Online Review Upload Services v1.0 Component Specification

1. Design

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.

The design provides the interfaces described in RS and their default implementations. The interfaces related to the external webservices uses the internal UploadService interface to delegate all operations. All implementation are configurable via ObjectFactory and the name of file stored by external implementation is configurable

1.1 Design considerations

1.1.1 Apache Axis

In the Class Diagram the classes related with Axis are not shown. This is because the generation of web services through Axis is completely transparent. Axis, in fact, will generate a Dynamic Proxy using the extern interface UploadExternalServices. Or it can generates the WSDL using the Java2WDSL application.

1.2 Design Patterns

1.2.1 Strategy

DefaultUploadServices can use different implementations of ManagersProvider. DefaultUploadExternalServices can use different implementations of UploadServices.

1.2.2 Façade

The Façade pattern can be said to be used in the DefaultUploadServices and DefaultUploadExternalServices, as it provides streamlined access to managing submissions that uses numerous other components.

1.2.3 Delegation

DefaultUploadExternalServices delegates all methods to DefaultUploadServices

1.3 Industry Standards

- XML – SOAP

1.4 Required Algorithms

There aren't precise algorithms to describe because all methods are described with many details in the doc tab.

1.4.1 Logging

You must log the operations with the follow levels:
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.

1.5 Component Class Overview

1.5.1 com.cronos.onlinereview.services.uploads

This package holds all the main interfaces and classes in this component.

ManagersProvides
It provides instances of entity managers used in UploadServices interface. It contains all managers kept from the external components.

UploadServices
Defines the contract for managing different type of uploads and set status of

submissions.

UploadExternalServices
Defines the web service contract for managing different type of uploads and set status of submissions.

1.5.2 com.cronos.onlinereview.services.uploads.impl

This package holds the implementations.

DefaultManagersProvider
This is the default implementation of ManagersProvides.It provides instances of entity managers used in UploadServices class. The managers are loaded using Object Factory or set through constructor. The log is performed for all methods.

DefaultUploadServices
This is the default implementation of UploadServices interface. It manages different type of upload. It used all managers from ManagerProvider to perform several operations. All the methods are logged. It's possible to construct the instance through configuration and Object Factory and set wia constructor.

DefaultUploadExternalServices
This is the default implementation of UploadExternalServices interface. For all upload* methods it saves the file into fileStorageLocation and then delegates to UploadServices implementation. The setSubmission delegate directly to UploadServices instance.

1.6 Component Exception Definitions

ConfigurationException
This is the configuration exception of this component. It's thrown when there are invalid configuration values and missing configuration properties.

InvalidProjectException
This is thrown when a project doesn't exists.

InvalidProjectPhaseException
This is thrown when the project phase is invalid. The Project phase is a "Project" class but in phase package, different from "Project" Class in project package.

InvalidSubmissionException
This is thrown when a submission doesn't exists.

InvalidSubmissionStatusException
This is thrown when a submission status doesn't exists.

InvalidUserException
This is thrown when a user doesn't exists or when some error occurs related to the user (for example: she/he is not winner submitter)

PersistenceException
This exception wraps all exceptions related with the Persistence layer.

UploadServicesException
This is the parent exception of the component. It's thrown when some error occurs and is not covered from the other custom exceptions.

1.7 Thread Safety

The component is virtually thread-safe, and under expected conditions, it is effectively thread-safe. The reason is that this component uses numerous interfaces from other component. The classes in this component don't have a changeable state: it's impossible to change the variables in the classes. However, It's not sure that these external instances are thread safe, hence the thread safety of the classes depends totally to the thread safety of the external interfaces (managers) used.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.5
- Compile target: Java 1.5

2.2 TopCoder Software Components

- Configuration Manager 2.1.5
    - Used for configuration of implementation classes
- Object Factory 2.0.1
    - Used to create the several implementation of managers and other interfaces inside the default implementations.
- Base Exception 2.0
    - TopCoder standard for all custom exceptions.
- Logging Wrapper 2.0
    - Used for logging the operations.
- Search Builder 1.3.1
    - Used to create the Filters used in DefaultUploadServices
- Phase Management 1.0.2
    - Used to retrieve the Projects from Project Phases component.

- Project Phases 2.0
    - Used for the Project bean of this component (notice that this is different from Project in Project Management)
- Resource Management 1.1
    - Used to retrieve the project's resources
- Project Management 1.0
    - Used to retrieve the Projects of this component and for the Project Bean. (notice that this is different from Project in Project Phases)
- Deliverable Management v1.0
    - Used to retrieve Submission and the Upload entity.
- Auto Screening Management v1.0
    - Used to perform all screening tasks
- GUID Generator v1.0
    - Used to create the UUID for the submission's filename to store

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

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

3. Installation and Configuration

3.1 Package Names

com.cronos.onlinereview.services.uploads

com.cronos.onlinereview.services.uploads.impl

3.2 Configuration Parameters

3.2.1 DefaultManagersProvider

Parameter	Description	Details
objectFactoryNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required	Example: "com.topcoder.specify"
resourceManagerIdentifier	Key for the ResourceManager to pass to ObjectFactory. Required	Valid key in ObjectFactory
phaseManagerIdentifier	Key for the PhaseManager to pass to ObjectFactory. Required	Valid key in ObjectFactory
projectManagerIdentifier	Key for the ProjectManager to pass to ObjectFactory. Required	Valid key in ObjectFactory
screeningManagerIdentifier	Key for the ScreeningManager to pass to ObjectFactory. Required	Valid key in ObjectFactory
uploadManagerIdentifier	Key for the UploadManager to pass to ObjectFactory. Required	Valid key in ObjectFactory

3.2.2 DefaultUploadServices

Parameter	Description	Details
objectFactoryNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Optional	Example: "com.topcoder.specify"
managersProviderIdentifier	Key for the ManagersProvider to pass to ObjectFactory. Optional: default is DefaultManagersProvider	Valid key in ObjectFactory

3.2.3 DefaultUploadExternalServices

Parameter	Description	Details

objectFactoryNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Optional	Example: "com.topcoder.specify"
uploadServicesIdentifier	Key for the UploadServices to pass to ObjectFactory. Optional:default is DefaultUploadServices	Valid key in ObjectFactory
filenamePattern	The pattern to write the filename Optional: default is "{0}_{1}"	Valid string
fileStorageLocation	The path to store the submission files Required	Valid path

3.2.4 filenamePattern

These parameters are included in the pattern. See MessageFormat class for more information.

Parameter	Description
{0}	Represents the UUID generated
{1}	Represents the filaname passed from external web services

3.3 Dependencies Configuration

3.3.1 TopCoder dependencies

All the dependencies are to be configured according to their component specifications.

4. Usage Notes

4.1 Required steps to test the component

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

4.2 Required steps to use the component

None

4.3 Demo

4.3.1 Setup

Much of the setup for any demo involves properly configuring all the components. This demo will not go into that depth, but it will use a sample configuration XML file for each implementation.

```
<Config name="DefaultManagersProvider">

        <Property name="objectFactoryNamespace">
                <Value>myObjectFactory</Value>
        </Property>

        <Property name="resourceManagerIdentifier">
                <Value>resourceManager</Value>
        </Property>

        <Property name="phaseManagerIdentifier">
                <Value>phaseManager</Value>
        </Property>

        <Property name="projectManagerIdentifier">
                <Value>projectManager</Value>
        </Property>

        <Property name="screeningManagerIdentifier">
                <Value>screeningManager</Value>
        </Property>

        <Property name="uploadManagerIdentifier">
                <Value>uploadManager</Value>
        </Property>

</Config>



<Config name="DefaultUploadServices">

        <Property name="objectFactoryNamespace">
                <Value>myObjectFactory</Value>
        </Property>

        <Property name="managersProviderIdentifier">
                <Value>managersProvider</Value>
        </Property>

</Config>



<Config name="DefaultUploadExternalServices">

        <Property name="objectFactoryNamespace">
                <Value>myObjectFactory</Value>
        </Property>
```

```
        <Property name="uploadServicesIdentifier">
                <Value>uploadServices</Value>
        </Property>

        <Property name="filenamePattern">
                <Value>submission-{0}-{1}</Value>
        </Property>

        <Property name="fileStorageLocation">
                <Value>/opt/mySubmissions</Value>
        </Property>

</Config>
```

4.3.2 Usage of internal interface

A typical usage scenario involves the full use of the methods of internal interface, for example from a Web application:

```
// Create the implementation
UploadServices services = new DefaultUploadServices();
// Implementation created

// In this demo it will show the different phases of a project
// The project has the id=20, the user has the id=600.
// The names of submission will b e different depend from phase

// upload the submission
long idSubmission = services.uploadSubmission(30, 600, "submissionMyComponent");
// submission uploaded and the id of submission is returned
// Upload the final fixe
long idFinalFix = services.uploadFinalFix(30, 600, "finalFixMyComponent");
// final fix uploaded and the id of final fix is returned

// Upload the test case
long idTestCases = services.uploadTestCases(30, 600, "testCaseMyComponent");
// test cases uploaded and the id of test cases is returned

// change the status of submission previous uploaded to the status of id=1
services.setSubmissionStatus(idSubmission, 1, "" + 30);
// the submission status has changed
```

4.3.3 Deployment configuration

The deployment configuration of Axis will be:

```
        <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>
```

```
                <parameter name="className"
value="com.cronos.onlinereview.services.uploads.impl.DefaultUploadExternalServices" />
                <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>
```

4.3.4 Usage of the external interface with Axis: Java2WSDL and WSDL2Java

The typical usage of Axis is to generate the WSDL with Java2WSDL library. After this the WSDL will be generated and published. The client which will want use the web services will use the Axis client. The Axis client is generated with WSDL2Java library. At this point the interface equivalent to UploadExternalServices will be generate and the client will use the web services without explicit SOAP call, all mechanism will be completely transparent.

```
// Create the implementation
 UploadExternalServices services = new DefaultUploadExternalServices();
 // Implementation created

 // In this demo it will show the different phases of a project
 // The project has the id=20, the user has the id=600.
 // The names of submission will b e different depend from phase

 // Create the DataHandler
FileDataSource dataSource = new FileDataSource(TestHelper.TEST_FILES +
"sample.jar");
 DataHandler dataHandler = new DataHandler(dataSource);
 // DataHandler created

 // upload the submission
long idSubmission = services.uploadSubmission(30, 600, "submissionMyComponent",
dataHandler);
 // submission uploaded and the id of submission is returned

 // upload the final fix
long idFinalFix = services.uploadFinalFix(30, 600, "finalFixMyComponent",
dataHandler);
 // final fix uploaded and the id of final fix is returned

 // upload the test cases
long idTestCases = services.uploadTestCases(30, 100, "testCaseMyComponent",
dataHandler);
 // test cases uploaded and the id of test cases is returned

 // change the status of submission previous uploaded to the status of id=1
 services.setSubmissionStatus(idSubmission, 1, "" + 30);
 // the submission status has changed
```

4.3.5 Usage the WebServices in SOAP mode

In this case the Axis classes of client are not generated and the normal SOAP call is used:

```
   // Create the service
Service service = new Service();

// Create the caller and set the endpoint
Call call = (Call) service.createCall();
call.setTargetEndpointAddress(new java.net.URL(TestHelper.END_POINT));

// Set the operation
 call.setOperationName(new QName("http://onlinereview.topcoder.com",
"uploadTestCases"));

QName qName = new QName("DataHandler");
call.addParameter("op1", XMLType.XSD_LONG, ParameterMode.IN);
call.addParameter("op2", XMLType.XSD_LONG, ParameterMode.IN);
call.addParameter("op3", XMLType.XSD_STRING, ParameterMode.IN);
call.addParameter(qName, XMLType.MIME_DATA_HANDLER, ParameterMode.IN);

call.setReturnType(XMLType.XSD_LONG);

// Invoke the webservice.
long id = (Long) call.invoke(new Object[] { projectId, userId,
"sample_accuracy.jar", testCase });
```

The other operations are equivalent.

5. Future Enhancements

Other implementations of UploadServices