Deliverable Management 1.2 Component Specification

1. Design

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

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

The Deliverable Management component provides deliverable management functionalities. Various deliverables need to be fulfilled for a project during a specific phase. Usually a phase can be concluded only when all the required deliverables are present. The component defines an API to track the deliverables. The actual mechanism to verify each deliverable will be pluggable.

It also supports two types of specific deliverables, document upload and submission. Submission is a type of upload that has additional properties.

This component just defines interfaces for persistence layer, but doesn't provide implementations for them.

In the version 1.1 new SubmissionType entity was defined. And now Submission aggregates a SubmissionType instance. UploadManager, PersistenceUploadManager and UploadPersistence were updated with new methods for managing submission types.

In the version 1.2 new methods were added to UploadManager interface to support Studio contests, Submission and Upload entities were updated with Studio-specific properties, new entities MimeType, SubmissionImage were defined.

***FOR DEVELOPERS*: Please update the source code of this component to make it use generic collections properly (i.e. List<Deliverable> instead of List, etc). This is required due to compile target update from Java 1.4 to Java 1.5.**

1.1 Design Patterns

The DeliverablePersistence interface and implementations uses the **Strategy Pattern**, as do the DeliverableManager, UploadPersistence, and UploadManager interfaces and implementations.

DAO/DTO pattern – UploadManager and DeliverableManager are DAOs for Submission, SubmissionType, SubmissionStatus, Upload, Deliverable, UploadType, UploadStatus, MimeType, SubmissionImage DTOs.

1.2 Industry Standards

None

1.3 Required Algorithms

This component doesn't use any complicated algorithms. Please see implementation details in method docs provided in TCUML.

1.4 Component Class Overview

UploadManager:

The UploadManager interface provides the ability to persist, retrieve and search for persisted upload and submission modeling objects. This interface provides a higher level of interaction than the UploadPersistence interface. The methods in this interface break down into dealing with the 6 Upload/Submission modeling classes in this component, and the methods for each modeling class are fairly similar. However, searching methods are provided only for the Upload and Submission objects.

Implementations of this interface are not required to be thread safe.

Change in 1.1:

- Added methods for managing submission types.

Changes in 1.2:

- 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.

DeliverableManager:

The DeliverableManager interface provides the ability to persist, retrieve and search for persisted deliverable modeling objects. This interface provides a higher level of interaction than the DeliverablePersistence interface. This interface simply provides two methods to search a persistence store for Deliverables.

Implementations of this interface are not required to be thread safe.

IdentifiableDeliverableStructure:

This abstract class is a base class for all deliverable structures that have long integer identifier. It is a simple JavaBean that provides getter and setter for private ID field.

This class is mutable and not thread safe.

AuditedDeliverableStructure:

The AuditedDeliverableStructure is the base class for the modeling classes in this component. It holds the information about when the structure was created and updated. This class simply holds the four data fields needed for this auditing information and exposes both getters and setters for these fields.

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

Change in 1.2:

- ID property was extracted into IdentifiableDeliverableStructure that is now extended by this class.

NamedDeliverableStructure:

The NamedDeliverableStructure class extends the AuditedDeliverableStructure class to hold a name and description. Like AuditedDeliverableStructure, it is an abstract class. The NamedDeliverableStructure class is simply a container for the name and description. Both these data fields have the getters and setters.

This class should be very easy to implement, as all the methods just set the underlying fields.

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

Upload:

The Upload class is the one of the main modeling classes of this component. It represents an uploaded document. The Upload class is simply a container for a few basic data fields. All data fields in this class are mutable and have get and set methods.

This class should be very easy to implement, as all the methods just set the underlying fields.

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

Change in 1.2:

- Added description property.

Submission:

The Submission class is the one of the main modeling classes of this component. It represents a submission, which consists of an upload and a status. The Submission class is simply a container for a few basic data fields. All data fields in this class are mutable and have get and set methods.

This class should be very easy to implement, as all the methods just set the underlying fields.

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

Changes in 1.1:

- submissionType attribute was added together with getter and setter.
- isValidToPersist() was updated to ensure that submission type is specified.

Changes in 1.2:

- Submission holds a list of uploads instead of a single upload.
- Added fields specific to Studio submissions: thumb, userRank, images and extra.

SubmissionStatus:

The SubmissionStatus class is a support class in the modeling classes. It is used to tag a submission as having a certain status. For development, this class will be very simple to implement, as has no fields of its own and simply delegates to the constructors of the base class.

This class is mutable because its base class is mutable.

SubmissionType:

The SubmissionType class is a support class in the modeling classes. It is used to tag a submission as having a certain type. For development, this class will be very simple to implement, as has no fields of its own and simply delegates to the constructors of the base class.

This class is mutable because its base class is mutable.

UploadStatus:

The UploadStatus class is a support class in the modeling classes. It is used to tag an upload as having a certain status. For development, this class will be very simple to implement, as has no fields of its own and simply delegates to the constructors of the base class.

This class is mutable because its base class is mutable.

UploadType:

The UploadType class is a support class in the modeling classes. It is used to tag an upload as being of a certain type. For development, this class will be very simple to implement, as has no fields of its own and simply delegates to the constructors of the base class.

This class is mutable because its base class is mutable.

MimeType:

This class represents a MIME type. It is a simple JavaBean that provides getters and setters for all private attributes.

This class is mutable and not thread safe.

SubmissionImage:

This class represents an image associated with a submission. It is a simple JavaBean that provides getters and setters for all private attributes. Additionally it provides isValidToPersist() method to be used by upload managers.

This class is mutable and not thread safe.

Deliverable:

The Deliverable class is the one of the main modeling classes of this component. It represents an item that must be delivered for the project. The Deliverable class is simply a container for a few basic data fields, but unlike the Upload and Submission class, a deliverable is largely immutable. The data fields (except for completion date) have only get methods.

This class should be very easy to implement, as all the methods just get or set the underlying fields.

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

DeliverableChecker:

The DeliverableChecker interface is responsible for deciding if a deliverable is complete. If so, it sets the completion date of the deliverable. Only a single method exists in this interface. No concrete implementation of this interface is required in this component.

Implementations of this interface are not required to be thread safe.

DeliverableFilterBuilder:

The DeliverableFilterBuilder class supports building filters for searching for Deliverables. This class consists of 2 parts. The first part consists of static Strings that name the fields that are available for searching. All DeliverableManager implementations should use SearchBundles that are configured to use these names. The second part of this class consists of convenience methods to create common filters based on these field names. Development-wise, this class should prove quite simple, as all methods are a single line.

This class has only final static fields/methods, so mutability is not an issue.

SubmissionFilterBuilder:

The SubmissionFilterBuilder class supports building filters for searching for Submissions. This class consists of 2 parts. The first part consists of static Strings that name the fields that are available for searching. All UploadManager implementations should use SearchBundles that are configured to use these names. The second part of this class consists of convenience methods to create common filters based on these field names. Development-wise, this class should prove quite simple, as all methods are a single line.

This class has only final static fields/methods, so mutability is not an issue.

Change in 1.1:

- Added method createSubmissionTypeIdFilter().

UploadFilterBuilder:

The UploadFilterBuilder class supports building filters for searching for Uploads. This class consists of 2 parts. The first part consists of static Strings that name the fields that are available for searching. All UploadManager implementations should use SearchBundles that are configured to use these names. The second part of this class consists of convenience methods to create common filters based on these field names. Development-wise, this class should prove quite simple, as all methods are a single line.

This class has only final static fields/methods, so mutability is not an issue.

UploadPersistence:

The UploadPersistence interface defines the methods for persisting and retrieving the object model in this component. This interface handles the persistence of the upload related classes that make up the object model – Uploads, Submissions, UploadTypes, UploadStatuses, SubmissionStatuses, etc. This interface is not responsible for searching the persistence for the various entities. This is instead handled by an UploadManager implementation.

Implementations of this interface are not required to be thread-safe or immutable.

Change in 1.1:

- Added methods for managing submission types in persistence.

Changes in 1.2:

- 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.

DeliverablePersistence:

The DeliverablePersistence interface defines the methods for persisting and retrieving the object model in this component. This interface handles the persistence of the deliverable related classes that make up the object model – this consists only of the Deliverable class. Unlike UploadPersistence, the DeliverablePersistence interface (currently) has no support for adding items. Only retrieval is supported. This interface is not responsible for searching the persistence for the various entities. This is instead handled by a DeliverableManager implementation.

Implementations of this interface are not required to be thread-safe or immutable.

PersistenceUploadManager:

The PersistenceUploadManager class implements the UploadManager interface. It ties together a persistence mechanism, several Search Builder instances (for searching for various types of data), and several id generators (for generating ids for the various types). This class consists of several methods styles. The first method style just calls directly to a corresponding method of the persistence. The second method style first assigns values to some data fields of the object before calling a persistence method. The third type of method uses a SearchBundle to execute a search and then uses the persistence to load an object for each of the ids found from the search.

This class is immutable and hence thread-safe.

Change in 1.1:

- Added methods for managing submission types.

Changes in 1.2:

- 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.

PersistenceDeliverableManager:

The PersistenceDeliverableManager class implements the DeliverableManager interface. It ties together a persistence mechanism and two Search Builder instances (for searching for various types of data). The methods in this class use a SearchBundle to execute a search and then use the persistence to load an object for each of the ids found from the search.

This class is immutable and hence thread-safe.

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

UploadPersistenceException:

The UploadPersistenceException indicates that there was an error accessing or updating a persisted storage. This exception is used to wrap the internal error that occurs when accessing the persistence store. For example, in the SqlUploadPersistence implementation it is used to wrap SqlExceptions.

This exception is initially thrown in UploadPersistence implementations and from there passes through UploadManager implementations and back to the caller. It is also thrown directly by some UploadManager implementations.

DeliverablePersistenceException:

The DeliverablePersistenceException indicates that there was an error accessing or updating a persisted storage. This exception is used to wrap the internal error that occurs when accessing the persistence store. For example, in the SqlDeliverablePersistence implementation it is used to wrap SqlExceptions.

This exception is initially thrown in DeliverablePersistence implementations and from there passes through DeliverableManager implementations and back to the caller. It is also thrown directly by some DeliverableManager implementations.

IdAlreadySetException:
The IdAlreadySetException is used to signal that the id of one of the modeling classes has already been set. This is used to prevent the id being changed once it has been set. This exception is initially thrown in the setId methods of the modeling classes.

DeliverableCheckingException

The DeliverableCheckingException indicates that there was an error when determining whether a Deliverable has been completed or not.

PersistenceException
The PersistenceException indicates that there was an error accessing or updating a persisted resource store. This exception is used to wrap the internal error that occurs when accessing the persistence store.

1.6 Thread Safety

This component is not thread safe. This decision was made because the modeling classes in this component are mutable while the persistence classes make use of non-thread-safe components such as Search Builder. 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 various methods). These tradeoffs cannot 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 modeling classes thread safe can be done by simply adding the synchronized keyword to the various set and get methods. The manager classes would be harder to make thread safe. In addition to making manager methods synchronized, there would need to be logic added to handle conditions that occur when multiple threads are manipulating the persistence. For example, "Upload removed between SearchBuilder query and loadUpload call".

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

- Search Builder 1.3.1: Used for searching for deliverables, uploads, and submissions
- DB Connection Factory 1.0: Used in SQL persistence implementation to connect to the database. Also used by the Search Builder component.
- Database Abstraction 1.1: Defines CustomResultSet returned by the Search Builder component and used to retrieve the ids of the items selected by the search.
- ID Generator 3.0: Used to create ids when new Uploads/Submissions and related objects are created.

- Configuration Manager 2.1.5: Used to configure the DB Connection Factory and Search Builder components. Can also be used with the Object Factory component. Not used directly in this component.
- Logging Wrapper 1.2: Used for logging errors and debug information.
- Base Exception 2.0: Provides base class for custom exceptions.
- Data Validation 1.1.1: Used to validate string and integer values.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.management.deliverable
com.topcoder.management.deliverable.logging
com.topcoder.management.deliverable.persistence
com.topcoder.management.deliverable.search

3.2 Configuration Parameters

No direct configuration is used for this component. The Object Factory component can be used to create PersistenceUploadManagers, and PersistenceDeliverableManagers, if desired. The SearchBundles and IDGenerators needed can be configured or created programmatically. For configuration, see the component specs for these components.

When using the SQL database structure given in the deliverable_management.sql script, the SearchBundles passed to the PersistenceDeliverableManager and PersistenceUploadManager should be configured to use the following contexts (queries minus where clause):

For searching Deliverables:

```
SELECT UNIQUE deliverable_id, resource_id
FROM deliverable_lu
INNER JOIN resource
ON deliverable_lu.resource_role_id = resource.resource_role_id
WHERE
```

For searching Deliverables with submission:

```
SELECT DISTINCT deliverable_id, resource.resource_id,
submission_id
FROM deliverable_lu
INNER JOIN resource ON resource.resource_role_id =
deliverable_lu.resource_role_id
INNER JOIN upload
ON upload.resource_id = resource.resource_id
~~INNER JOIN submission~~
~~ON submission.upload_id = upload.upload_id~~
INNER JOIN upload_submission
ON upload_submission.upload_id = upload.upload_id
```

```
INNER JOIN submission
ON submission.submission_id = upload_submission.submission_id
INNER JOIN submission_status_lu
ON
submission.submission_status_id=submission_status_lu.submission_s
tatus_id
WHERE submission_status_lu.name='Active' AND
deliverable_lu.per_submission=1 AND
upload.project_id=resource.project_id
AND
```

For searching Submissions:

```
SELECT DISTINCT submission_id
FROM submission
~~INNER JOIN upload~~
~~    ON submission.upload_id = upload.upload_id~~
INNER JOIN upload_submission
    ON submission.submission_id = upload_submission.submission_id
WHERE
```

For searching Uploads:

```
SELECT upload_id
FROM upload
WHERE
```

3.3 Dependencies Configuration

None

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/dbschema.sql.

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

  b. Update test_files/SearchBundleManager.xml if needed.
- 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

Unfortunately, as this component cannot really be described in a full customer scenario without including all the dependency components, this demo will not be a customer oriented demo but a rundown of the requirements in a demo form. Where it makes sense, examples of what a customer might do with the information are shown.

4.3.1 Create Upload Manager

```
UploadPersistence uploadPersistence = ...

UploadManager manager =
new PersistenceUploadManager(
  uploadPersistence,
  searchBundleManager.getSearchBundle("Upload Search Bundle"),
  searchBundleManager.getSearchBundle("Submission  Search  Bundle"),
  IDGeneratorFactory.getIDGenerator("Upload Id Generator"),
  IDGeneratorFactory.getIDGenerator("Upload Type Id Generator"),
  IDGeneratorFactory.getIDGenerator("Upload  Status  Id  Generator"),
  IDGeneratorFactory.getIDGenerator("Submission Id Generator"),
  IDGeneratorFactory.getIDGenerator("Submission Status Id
  Generator"),
  IDGeneratorFactory.getIDGenerator("Submission Type Id
  Generator"));
```

4.3.2 Create an Upload and Submission (with supporting classes)

```
// Load tagging instances (also demonstrates
// manager interactions)
UploadType uploadType = manager.getAllUploadTypes()[0];
SubmissionStatus submissionStatus =
    manager.getAllSubmissionStatuses()[0];
SubmissionType submissionType =
    manager.getAllSubmissionTypes()[0];
UploadStatus uploadStatus = manager.getAllUploadStatuses()[0];

// 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 and manager

```
DeliverablePersistence deliverablePersistence = ...

// The checker is used when deliverable instances
// are retrieved
Map<String, DeliverableChecker> checker =
    new HashMap<String, DeliverableChecker>();
checker.put("name1", new CustomerDeliverableChecker());
checker.put("name2", new CustomerDeliverableChecker());

DeliverableManager manager =
new PersistenceDeliverableManager(
        deliverablePersistence,
        checker,
        searchBundleManager.getSearchBundle("Deliverable Search
         Bundle"),
         searchBundleManager.getSearchBundle("Deliverable With
         Submission Search Bundle"));

// Search for deliverables (see 4.3.5)
```

4.3.4 *Save the created Upload and Submission*

```
manager.createUpload(upload, "Operator #1");
manager.createSubmission(submission, "Operator #1");
// 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
manager.updateUpload(upload, "Operator #1");

// Remove it from the persistence
manager.removeUpload(upload, "Operator #1");

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

4.3.5 *Retrieve and search for uploads*

```
// Get an upload for a given id
Upload upload2 = manager.getUpload(14402);
// The properties of the upload can then be queried
// and used by the client of this component.  Submissions
// can be retrieved similarly.

// Search for uploads
// Build the uploads - this example shows searching for
// all uploads related to a given project and having a
// given upload type
Filter projectFilter =
      UploadFilterBuilder.createProjectIdFilter(953);
Filter uploadTypeFilter =
      UploadFilterBuilder.createUploadTypeIdFilter(4);

Filter fullFilter =
      SearchBundle.buildAndFilter(
            projectFilter, uploadTypeFilter);

// Search for the Uploads
Upload[] matchingUploads =
      manager.searchUploads(fullFilter);
```

```
// Submissions and Deliverables can be searched similarly by
// using the other FilterBuilder classes and the corresponding
// UploadManager or DeliverableManager methods.

// Get all the lookup table values.
UploadType[] uploadTypes = manager.getAllUploadTypes();
UploadStatus[] uploadStatuses = manager.getAllUploadStatuses();
SubmissionStatus[] submissionStatuses =
      manager.getAllSubmissionStatuses();
SubmissionType[] submissionTypes =
      manager.getAllSubmissionTypes();

// Alter a lookup table entry and update the persistence
uploadTypes[0].setName("Changed name");
manager.updateUploadType(uploadTypes[0], "admin");

// Lookup table entries can be created/removed through parallel
// methods to those shown in section 4.3.4
```

4.3.6 *Search for submissions with specific submission type*

```
long specificationSubmissionTypeId = 1;
Filter specificationSubmissionFilter =
   SubmissionFilterBuilder.createSubmissionTypeIdFilter(
           specificationSubmissionTypeId);

Submission[] specificationSubmissions =
   manager.searchSubmissions(specificationSubmissionFilter);
```

4.3.7 *Usage of methods defined in the version 1.2*

```
// Create upload manager (see section 4.3.1 above)
UploadManager manager = ...

// Create submission image
SubmissionImage submissionImage = new SubmissionImage();
submissionImage.setSubmissionId(submission.getId());
submissionImage.setImageId(1);
submissionImage.setSortOrder(1);
manager.createSubmissionImage(submissionImage, "admin");

// Update the submission image
submissionImage.setSortOrder(0);
manager.updateSubmissionImage(submissionImage, "admin");

// Remove the submission image
manager.removeSubmissionImage(submissionImage, "admin");

// Retrieve the MIME type with ID=1
MimeType mimeType = manager.getMimeType(1);

// Retrieve all MIME types
MimeType[] mimeTypes = manager.getAllMimeTypes();

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

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

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

5. Future Enhancements

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