1);
review.resetScorecard();

// Create comment type (in any real system, this would definitely
// be loaded from some persistence of standard comment types)
CommentType commentType = new CommentType(1, "Standard Comment");

// Create comment for review
Comment reviewComment = new Comment(1);

// Set comment data values
reviewComment.setCommentType(commentType);
reviewComment.setAuthor(472);
```

```
// Note how for non-primitive values, null is allowed in the
// setter as a way to set the value as “unassigned”
reviewComment.setExtraInfo(null);

// Add comment to review
review.addComment(reviewComment);

// Create items
Item item1 = new Item(445);
item1.setAnswer(“Great way to solve this problem!”);
item1.setQuestion(46);
item1.setDocument(new Long(14566));

Item item2 = new Item();
item2.setId(5);
item2.setAnswer(“See comment”);
item2.setQuestion(8);
// Note no need to set document, as no attached document is the
// default state.

// Create item comment
Comment itemComment = new Comment();
itemComment.setCommentType(commentType);
itemComment.setAuthor(472);
itemComment.setExtraInfo(new Double(12.2));

// Add comment to item
item2.addComment(itemComment);

// Add items to review
review.addItems(new Item[] {item1, item2});
```

4.3.2 *Retrieving data from a Review hierarchy*

Typically, to inspect the data in a review, the review would be loaded from some persistence source, probably by using the Review Management component. This demo will demonstrate the retrieval functionality using the review created above.

```
// In reality, this sort of a line would be used to get a review
// for inspection.
// review = reviewManager.getReview(12345);

// Retrieve data from the review
long id = review.getId();           // will be 1
long author = review.getAuthor();   // will be 45
// … Continue in like manner for other data fields

// Retrieve comments
Comment[] comments = review.getAllComments();
// And iterate over comments
for (int i = 0; i < comments.length; i++) {
      Comment comment = comments[i];
      long commentId = comment.getId();
      // … Continue in like manner for other data fields
      CommentType commentType = comment.getCommentType();
      String commentTypeName = commentType.getName();
```

```
}

// Retrieve items
Item[] items = review.getAllItems();

// Iterate over the items
for (int i = 0; i < items.length; i++) {
      Item item = items[i];
      long itemId = review.getId();
      // … Continue in like manner for other data fields
      Comment[] itemComments = item.getAllComments();
      // Iterate over comments as above
}
```

4.3.3 Using ReviewEditor

The ReviewEditor can be used by a client to edit the properties of a review. Either one created as in the last section, or one loaded from persistence (see the Review Management component).

```
// Create a ReviewEditor for editing the properties of the review
// while automatically updating modification
// user and date/time
ReviewEditor editor = new ReviewEditor(review, “The Grinch”);

// Update a property of the review, automatically updating the
// modification user and date/time
editor.setScorecard(7);
editor.resetScorecard();

// Retrieve the modification date and user.  These will now
// be the “The Grinch” and the current date/time
String modificationUser =
      editor.getReview().getModificationUser();
Date modificationDate =
      editor.getReview().getModificationTimestamp();
```

5. Future Enhancements

As this is a custom component with a well defined (and quite limited) scope, no enhancements are expected.