Deliverable Management Persistence 1.2 Component Specification

1. Design

All changes performed when synchronizing documentation with the version 1.1 of the source code of this component are marked with **purple**.

All changes made in the version 1.2 are marked with **blue**.

All new items in the version 1.2 are marked with **red**.

This component provides implementations of UploadPersistence and DeliverablePersistence interfaces defined in Deliverable Management component. SqlUploadPersistence and SqlDeliverablePersistence use JDBC to manage deliverable objects in the database.

In the version 1.1 a support of submission types was added to this component.

In the version 1.2 this component was updated to support changes in the Deliverable Management 1.2 component. Added support for description property of Upload entity and some Studio-specific properties of Submission entity. Added methods for managing SubmissionImage entities, retrieving MimeType entities, retrieving project/user submissions and retrieving images associated with submission to SqlUploadPersistence class. The DDL was updated respectively (please see updated_db_schema.sql file provided together with this specification).

***FOR DEVELOPERS*: Please update the source code of this component to make it use generic collections properly (i.e. List<Submission> instead of List, etc). This is required due to compile target update.**

1.1 Design Patterns

DAO pattern – SqlUploadPersistence and SqlDeliverablePersistence serve as DAOs for Submission, Upload, SubmissionStatus, UploadType, UploadStatus, SubmissionType, Deliverable, SubmissionImage, MimeType DTOs.
Strategy pattern – SqlUploadPersistence and SqlDeliverablePersistence are used as strategy implementations in Deliverable Management component.

1.2 Industry Standards

JDBC, SQL

1.3 Required Algorithms

1.3.1 SQL Queries for SqlUploadPersistence

This section lists the SQL queries and statements that will be needed by the SqlUploadPersistence class, accompanied by a written explanation of what to do, when more logic than just a sequence of SQL statements is needed. All of these SQL statements should be executed using PreparedStatements. A typical method of this class would look like (this example uses the "Update Upload Type" statement):

SQL statement to execute:

```
UPDATE upload_type_lu
SET name = ?, description = ?, modify_user = ?, modify_date = ?
WHERE upload_type_id = ?
```

Sample Code:

```
// Open connection
Connection connection = connectionFactory.createConnection();
// Create a prepared statement
PreparedStatement ps =
      connection.prepareStatement(< above text >);
// Use data in NotificationType passed to method to set
// parameters in prepared statement
ps.setString(1, uploadType.getName());
ps.setString(2, uploadType.getDescription());
ps.setString(3, uploadType.getModificationUser());
ps.setDate(4, uploadType.getModificationDate());
ps.setLong(5, uploadType.getId());
// execute PreparedStatement
ps.execute();
// close connection
connection.close();
```

1.3.1.1 Add Upload Type

```
INSERT INTO upload_type_lu
(upload_type_id, name, description, create_user, create_date,
modify_user, modify_date)
VALUES (?, ?, ?, ?, ?, ?, ?)
```

1.3.1.2 Remove Upload Type

```
DELETE FROM upload_type_lu
WHERE upload_type_id = ?
```

1.3.1.3 Update Upload Type

```
UPDATE upload_type_lu
SET name = ?, description = ?, modify_user = ?, modify_date = ?
WHERE upload_type_id = ?
```

1.3.1.4 Load Upload Type

```
SELECT upload_type_id, name, description, create_user,
create_date, modify_user, modify_date
FROM upload_type_lu
WHERE upload_type_id IN (<id_value(s)>)
```

1.3.1.4.1 Load Upload Types

The query specified in 1.3.1.4 is used.

1.3.1.5 Add Upload Status

```
INSERT INTO upload_status_lu
(upload_status_id, name, description, create_user, create_date,
modify_user, modify_date)
VALUES (?, ?, ?, ?, ?, ?, ?)
```

1.3.1.6 Remove Upload Status

```
DELETE FROM upload_status_lu
WHERE upload_status_id = ?
```

1.3.1.7 Update Upload Status

```
UPDATE upload_status_lu
SET name = ?, description = ?, modify_user = ?, modify_date = ?
WHERE upload_status_id = ?
```

1.3.1.8 Load Upload Status

```
SELECT upload_status_id, name, description, create_user,
create_date, modify_user, modify_date
FROM upload_status_lu
WHERE upload_status_id IN (<id_value(s)>)
```

1.3.1.8.1 Load Upload Statuses

The query specified in 1.3.1.8 is used.

1.3.1.9 Add Submission Status

```
INSERT INTO submission_status_lu
(submission_status_id, name, description, create_user,
create_date, modify_user, modify_date)
VALUES (?, ?, ?, ?, ?, ?, ?)
```

1.3.1.10 Remove Submission Status

```
DELETE FROM submission_status_lu
WHERE submission_status_id = ?
```

1.3.1.11 Update Submission Status

```
UPDATE submission_status_lu
SET name = ?, description = ?, modify_user = ?, modify_date = ?
WHERE submission_status_id = ?
```

1.3.1.12 Load Submission Status

```
SELECT submission_status_id, name, description, create_user,
create_date, modify_user, modify_date
FROM submission_status_lu
WHERE submission_status_id IN (<id_value(s)>)
```

1.3.1.12.1 Load Submission Statuses

The query specified in 1.3.1.12 is used.

1.3.1.13 Add Upload

```
INSERT INTO upload
(upload_id, project_id, resource_id, upload_type_id,
upload_status_id, parameter, create_user, create_date,
modify_user, modify_date, upload_desc)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

1.3.1.14 Remove Upload

```
DELETE FROM upload
WHERE upload_id = ?
```

1.3.1.15 Update Upload

```
UPDATE upload
SET project_id = ?, resource_id = ?, upload_type_id = ?,
upload_status_id = ?, parameter = ?, modify_user = ?, modify_date
= ?, upload_desc = ?
WHERE upload_id = ?
```

1.3.1.16 Load Uploads

```
SELECT upload_id, project_id, resource_id, upload_type_id,
upload_status_id, parameter, upload.create_user,
upload.create_date, upload.modify_user, upload.modify_date,
upload_desc, upload_type_lu.name, upload_type_lu.upload_type_id,
upload_status_lu.name, upload_status.upload_status_id,
upload_type_lu.description, upload_status_lu.description,
upload_type_lu.create_user, upload_type_lu.create_date,
upload_type_lu.modify_user, upload_type_lu.modify_date,
upload_status_lu.create_user, upload_status_lu.create_date,
upload_status_lu.modify_user, upload_status_lu.modify_date
FROM upload
INNER JOIN upload_type_lu
      ON upload_type.upload_type_id = upload.upload_type_id
INNER JOIN upload_status_lu
      ON upload_status.upload_status_id = upload.upload_status_id
WHERE upload_id IN (<id_value(s)>)
```

1.3.1.17 Add Submission

```
INSERT INTO submission
(submission_id, ~~upload_id,~~ submission_status_id,
submission_type_id, create_user, create_date, modify_user,
modify_date , screening_score, initial_score, final_score,
placement, feedback_thumb, user_rank, mark_for_purchase,
prize_id)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

1.3.1.17.1 Associate Uploads with Submission

```
INSERT INTO upload_submission (upload_id, submission_id)
VALUES (?, ?)
(Must be performed for each upload)
```

1.3.1.18 Remove Submission

```
DELETE FROM submission
WHERE submission_id = ?
```

1.3.1.19 Update Submission

```
UPDATE submission
SET ~~upload_id = ?,~~ submission_status_id = ?, submission_type_id =
?, modify_user = ?, modify_date = ? , screening_score = ?,
initial_score = ?, final_score = ?, placement = ?, feedback_thumb
= ?, user_rank = ?, mark_for_purchase = ?, prize_id = ?
WHERE submission_id = ?
```

1.3.1.19.1 Update associations between Submission and Uploads

```
DELETE FROM upload_submission
```

```
WHERE submission_id = ?

INSERT INTO upload_submission (upload_id, submission_id)
VALUES (?, ?)
(Must be performed for each upload)
```

1.3.1.20 Load Submission

```
SELECT submission.submission_id, submission.create_user,
submission.create_date, submission.modify_user,
submission.modify_date,
submission_status_lu.submission_status_id,
submission_status_lu.create_user,
submission_status_lu.create_date,
submission_status_lu.modify_user,
submission_status_lu.modify_date, submission_status_lu.name,
submission_status_lu.description,
submission_type_lu.submission_type_id,
submission_type_lu.create_user, submission_type_lu.create_date,
submission_type_lu.modify_user, submission_type_lu.modify_date,
submission_type_lu.name, submission_type_lu.description,
upload.upload_id, upload.create_user, upload.create_date,
upload.modify_user, upload.modify_date, upload.project_id,
upload.resource_id, upload.parameter,
upload_type_lu.upload_type_id, upload_type_lu.create_user,
upload_type_lu.create_date, upload_type_lu.modify_user,
upload_type_lu.modify_date, upload_type_lu.name,
upload_type_lu.description, upload_status_lu.upload_status_id,
upload_status_lu.create_user, upload_status_lu.create_date,
upload_status_lu.modify_user, upload_status_lu.modify_date,
upload_status_lu.name, upload_status_lu.description,
submission.screening_score, submission.initial_score,
submission.final_score, submission.placement,
submission.feedback_thumb, submission.user_rank,
submission.mark_for_purchase,
prize.project_id,
prize.prize_id,
prize.place,
prize.prize_amount,
prize.prize_type_id,
prize.create_date,
prize_type_lu.prize_type_desc
FROM submission
INNER JOIN submission_status_lu ON
      submission.submission_status_id =
      submission_status_lu.submission_status_id
INNER JOIN submission_type_lu ON submission.submission_type_id =
      submission_type_lu.submission_type_id
INNER JOIN prize ON submission.prize_id = prize.prize_id
INNER JOIN prize_type_lu ON prize.prize_type_id =
      prize_type_lu.prize_type_id
INNER JOIN upload ON submission.upload_id=upload.upload_id
INNER JOIN upload_type_lu ON upload.upload_type_id =
      upload_type_lu.upload_type_id
INNER JOIN upload_status_lu ON upload.upload_status_id =
      upload_status_lu.upload_status_id
WHERE submission_id IN (<id_value(s)>)
```

1.3.1.21 Load all Upload Type ids

```
SELECT upload_type_id
FROM upload_type_lu
WHERE TRUE
```

1.3.1.22 Load all Upload Status ids

```
SELECT upload_status_id
FROM upload_status_lu
WHERE TRUE
```

1.3.1.23 Load all Submission Status ids

```
SELECT submission_status_id
FROM submission_status_lu
WHERE TRUE
```

1.3.1.24 Add Submission Type

```
INSERT INTO submission_type_lu
(submission_type_id, name, description, create_user, create_date,
modify_user, modify_date)
VALUES (?, ?, ?, ?, ?, ?, ?)
```

1.3.1.25 Remove Submission Type

```
DELETE FROM submission_type_lu
WHERE submission_type_id = ?
```

1.3.1.26 Update Submission Type

```
UPDATE submission_type_lu
SET name = ?, description = ?, modify_user = ?, modify_date = ?
WHERE submission_type_id = ?
```

1.3.1.27 Load Submission Type

```
SELECT submission_type_id, name, description, create_user,
create_date, modify_user, modify_date
FROM submission_type_lu
WHERE submission_type_id IN (<id_value(s)>)
```

1.3.1.27.1 Load Submission Types

The query specified in 1.3.1.27 is used.

1.3.1.28 Load all Submission Type ids

```
SELECT submission_type_id
FROM submission_type_lu
WHERE TRUE
```

1.3.1.29 Add Submission Image

```
INSERT INTO submission_image
(submission_id, image_id, sort_order, modify_date, create_date)
VALUES (?, ?, ?, ?, ?)
```

1.3.1.30 Update Submission Image

```
UPDATE submission_image
SET sort_order = ?, modify_date = ?, create_date = ?
WHERE submission_id = ? AND image_id = ?
```

1.3.1.31 Remove Submission Image

```
DELETE FROM submission_image
WHERE submission_id = ? AND image_id = ?
```

1.3.1.32 Load all MIME Type IDs

```
SELECT mime_type_id
FROM mime_type_lu
```

1.3.1.33 Load MIME Types

```
SELECT mime_type_id, file_type_lu.file_type_id,
file_type_lu.file_type_desc, file_type_lu.sort,
file_type_lu.image_file_ind, file_type_lu.extension,
file_type_lu.bundled_file_ind, mime_type_desc
FROM mime_type_lu
INNER JOIN file_type_lu ON
      mime_type_lu.file_type_id = file_type_lu.file_type_id
WHERE mime_type_id IN (<id_value(s)>)
```

1.3.1.34 Get User Submissions for Project

```
SELECT submission.submission_id, submission.create_user,
submission.create_date, submission.modify_user,
submission.modify_date,
submission_status_lu.submission_status_id,
submission_status_lu.create_user,
submission_status_lu.create_date,
submission_status_lu.modify_user,
submission_status_lu.modify_date, submission_status_lu.name,
submission_status_lu.description,
submission_type_lu.submission_type_id,
submission_type_lu.create_user, submission_type_lu.create_date,
submission_type_lu.modify_user, submission_type_lu.modify_date,
submission_type_lu.name, submission_type_lu.description,
submission.screening_score, submission.initial_score,
submission.final_score, submission.placement,
submission.feedback_thumb, submission.user_rank,
submission.mark_for_purchase,
prize.project_id,
prize.prize_id,
prize.place,
prize.prize_amount,
prize.prize_type_id,
prize.create_date,
prize_type_lu.prize_type_desc
FROM submission
INNER JOIN submission_status_lu ON
      submission.submission_status_id =
      submission_status_lu.submission_status_id
```

```
INNER JOIN submission_type_lu ON submission.submission_type_id =
      submission_type_lu.submission_type_id
INNER JOIN prize ON submission.prize_id = prize.prize_id
INNER JOIN prize_type_lu ON prize.prize_type_id =
      prize_type_lu.prize_type_id
WHERE EXISTS (
      SELECT upload_id
      FROM upload
      INNER JOIN upload_submission ON
            upload.upload_id = upload_submission.upload_id
      WHERE upload_submission.submission_id =
                submission.submission_id AND
            upload.project_id = ? AND
            upload.resource_id = ?
)
```

Note that resource_id field corresponds to ownerId method parameter.

1.3.1.35 Get Project Submissions

```
SELECT submission.submission_id, submission.create_user,
submission.create_date, submission.modify_user,
submission.modify_date,
submission_status_lu.submission_status_id,
submission_status_lu.create_user,
submission_status_lu.create_date,
submission_status_lu.modify_user,
submission_status_lu.modify_date, submission_status_lu.name,
submission_status_lu.description,
submission_type_lu.submission_type_id,
submission_type_lu.create_user, submission_type_lu.create_date,
submission_type_lu.modify_user, submission_type_lu.modify_date,
submission_type_lu.name, submission_type_lu.description,
submission.screening_score, submission.initial_score,
submission.final_score, submission.placement,
submission.feedback_thumb, submission.user_rank,
submission.mark_for_purchase,
prize.project_id,
prize.prize_id,
prize.place,
prize.prize_amount,
prize.prize_type_id,
prize.create_date,
prize_type_lu.prize_type_desc
FROM submission
INNER JOIN submission_status_lu ON
      submission.submission_status_id =
      submission_status_lu.submission_status_id
INNER JOIN submission_type_lu ON submission.submission_type_id =
      submission_type_lu.submission_type_id
INNER JOIN prize ON submission.prize_id = prize.prize_id
INNER JOIN prize_type_lu ON prize.prize_type_id =
      prize_type_lu.prize_type_id
WHERE EXISTS (
      SELECT upload_id
      FROM upload
      INNER JOIN upload_submission ON
            upload.upload_id = upload_submission.upload_id
```

```
        WHERE upload_submission.submission_id =
                  submission.submission_id AND
              upload.project_id = ?
)
```

1.3.1.36 Get Images for Submission

```
SELECT image_id, sort_order, modify_date, create_date
FROM submission_image
WHERE submission_id = ?
```

1.3.1.37 Load Uploads associated with Submission

```
SELECT upload.upload_id, upload.create_user, upload.create_date,
upload.modify_user, upload.modify_date, upload.project_id,
upload.resource_id, upload.parameter, upload.upload_desc,
upload_type_lu.upload_type_id, upload_type_lu.create_user,
upload_type_lu.create_date, upload_type_lu.modify_user,
upload_type_lu.modify_date, upload_type_lu.name,
upload_type_lu.description, upload_status_lu.upload_status_id,
upload_status_lu.create_user, upload_status_lu.create_date,
upload_status_lu.modify_user, upload_status_lu.modify_date,
upload_status_lu.name, upload_status_lu.description
FROM upload
INNER JOIN upload_submission ON
      upload.upload_id = upload_submission.upload_id
INNER JOIN submission ON
      upload_submission.submission_id = submission. submission_id
WHERE submission_id = ?
```

1.3.2 SQL Queries for SqlDeliverablePersistence

This section lists the SQL queries and statements that will be needed by the SqlDeliverablePersistence class, accompanied by a written explanation of what to do, when more logic than just a sequence of SQL statements is needed. For more development oriented details, see previous section.

1.3.2.1 Load deliverable with submission

```
SELECT upload.project_id, project_phase.project_phase_id,
resource.resource_id, submission.submission_id,
deliverable_lu.required, deliverable_lu.deliverable_id,
deliverable_lu.create_user, deliverable_lu.create_date,
deliverable_lu.modify_user, deliverable_lu.modify_date,
deliverable_lu.name, deliverable_lu.description
FROM deliverable_lu
INNER JOIN submission ON submission.upload_id = upload.upload_id
INNER JOIN submission_status_lu
      ON submission.submission_status_id =
            submission_status_lu.submission_status_id
INNER JOIN submission_type_lu
      ON submission.submission_type_id =
            submission_type_lu.submission_type_id
WHERE submission_status_lu.name = 'Active' AND
deliverable_lu.per_submission = 1 AND
deliverable_lu.deliverable_id = ? AND
```

```
submission.submission_id = ? AND
resource.resource_id = ? AND
project_phase.project_phase_id = ?
```

For selecting multiple deliverables (the loadDeliverables(long[], long[], long[], long[]) overload), the WHERE clause should be:

```
WHERE submission_status_lu.name = 'Active'
      AND deliverable_lu.per_submission = 1
      AND (
      (deliverable_lu.deliverable_id = ? AND submission.submission_id = ? AND
      resource.resource_id = ? AND project_phase.project_phase_id = ?)
      OR (deliverable_lu.deliverable_id = ? AND submission.submission_id = ? AND
      resource.resource_id = ? AND project_phase.project_phase_id = ?)
      OR (deliverable_lu.deliverable_id = ? AND submission.submission_id = ? AND
      resource.resource_id = ? AND project_phase.project_phase_id = ?)
      OR …)
```

1.3.2.2 Load deliverable – generic

First, deliverables without submissions are retrieved:

```
SELECT resource.project_id, project_phase.project_phase_id,
resource.resource_id, deliverable_lu.required,
deliverable_lu.deliverable_id, deliverable_lu.create_user,
deliverable_lu.create_date, deliverable_lu.modify_user,
deliverable_lu.modify_date, deliverable_lu.name,
deliverable_lu.description
FROM deliverable_lu
INNER JOIN resource ON resource.resource_role_id =
   deliverable_lu.resource_role_id
INNER JOIN project_phase ON project_phase.project_id =
   resource.project_id AND project_phase.phase_type_id =
   deliverable_lu.phase_type_id
WHERE deliverable_lu.per_submission = 0
      AND deliverable_id = ?
      AND resource.resource_id = ?
      AND project_phase.project_phase_id = ?
```

For selecting multiple deliverables (the loadDeliverables(long[], long[], long[]) overload), the WHERE clause should be:

```
WHERE deliverable_lu.per_submission = 0 AND
      (deliverable.deliverable_id = ? AND
       AND resource.resource_id = ?
       AND project_phase.project_phase_id = ?) OR
      (deliverable.deliverable_id = ? AND
       AND resource.resource_id = ?
       AND project_phase.project_phase_id = ?) OR ...
```

Next deliverables with submissions are retrieved (see selects in the section 1.3.2.1, but use WHERE clauses specified in this section above for matching deliverable_id, resource_id and project_phase_id).

All retrieved deliverables are combined into a single array.

1.3.2.3 Logging

Logging in all new methods must be performed consistently with the existing methods. LogMessage class should be used for this. Please see the source code of all submission status specific methods for reference.

1.4 Component Class Overview

SqlUploadPersistence:

The SqlUploadPersistence class implements the UploadPersistence interface, in order to persist to the database structure given in the deliverable_management.sql script. This class does not cache a Connection to the database. Instead a new Connection is used on every method call. Most methods in this class will just create and execute a single PreparedStatement. However, some of the methods will need to execute several PreparedStatements in order to accomplish their tasks.

This class is immutable and thread-safe in the sense that multiple threads cannot corrupt its internal data structures. However, the results if used from multiple threads can be unpredictable as the database is changed from different threads. This can equally well occur when the component is used on multiple machines or multiple instances are used, so this is not a thread-safety concern.

Changes in 1.1:

- Added methods for managing SubmissionType entities.
- Updated submission add/update/load methods to support submission types.

Changes in 1.2:

- Updated existing methods to support new Upload and Submission properties.
- Added methods for managing SubmissionImage entities.
- Added methods for retrieving MimeType entities.
- Added methods for retrieving project/user submissions.
- Added method for retrieving images associated with submission.

SqlDeliverablePersistence:

The SqlDeliverablePersistence class implements the DeliverablePersistence interface, in order to persist to the database structure given in the deliverable_management.sql script. This class does not cache a Connection to the database. Instead a new Connection is used on every method call. PreparedStatements should be used to execute the SQL statements.

This class is immutable and thread-safe in the sense that multiple threads cannot corrupt its internal data structures. However, the results if used from multiple threads can be unpredictable as the database is changed from different threads. This can equally well occur when the component is used on multiple machines or multiple instances are used, so this is not a thread-safety concern.

Changes in 1.1:

- Class was updated to ensure that submission types are specified for submissions when loading deliverables.

LogMessage:

This class encapsulates the entry log data and generates consistent log messages.

This class is mutable and not thread safe.

1.5 Component Exception Definitions

This component doesn't define any exceptions.

1.6 Thread Safety

This component is not thread safe, though both SqlUploadPersistence and SqlDeliverablePersistence are immutable classes. Thread safety cannot be guaranteed since updating the database from multiple threads can be unpredictable.

SqlUploadPersistence uses auto-commit mode when updating data in persistence, i.e. transactions are not used.

Thread safety of this component was not changed in the version 1.2.

2. Environment Requirements

2.1 Environment

Java 1.5+ is required for compilation, testing, or use of this component

2.2 TopCoder Software Components

- DB Connection Factory 1.0: Used in SQL persistence implementation to connect to the database.
- Database Abstraction 1.1: Defines CustomResultSet entity used by SqlUploadPersistence.
- Logging Wrapper 1.2: Used for logging errors and debug information.
- Deliverable Management 1.2: Defines UploadPersistence and DeliverablePersistence interfaces implemented in this component and deliverable entities

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.management.deliverable.persistence.sql
com.topcoder.management.deliverable.persistence.sql.logging

3.2 Configuration Parameters

No direct configuration is used for this component.

3.3 Dependencies Configuration

Please see docs of Logging Wrapper to and DB Configuration Factory to configure them properly.

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 and configure the other TopCoder component following their instructions. Then follow section 4.1 and the demo.

4.3 Demo

4.3.1 Create Upload Persistence

```
UploadPersistence uploadPersistence = new SqlUploadPersistence(
            new  DBConnectionFactoryImpl(DB_CONNECTION_NAMESPACE));
```

4.3.2 Create an Upload and Submission (with supporting classes)

```
UploadType uploadType = uploadPersistence.loadUploadType(1);
SubmissionStatus submissionStatus =
    uploadPersistence.loadSubmissionStatus(1);
SubmissionType submissionType =
    uploadPersistence.loadSubmissionType(1);
UploadStatus uploadStatus =
    uploadPersistence.loadUploadStatus(1);

// Create upload
Upload upload = new Upload(1234);
upload.setProject(24);
upload.setUploadType(uploadType);
upload.setUploadStatus(uploadStatus);
upload.setOwner(553);
upload.setParameter("The upload is somewhere");
upload.setDescription("This is a sample upload");

// Create Submission
Submission submission = new Submission(823);
List<Upload> uploads = new ArrayList<Upload>();
uploads.add(upload);
submission.setUploads(uploads);
submission.setSubmissionStatus(submissionStatus);
submission.setSubmissionType(submissionType);
submission.setThumb(true);
```

```
submission.setUserRank(2);
submission.setExtra(true);
```

4.3.3 Create deliverable persistence

```
DeliverablePersistence deliverablePersistence =
      new SqlDeliverablePersistence(
            <connection factory loaded from configuration>);
```

4.3.4 Save the created Upload and Submission

```
uploadPersistence.addUpload(upload);
uploadPersistence.addSubmission(submission);
// New instances of the tagging classes can be created through
// similar methods.

// Change a property of the Upload
upload.setProject(14424);

// And update it in the persistence
uploadPersistence.updateUpload(upload);

// Remove it from the persistence
uploadPersistence.removeUpload(upload);

// Submissions can be changed and removed similarly
```

4.3.5 Retrieve all submission types

```
long[] submissionTypeIds =
    uploadPersistence.getAllSubmissionTypeIds();

// For other lookup values IDs can be retrieved similarly
```

4.3.6 Load deliverables

```
long deliverableId = 101;
long resourceId = 10;
long phaseId = 3;
long submissionId = 10001;

// Load deliverables by deliverable, resource and phase IDs
Deliverable[] deliverables =
     deliverablePersistence.loadDeliverables(deliverableId,
         resourceId, phaseId);

// Load deliverable by deliverable, resource,
// phase and submission IDs
Deliverable deliverable =
     deliverablePersistence.loadDeliverable(deliverableId,
         resourceId, phaseId, submissionId);

// Loading deliverables for multiple IDs (of each type) is
performed similarly
```

4.3.7 Sample input/output for the version 1.1

Assume that submission_type_lu table initially contains the following data:

submission_type_id	name	description	create_user	create_date	modify_user	modify_date

1	Submission	The contest submission	admin	2010/06/10 12:12:05	admin	2010/06/10 12:12:05

And the following code is executed:

```
UploadPersistence uploadPersistence =
    new SqlUploadPersistence(new DBConnectionFactoryImpl(DB_CONNECTION_NAMESPACE));

// Load submission type with ID=1
SubmissionType submissionType = uploadPersistence.loadSubmissionType(1);
// submissionType.getId() must be 1
// submissionType.getName() must be "Submission"

// Update name of submission type together with audit data
submissionType.setName("Contest Submission");
submissionType.setModificationTimestamp(new Date());
submissionType.setModificationUser("user1");

// Update submission type in persistence
uploadPersistence.updateSubmissionType(submissionType);

// Create SubmissionType instance for "Specification Submission" type
submissionType = new SubmissionType();
submissionType.setId(2);
submissionType.setName("Specification Submission");
submissionType.setDescription("The specification submission");
submissionType.setCreationTimestamp(new Date());
submissionType.setModificationTimestamp(new Date());
submissionType.setCreationUser("user1");
submissionType.setModificationUser("user1");

// Add this new submission type to persistence
uploadPersistence.addSubmissionType(submissionType);

// Retrieve all existing submission type IDs
long[] submissionTypeIds = uploadPersistence.getAllSubmissionTypeIds();
// submissionTypeIds.length must be 2
// submissionTypeIds[0] must be 1
// submissionTypeIds[1] must be 2
// Order of elements in the array can be different
```

After executing the given code submission_type_lu table can contain the following records:

submission_type_id	name	description	create_user	create_date	modify_user	modify_date
1	Contest Submission	The contest submission	admin	2010/06/10 12:12:05	user1	2010/06/19 18:19:21
2	Specification Submission	The specification submission	user1	2010/06/19 18:19:21	user1	2010/06/19 18:19:21

4.3.8 *Usage of methods defined in the version 1.2*

```
// Create submission image
SubmissionImage submissionImage = new SubmissionImage();
submissionImage.setSubmissionId(submission.getId());
submissionImage.setImageId(1);
submissionImage.setSortOrder(1);
uploadPersistence.addSubmissionImage(submissionImage);

// Update the submission image
submissionImage.setSortOrder(0);
uploadPersistence.updateSubmissionImage(submissionImage);

// Remove the submission image
uploadPersistence.removeSubmissionImage(submissionImage);

// Retrieve the MIME type with ID=1
```

```
MimeType mimeType = uploadPersistence.loadMimeType(1);

// Retrieve IDs of all MIME types
long[] mimeTypeIds = uploadPersistence.getAllMimeTypeIds();

// Retrieve all MIME types by their IDs
MimeType[] mimeTypes = uploadPersistence.loadMimeTypes(mimeTypeIds);

// Retrieve the submissions for project with ID=1 and user with ID=1
Submission[] submissions = uploadPersistence.getUserSubmissionsForProject(1, 1);

// Retrieve all submissions for project with ID=1
submissions = uploadPersistence.getProjectSubmissions(1);

// Retrieve the images for submission with ID=1
SubmissionImage[] submissionImages = uploadPersistence.getImagesForSubmission(1);
```

5. Future Enhancements

At the current time, no future enhancements are expected for this component.