Online Review Upload Services v1.0 Requirements Specification

1. Scope

1.1 Overview

The Online Review Submission Services will encapsulate the functionality of managing different Upload types. This component will be used in two different ways. Firstly, it will be used by the Online Review web site. Secondly, this component will provide a web services interfaces to allow external applications to upload files to Online Review.

1.2 Logic Requirements

1.2.1 Upload Services Class

The component has to provide an implementation of the UploadServices interface from section 2.1.2.

1.2.2 Uploads Types

During the processing of the different upload types all entities creation or modifications must be performed through the corresponding entity manager, see section 1.2.4.

1.2.2.1 Submission Upload

Submission Uploads are uploaded by submitters during the Submission phase.
When Uploading a Submission Upload the following tasks have to be performed:

- Check that the project exists.
- Check that the user exists and has the submitter role.
- Check that the Submission or Screening phase is open for the project.
- Create and persist the new Submission and Upload entities.
- Associate the submission with the submitter resource.
- Initiate the screening task.
- Change previous submissions status to “Deleted” if the project doesn’t allow multiple submissions per submitter. If the project allows multiple submissions, its property “Allow multiple submissions” will be true.

1.2.2.2 Final Fix Upload

Final Fix Uploads is uploaded by the winner submitter during then Final Fix phase.
When Uploading a Final Fix Upload the following tasks have to be performed:

- Check that the project exists.
- Check that the user exists, has the submitter role, and is the winner. The winner’s user id is stored in the Project property: “Winner External Reference ID”.
- Check that the Final Fix phase is open for the project.
- Create and persist the new Upload entity.
- Associate the submission with the submitter resource.
- Initiate the screening task.
- Change previous submissions status to “Deleted”.

1.2.2.3 Test Case Upload

Test Case Uploads are uploaded by reviewers during the Review phase of a development competition.
When Uploading a Test Case Upload the following tasks have to be performed:

- Check that the project exists.
- Check that the user exists and has at least one of the following roles: Accuracy Reviewer,

Failure Reviewer or Stress Reviewer.
- Check that the Review phase is open for the project.
- Create and persist the Upload entity.
- Associate the submission with the submitter resource.
- Change previous submissions status to “Deleted”.

1.2.3 Web Services Interface

The component will provide a web service interface to allow external systems to upload files. This component must implement the interface UploadExternalServices defined in section 2.1.2. If there is an exception during the processing of the request no file has to be stored.
Apache Axis v1.4 (http://ws.apache.org/axis/) must be used to implement the web service interface. Configuration for Axis is provided in section 2.1.2.1

1.2.3.1 File Storage Location

Files received through the ws interface will be stored on the local file system. This location has to be configurable using the Configuration Manager component.

1.2.3.2 File Unique Name Generation

A prefix will be appended to the file name before storing it in the file system. The prefix will be formed by a GUID, obtained from GUID Generator component, plus an underscore. The component has to generate this unique name until it doesn’t exist in the target location.

1.2.4 Entity Managers

The component users will provide an implementation of the ManagerProvider interface to be used by this component. Such implementation will be obtained using the Object Factory component.

1.2.5 Logging Policy

Each class must have its own logger instance using its full name to obtain it.

Fatal Level:
- Missing required configurations properties.
- Missing required files. Basically, every issue that makes the application failed to start.

Error Level:
- All Exceptions must be logged at least one time. A lot of caution has to be put on in order to not log the same exception several times. Given that sometimes this requirement is difficult to be accomplished, it shouldn’t be taken as a “must”.

Warn Level:
- Login failures.
- Authorization failures. Every time a user tries to perform some action for which he/she doesn’t have permission must be logged.
- Every time a phase can’t be opened or closed, an entry in the log must be added explaining why.

Info Level:
- When an entity is created, updated or deleted a log entry with the primary key must be added.
- When a phase is opened or closed a log entry containing the project id, phase id, phase type must be added.

- When configuration parameters are read from its persistence storage, for instance every time a component reads a parameter from Configuration Manager must be logged.

For all levels, whenever it's possible, add to the log message the id of the user doing the action. Finally, a great effort have to be put on generating a consistent logging message in order to easy the log viewing and filtering.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

Others applications or components will use the web service interface to upload submission to Online Review.

1.5 Future Component Direction

None.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

```
/**
 * Provides the actual implementations of Entities Managers
 */
public interface ManagersProvider {
        /**
         * @return a ResourceManager instance
         */
        ResourceManager getResourceManager();

        /**
         * @return a ProjectManager instance
         */
        ProjectManager getProjectManager();

        /**
         * @return a PhaseManager instance
         */
        PhaseManager getPhaseManager();

        /**
         * @return a ScreeningManager instance
         */
        ScreeningManager getScreeningManager();

        /**
         * @return a UploadManager instance
         */
        UploadManager getUploadManager();
}

/**
 * Defines the web service contract for managing different type of uploads.
 */
public interface UploadExternalServices {
        /**
         * Adds a new submission for an user in a particular project.
         * If the project allows multiple submissions for users, it will add the new submission
and return.
         * If multiple submission are not allowed for the project. Firstly, it will add the new
submission,
         * secondly mark previous submissions as deleted and then return.
         *
         * @param projectId the project's id
         * @param userId the user's id.
         * @param filename the file name to use.
         * @param submission the submission file data.
         * @return the id of the new submission.
         *
         * @throws InvalidProjectException if the project doesn't exists.
         * @throws InvalidProjectPhase if neither Submission or Screening phase are opened.
         * @throws InvalidUserException if the user doesn't exists or hasn't the submitter role.
         * @throws RemoteException if an internal exception occurs.
         */
        long uploadSubmission(long projectId, long userId, String filename, DataHandler
submission) throws RemoteException, InvalidProjectException, InvalidProjectPhase,
InvalidUserException;

        /**
         * Adds a new final fix upload for an user in a particular project.
         * This submission always overwrite the previous ones.
         *
         * @param projectId the project's id
         * @param userId the user's id.
         * @param filename the file name to use.
         * @param finalFix the final fix file data.
         * @return the id of the created final fix submission.
         * @throws InvalidProjectException if the project doesn't exists.
         * @throws InvalidProjectPhase if Final Fix phase isn't opened.
         * @throws InvalidUserException if the user doesn't exists or she/he is not winner
submitter.
         * @throws RemoteException if an internal exception occurs.
         */
        long uploadFinalFix(long projectId, long userId, String filename, DataHandler finalFix)
throws RemoteException, InvalidProjectException, InvalidProjectPhase, InvalidUserException;

        /**
         * Adds a new test case upload for an user in a particular project.
         * This submission always overwrite the previous ones.
         *
         * @param projectId the project's id.
         * @param userId the user's id.
         * @param filename the file name to use.
         * @param testCases the test cases data.
         * @return the id of the created test cases submission.
         * @throws InvalidProjectException if the project doesn't exists.
         * @throws InvalidProjectPhase if Review phase isn't opened.
         * @throws InvalidUserException if the user doesn't exists or hasn't the submitter role.
         * @throws RemoteException if an internal exception occurs.
         */
        long uploadTestCases(long projectId, long userId, String filename, DataHandler testCases)
throws RemoteException, InvalidProjectException, InvalidProjectPhase, InvalidUserException;

        /**
         * Sets the status of a existing submission.
         *
         * @param submissionId the submission's id
         * @param submissionStatus the submission status' id
         * @throws RemoteException if an internal exception occurs.
         * @throws InvalidSubmissionException if the submission doesn't exists.
         * @throws InvalidSubmissionStatusException if the submission status doesn't exists.
         */
        void setSubmissionStatus(long submissionId, int submissionStatusId) throws
RemoteException, InvalidSubmissionException, InvalidSubmissionStatusException;
}

/**
 * Defines the web service contract for managing different type of uploads.
 */
public interface UploadExternalServices {
        /**
```

```
         * Adds a new submission for an user in a particular project.
         * If the project allows multiple submissions for users, it will add the new submission
and return.
         * If multiple submission are not allowed for the project. Firstly, it will add the new
submission,
         * secondly mark previous submissions as deleted and then return.
         *
         * @param projectId the project's id
         * @param userId the user's id.
         * @param filename the file name to use.
         * @param submission the submission file data.
         * @return the id of the new submission.
         *
         * @throws InvalidProjectException if the project doesn't exists.
         * @throws InvalidProjectPhase if neither Submission or Screening phase are opened.
         * @throws InvalidUserException if the user doesn't exists or hasn't the submitter role.
         * @throws RemoteException if an internal exception occurs.
         *
         */
        long uploadSubmission(long projectId, long userId, String filename, DataHandler
submission) throws RemoteException, InvalidProjectException, InvalidProjectPhase,
InvalidUserException;


        /**
         * Adds a new final fix upload for an user in a particular project.
         * This submission always overwrite the previous ones.
         *
         * @param projectId the project's id
         * @param userId the user's id.
         * @param filename the file name to use.
         * @param finalFix the final fix file data.
         * @return the id of the created final fix submission.
         * @throws InvalidProjectException if the project doesn't exists.
         * @throws InvalidProjectPhase if Final Fix phase isn't opened.
         * @throws InvalidUserException if the user doesn't exists or she/he is not winner
submitter.
         * @throws RemoteException if an internal exception occurs.
         */
        long uploadFinalFix(long projectId, long userId, String filename, DataHandler finalFix)
throws RemoteException, InvalidProjectException, InvalidProjectPhase, InvalidUserException;

        /**
         * Adds a new test case upload for an user in a particular project.
         * This submission always overwrite the previous ones.
         *
         * @param projectId the project's id.
         * @param userId the user's id.
         * @param filename the file name to use.
         * @param testCases the test cases data.
         * @return the id of the created test cases submission.
         * @throws InvalidProjectException if the project doesn't exists.
         * @throws InvalidProjectPhase if Review phase isn't opened.
         * @throws InvalidUserException if the user doesn't exists or hasn't the submitter role.
         * @throws RemoteException if an internal exception occurs.
         */
        long uploadTestCases(long projectId, long userId, String filename, DataHandler testCases)
throws RemoteException, InvalidProjectException, InvalidProjectPhase, InvalidUserException;

        /**
         * Sets the status of a existing submission.
         *
         * @param submissionId the submission's id
         * @param submissionStatus the submission status' id
         * @throws RemoteException if an internal exception occurs.
         * @throws InvalidSubmissionException if the submission doesn't exists.
         * @throws InvalidSubmissionStatusException if the submission status doesn't exists.
         */
        void setSubmissionStatus(long submissionId, int submissionStatusId) throws
RemoteException, InvalidSubmissionException, InvalidSubmissionStatusException;
}
```

2.1.2.1 Apache Axis Server Configuration

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

        <handler name="URLMapper" type="java:org.apache.axis.handlers.http.URLMapper" />

        <typeMapping qname="OR:DataHandler" xmlns:OR="http://onlinereview.topcoder.com"
                serializer="org.apache.axis.encoding.ser.JAFDataHandlerSerializerFactory"
                deserializer="org.apache.axis.encoding.ser.JAFDataHandlerDeserializerFactory"
                languageSpecificType="java:javax.activation.DataHandler"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        />

        <service name="UploadService" provider="java:RPC">
                <namespace>http://onlinereview.topcoder.com</namespace>
                <!-- The class name have to be completed by the designer of the component -->
                <parameter name="className" value="com.cronos.onlinereview.services.uploads..." />
                <parameter name="allowedMethods" value="*" />
        </service>

        <service name="AdminService" provider="java:MSG">
                <parameter name="allowedMethods" value="*" />
                <parameter name="className" value="org.apache.axis.utils.Admin" />
        </service>
        <service name="Version" provider="java:RPC">
                <parameter name="allowedMethods" value="getVersion" />
                <parameter name="className" value="org.apache.axis.Version" />
        </service>

        <transport name="http">
                <requestFlow>
                        <handler type="URLMapper" />
                </requestFlow>
        </transport>
</deployment>
```

2.1.3 Environment Requirements

- Development language: Java 5.0
- Compile target: Java 5.0

2.1.4 Package Structure

com.cronos.onlinereview.services.uploads

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The location for storing the uploaded files.
- The actual ManagersProvider implementation.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

- XML – SOAP

3.2.2 TopCoder Software Component Dependencies:

- Resource Management v1.0
- Upload Management v1.0
- Project Management v1.0
- Phase Management v1.0

- Auto Screening Management v1.0
- GUID Generator v1.0
- Object Factory v2.0
- Configuration Manager v2.1.5
- Search Builder v1.3
- Logging Wrapper v1.2

3.2.3 Third Party Component, Library, or Product Dependencies:

- Apache Axis 1.4: http://ws.apache.org/axis/

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.