Studio Service 1.3 Component Specification

1. Design

Changes in the version 1.3 are marked with **blue**, new items are marked with **red**.

This component provides service to get, update, and create contest data; get, remove and update submission data; get some additional information like content's categories, statuses and file types. It defines web service interface and provides its EJB endpoint implementation. Component uses Contest Manager and Submission Manager for providing all operations with persistence. It should not access to persistence by itself.

Each contest has only one client who is owner of contest project. Only this client can access to contest, submissions, document etc. Administrators have also access to contests and their privileges are higher than client ones.

The design provides the interface depicted and the EJB implementation. To create the interface, firstly I updated the WSDL with the changes discussed in forum and other necessary changes. Then I generated the interface and all related classes and exception using *wsconsume* tool of JBoss, to make the interface consistent with WSDL. Then I removed the classes not necessary (request and responses classes) and annotations that are optional. Note that with JAX-WS and JBoss and WSDL is not necessary anymore because the WSDL will be generated by the endpoint interface.
At this point I did a refactoring: I changed the classes' names (with the same value in the annotations) and other changes: these changes are permitted and don't delete the functionality of the classes.
The component strongly uses the annotations in all levels: methods and classes. The exceptions use the fault bean to make the exceptions available to the client side. The administrator identity of the StudioServiceBean is set using the annotations.

In the version 1.3 the following entities were added:

- ContestGeneralInfoData
- ContestSpecificationsData
- MilestonePrizeData
- ContestMultiRoundInformationData

ContestData entity was updated to hold instances of the mentioned entities.

Also the following method were added to StudioService and its EJB implementation:

- getUserContests(username) – retrieves the list of data for contests for which the user with the given name is a resource.
- getMilestoneSubmissionsForContest(contestId) – gets the milestone submissions for the given contest.
- getFinalSubmissionsForContest(contestId:long) – gets the final

submissions for the given contest.

- setSubmissionMilestonePrize(submissionId, milestonePrizeId) – sets the specified milestone prize for submission with the given ID.

1.1 Design Patterns

1.1.1 Strategy

StudioServiceBean uses strategically the ContestManager (through the local interface) and SubmissionManager (through the local interface) implementations.

1.1.2 Adapter and Delegate

I can say that StudioServiceBean adapts the entities classes provided in this component to the entities of the Contest Manager and Submission Manager classes. Then StudioServiceBean delegates the work to these entities.

1.1.1 DTO

This component defines entities that are used as DTOs transferred from WCF service to WCF clients.

1.2 Industry Standards

- EJB (specifically, to ensure this component is compatible with EJB restrictions, as defined in http://java.sun.com/blueprints/qanda/ejb_tier/restrictions.html).

1.3 Required Algorithms

1.3.1 Logging

Almost all methods in StudioServiceBean (init included and except the getters) should be logged in the following manner:

- Method entrance and exit at INFO level.
- All exceptions and errors at ERROR level. This includes illegal arguments.
- the I/O errors of the underlying managers should be logged as ERROR level
- Any additional logging is at the developer's discretion.

The logging is optional, therefore if the logName is null then don't provide the logging.

1.3.2 Exception creation in the StudioServiceBean

The exceptions in the StudioServiceBean are constructed using the constructors with the fault beans (the classes with "fault" suffix). Firstly create the fault bean, then set the necessary information to the fault bean. At this point construct the exception with the fault bean.

This mechanism is necessary to create correctly the SOAP fault message.
The fault beans doesn't implement Serializable because they are not serialized as java objects, they are serialized in the SOAP environment.

1.3.3 Check the role

The role checking of the user or administrator is made in almost all methods. This is the algorithm:

- use the sessionContext.isCallerInRole("User") (or "Administrator")
- the "User" and "Administrator" are declared in the @DeclareRoles annotation
- if the role checking fails then throw the related authorization exception

1.3.4 Check the user id

The methods, in addition to the role checking, check the user id. This is the algorithm:

- get the Principal from SessionContext and cast to UserProfilePrincipal
- The member id is retrieved from UserProfilePrincipal using the related method to get the user id.
- if the user id checking fails then throw the related authorization exception

1.3.5 Arguments checking in the beans

The arguments checking in all bean classes is not performed, it is performed only in the StudioServiceBean. In this mode the user can set all the possible values in the beans.

1.3.6 Save and retrieve the documents

The contest is created using the ContestData entities, the mapping ContestData --> Contest is not direct.
This is the algorithm to save the document from ContestData:

- get the documentationUploads
- for uploadedDocument in documentationUploads:
  - create the Document and fill the attributes with the attributes of uploadedDocument (except file)
  - check if the document content exist in the contest manager using the method existDocumentContent (it will be added, see the forum)
  - if it doesn't exist then
    - save the document and the document content (the file attribute) using the saveDocument method in contest Manager

The get contest method related to get the documents is the mirror of the previous algorithm:

- for each document in contest.documents:
  - create the UploadedDocument and fill its attributes with the document's attributes
  - get the document content from contestManager passing the document id
  - fill the UploadedDocument.file with the document content

If some I/O errors occur in the algorithm then don't throw exceptions, simply log the exceptions. In the first case don't save the document and in the last case don't set the *file* attribute.

1.3.5 ContestData <--> Contest mapping

The ContestData mapping to Contest is not direct. There are some attributes that need an explanation.

ContestData field	**Contest field**
contestCategoryId: use the id of contestCategory	contestCategory: Get the ContestCategory from contestManager using the id
durationInHours: use contest.endDate – contest.startDate in hour. LaunchDateAndTime is the startDate	StartDate and endDate: the startDate is the launchDateAndTime, the endDate is launchDateTime+durationInHours
finalFileFormat: use the extension from Contest.ContestCategory.StudioFileType.extension	When create the ContestCategory and create the StufioFileType, set the extension
otherFileFormats: get the extensions from Contest.fileTypes	Create the StudioFileType and set the extensions
contestPayload: get the config entities config from Contest.config; create a new ContestPayload for each config and set the name,value and description	This attribute should be only retrieved
contestDescriptionAndRequirements: get the "contestDescriptionAndRequirements" config from Contest.config; iterate over all names and get the description	Create the Config and add it to the Contest.config: set the name as the name of parameter ("contestDescriptionAndRequirements") and the value is the value of parameter
requiredOrRestrictedColors: get the "requiredOrRestrictedColors" config from Contest.config; iterate over all names and get the description	Create the Config and add it to the Contest.config: set the name as the name of parameter ("requiredOrRestrictedColors") and the value is the value of parameter
requiredOrRestrictedFonts: get the "requiredOrRestrictedFonts" config from Contest.config; iterate over all names and get the description	Create the Config and add it to the Contest.config: set the name as the name of parameter ("requiredOrRestrictedFonts") and the value is the value of parameter
sizeRequirements: get the "sizeRequirements" config from Contest.config; iterate over all names and get the description	Create the Config and add it to the Contest.config: set the name as the name of parameter ("sizeRequirements") and the value is the value of parameter
otherRequirementsOrRestrictions: get the "otherRequirementsOrRestrictions" config from Contest.config; iterate over all names and get the description	Create the Config and add it to the Contest.config: set the name as the name of parameter ("otherRequirementsOrRestrictions") and the value is the value of parameter

Prizes: retrieve the prizes from contestManager using the contestId (this method will be added also to contestManager, see the forum)	Persist the prizes for contest (the related method will be added also to contestManager, see the forum)

1.3.6 SubmissionData <--> Submission mapping

The ContestData mapping to Contest is not direct. There are some attributes that need an explanation.

SubmissionData	**Submission**
placement	rank
paidFor	Get the submissionPayment from SubmissionManager and get the PaymentStatus from SubmissionPayment and check if the id is equal to the configured ids: paid or unpaid and set true or false respectively (if the id is not equal to all two values then set to false)
markedForPurchase	Get the submissionPayment from SubmissionManager a and get the PaymentStatus from SubmissionPayment nd check if the id is equal to the configured id: if the id is not equal to the value then set to false otherwise to true
removed: return true (see the forum)	There is not this field
submissionType	Submission.getType().getDescription()

1.4 Component Class Overview

1.4.1 com.topcoder.studio.service

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

ContestStatusData
It is the DTO class which is used to transfer contest status data. The information can be null or can be empty, therefore this check is not present in the setters. It's the related to the equivalent ContestStatus entity.
This class is not thread safe because it's highly mutable

SubmissionData
It is the DTO class which is used to transfer submission data. The information can be null or can be empty, therefore this check is not present in the setters. It's the related to the equivalent Submission entity.
This class is not thread safe because it's highly mutable
Changes v1.3: Added the following field with the corresponding getter and setter: submissionType. Updated @XmlType annotation.

ContestData

It is the DTO class which is used to transfer contest data. The information can be null or can be empty, therefore this check is not present in the setters. It's the related to the equivalent Contest entity.
This class is not thread safe because it's highly mutable
Changes in v1.3: Added the following fields with the corresponding getters and setters: generalInfo, specifications, multiRoundData, milestonePrizeData and nonWinningSubmissionsPurchased. @XmlType annotation was updated with new fields.

PrizeData
It is the DTO class which is used to transfer prize data. The information can be null or can be empty, therefore this check is not present in the setters. It is related with the Prize entity.
This class is not thread safe because it's highly mutable

StudioService
This is service interface which defines operations available for client. It allows providing different operations under contest and submission. It allows getting, updating, and creating contest data; get, remove and update submission data; get some additional information like content's categories, statuses and file types.
This interface defines the webservices endpoint with the operation. It uses the annotations to defined the requests and responses definitions.
Changes in v1.3: Added methods getUserContests(), getMilestoneSubmissionsForContest(), getFinalSubmissionsForContest() and setSubmissionMilestonePrize().

ContestPayload
It is the DTO class which is used to transfer contest type configuration data. The information can be null or can be empty, therefore this check is not present in the setters. the ContestPayload is related with a configuration parameter (Config class). It can only be retrieved.
This class is not thread safe because it's highly mutable

UploadedDocument
It is the DTO class which is used to transfer document and document content data. The information can be null or can be empty, therefore this check is not present in the setters. It's the related to the equivalent Document entity.
This class is not thread safe because it's highly mutable

ContestMultiRoundInformationData
It is the DTO class which is used to transfer contest multiround information. The information can be null or can be empty, therefore this check is not present in the setters. It's the related to the equivalent ContestMultiRoundInformation entity.

MilestonePrizeData
It is the DTO class which is used to transfer contest milestone prize data. The

information can be null or can be empty, therefore this check is not present in the setters. It's the related to the equivalent MilestonePrize entity.

ContestSpecificationsData
It is the DTO class which is used to transfer contest specifications data. The information can be null or can be empty, therefore this check is not present in the setters. It's the related to the equivalent ContestSpecifications entity.

ContestGeneralInfoData
It is the DTO class which is used to transfer contest general info data. The information can be null or can be empty, therefore this check is not present in the setters. It's the related to the equivalent ContestGeneralInfo entity.

1.4.2 com.topcoder.studio.service.ejb

StudioServiceBean
This is service interface which defines operations available for client. It allows providing different operations under contest and submission. It allows getting, updating, and creating contest data; get, remove and update submission data; get some additional information like content's categories, statuses and file types.
This is the EJB implementation of the StudioService interface webservice endpoint. It uses an instance of ContestManager and an instance of SubmissionManager (injected as EJB) to perform the logic of the methods. The webservices annotations are presents in the endpoint interface, this bean contains only the annotation to connect the endpoint interface and this implementation. The security is provided programmatically and also with annotations. The exceptions are constructed using the fault bean because the fault message can be consumed as SOAP message, this is necessary because it's a webservices.
This implementations is designed to be used by the related interface and also by a different webservices client: all the response, request and exceptions are automatically transformed to SOAP element.
Thread safety: this class is thread safe if the managers used are thread safe. Considering that probably the managers beans will use the transactions, this stateless bean is thread safe.
Changes in v1.3: Added methods getUserContests(), getMilestoneSubmissionsForContest(), getFinalSubmissionsForContest() and setSubmissionMilestonePrize(). Private methods for converting Contest to ContestData and back were updated to support new Contest fields. The same for conversion of Submission to SubmissionData.

IllegalArgumentWSFault
This class represents the "illegal_argument_fault" element in WSDL. It's contained in the related exception-illegalArgumentMessage class. It was generated by JBoss tools and I changed the name and other little changes.
This class is not thread safe because it's highly mutable

UserNotAuthorizedFault
This class represents the "user_not_authorized_fault" element in WSDL. It's contained in the related exception-message class. It was generated by JBoss tools and I changed the name and other little changes.
This class is not thread safe because it's highly mutable

StatusNotFoundFault
This class represents the "status_not_found_fault" element in WSDL. It's contained in the related exception-message class, it's the fault bean. It was generated by JBoss tools and I changed the name and other little changes.
This class is not thread safe because it's highly mutable

StatusNotAllowedFault
This class represents the "status_not_allowed_fault" element in WSDL. It's contained in the related exception-message class, it's the fault bean. It was generated by JBoss tools and I changed the name and other little changes.
This class is not thread safe because it's highly mutable

ContestNotFoundFault
This class represents the "contest_not_found_fault" element in WSDL. It's contained in the related exception-message class, it's the fault bean. It was generated by JBoss tools and I changed the name and other little changes.
This class is not thread safe because it's highly mutable

DocumentNotFoundFault
This class represents the "document_not_found_fault" element in WSDL. It's contained in the related exception-message class, it's the fault bean. It was generated by JBoss tools and I changed the name and other little changes.
This class is not thread safe because it's highly mutable

1.5 Component Exception Definitions

ContestNotFoundException
This exception is thrown automatically by the web service interface when a contest is not found. It is related with "contest_not_found_faultMsg" fault in WSDL, but it contains only annotations about "contest_not_found_fault" because the messages are automatically constructed. It was generated by JBoss tools and I changed the name, the inheritance and other changes.
The constructors with ContestNotFoundFault and the getter are necessary for the correct webservices serialization/deserialization. The other constructors are added for the future implementations of the service interface.

DocumentNotFoundException
This exception is thrown automatically by the web service interface when a document is not found. It is related with "document_not_found_faultMsg" fault in

WSDL, but it contains only annotations about "document_not_found_fault" because the messages are automatically constructed. It was generated by JBoss tools and I changed the name, the inheritance and other changes.
The constructors with DocumentNotFoundFault and the getter are necessary for the correct webservices serialization/deserialization. The other constructors are added for the future implementations of the service interface.

ProjectNotFoundException
This exception is thrown automatically by the web service interface when a project is not found. It is related with "project_not_found_faultMsg" fault in WSDL, but it contains only annotations about "project_not_found_fault" because the messages are automatically constructed. It was generated by JBoss tools and I changed the name, the inheritance and other changes.
The constructors with ProjectNotFoundFault and the getter are necessary for the correct webservices serialization/deserialization. The other constructors are added for the future implementations of the service interface.

PersistenceException
This exception is thrown automatically by the web service interface when an error occurs in the persistence layer. It wraps the generic persistence exceptions (not specific thrown by the interface of this component) thrown by the DocumentManagerBean used in the ejb implementation. It is related with "persistence_faultMsg" fault in WSDL, but it contains only annotations about "persistence_fault" because the messages are automatically constructed. It was generated by JBoss tools and I changed the name, the inheritance and other changes.
The constructors with PersistenceFault and the getter are necessary for the correct webservices serialization/deserialization. The other constructors are added for the future implementations of the service interface.

StatusNotAllowedException
This exception is thrown automatically by the web service interface when a status is not allowed. It is related with "status_not_allowed_faultMsg" fault in WSDL, but it contains only annotations about "status_not_allowed_fault" because the messages are automatically constructed. It was generated by JBoss tools and I changed the name, the inheritance and other changes.
The constructors with StatusNotAllowedFault and the getter are necessary for the correct webservices serialization/deserialization. The other constructors are added for the future implementations of the service interface.

StatusNotFoundException
This exception is thrown automatically by the web service interface when a status is not found. It is related with "status_not_found_faultMsg" fault in WSDL, but it contains only annotations about "status_not_found_fault" because the messages are automatically constructed. It was generated by JBoss tools and I changed the name, the inheritance and other changes.

The constructors with StatusNotFoundFault and the getter are necessary for the correct webservices serialization/deserialization. The other constructors are added for the future implementations of the service interface.

StudioServiceException
This exception extends the BaseCriticalException. It is also the parent exception class for all the other custom exceptions in the Studio package.

1.6 Thread Safety

The thread safety is an important aspect since this component is a set of web services. The Java beans are not thread safe, but it's not required that they would be thread safe because they are used within a single thread. The StudioServiceBean is thread safe because it uses an instance of ContestManager and an instance of SubmissionManager (through the remote interfaces) and the implementations of these interfaces are required to be thread safe: the implementations of these interfaces will be thread safe because it will use the transaction mechanism.
Therefore the component is completely thread safe.

Thread safety of the component was not changed in the version 1.3.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

- Base Exception 2.0
  - TopCoder standard for all custom exceptions.
- Logging Wrapper 2.0
  - Used for logging the operations.
- Contest Manager 1.3
  - provides the contest manager and the contest entities
- Submission Manager 1.2
  - provides the submission manager and the submission entities
- Search Builder 1.4.1
  - defines Filter entity used in this component
- Contest and Submission Entities 1.2
  - provides the data entities used in this design
- JBoss Login Module 2.0

- defines the UserProfilePrincipal class used in this design

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

- J2EE 5
- EJB 3.0
- JAX-WS

3. Installation and Configuration

3.1 Package Names

com.topcoder.studio.service

com.topcoder.studio.service.ejb

3.2 Configuration Parameters

3.2.1 StudioServiceBean

These parameters must be set in the configuration of the EJB (see the demo):

Parameter	Description	Details
logName	Name of the log used to retrieve the Log with LogManager Optional, the logging is optional	A valid log name, not null, not empty
draftStatusId	The id of draft contest status	A valid id, long >=0
scheduledStatusId	The id of scheduled status used to retrieve the check the draft status.	A valid id, long >=0
unactiveStatusId	The id of the inactive status used to retrieve the check the draft status.	A valid id, long >=0
activeStatusId	The id of the active status.	A valid id, long >=0
submissionPaidStatusId	The id of status of submission paid	A valid id, long >=0
submissionUnpaidStatusId	The id of status of submission unpaid	A valid id, long >=0
submissionMarkedForPurchase	The id of status of submission marked for purchase	A valid id, long >=0
submissionPassedStatus	The id of status of submission that has passed the review.	A valid id, long >=0

reviewFailedStatusId	The id of review status of a submission that failed the screen.	A valid id, long >=0
submissionContentBaseURI	The base URI of submission content	A valid URI
submissionContentSubmissionIdParameterName	The parameter name of submission id parameter used in the submission content URI	A string not empty
submissionContentSubmissionTypeParameterName	The parameter name of submission type parameter used in the submission content URI	A string not empty
submissionContentPaidParameterValue	The value of paid type parameter.	A string not empty
submissionContentUnpaidParameterValue	The value of unpaid type parameter.	A string not empty
defaultMimeType	The default mimetype that should be used when no match is found.	A string not empty
contestPropertyShortSummaryId	The id for the Contest property "Short Summary".	A valid id, long >=0
contestPropertyContestOverviewTextId	The id for the Contest property "Contest Overview Text".	A valid id, long >=0
contestPropertyColorRequirementsId	The id for the Contest property "Color Requirements".	A valid id, long >=0
contestPropertyFontRequirementsId	The id for the Contest property "Font Requirements".	A valid id, long >=0
contestPropertySizeRequirementsId	The id for the Contest property "Size Requirements".	A valid id, long >=0
contestPropertyOtherRequirementsId	The id for the Contest property "Other Requirements".	A valid id, long >=0
contestPropertyFinalFileFormatId	The id for the Contest property "Final File Format".	A valid id, long >=0
contestPropertyOtherFileFormatsId	The id for the Contest property "Other File Formats".	A valid id, long >=0
contestPropertyCreateUserHandleId	The id for the Contest property "Create User Handle".	A valid id, long >=0
contestPropertyRequiresPreviewImageId	The id for the Contest property "Requires Preview Image".	A valid id, long >=0
contestPropertyRequiresPreviewFileId	The id for the Contest property "Requires Preview File".	A valid id, long >=0
contestPropertyMaximumSubmissionsId	The id for the Contest property "Maximum Submissions".	A valid id, long >=0
contestPropertyEligibilityId	The id for the Contest property "Eligibility".	A valid id, long >=0
contestPropertyNotesOnWinnerSelectionId	The id for the Contest property "Notes on Winner Selection".	A valid id, long >=0

contestPropertyPrizeDescriptionId	The id for the Contest property "Prize Description".	A valid id, long >=0
defaultContestEligibilityText	The default text for the Contest property "Eligibility".	A string not empty
defaultContestNotesOnWinnerSelectionText	The default text for the Contest property "Notes on Winner Selection".	A string not empty
defaultContestPrizeDescriptionText	The default text for the Contest property "Prize Description".	A string not empty
defaultContestNotesOnSubmissionFiles	The default text for the Contest property "*Notes on Submission File(s)".	A string not empty
contestPrizeTypeId	The id for the Contest property "Contest PrizeType Id".	A valid id, long >=0
clientSelectionPrizeTypeId	The id for the Contest property "Client Selection PrizeType Id".	A valid id, long >=0
documentBasePath	The base path for the documents.	A string not empty
additionalSubmissionPurchasePriceRatio	The additional submission purchase price ratio	Positive float
submissionRemovedStatusId	The id for submission status "removed".	A valid id, long >=0
contestPropertyDigitalRunPointsId	The id for the Contest property "Digital Run Points".	A valid id, long >=0
contestPropertyContestAdministrationFeeId	The id for the Contest property "Contest Administration Fee".	A valid id, long >=0
autoPaymentsEnabled	The value that indicates whether auto payments are enabled.	Boolean
pactsServiceLocation	The location of the PACTS services for submission auto-paymetns.	A string not empty
forumBeanProviderUrl	The forum bean provider URL.	A string not empty
submissionSiteBaseUrl	The submission site base url.	A string not empty
submissionSiteFilePath	The submission site file path.	A string not empty

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

In the server side there is no need of other configuration: web service.xml and the JAX-RPC are not required because the annotations provide automatically these information.
About the deployment in JBoss, you can see here:
http://jbws.dyndns.org/mediawiki/index.php?title=Quick_Start#Consuming_web_services

" Simply wrap up the service implementation class, the endpoint interface and any custom data types in a JAR and drop them in the deploy directory. No additional deployment descriptors required. Any meta data required for the deployment of the actual web service is taken from the annotations provided on the implementation class and the service endpoint interface. JBossWS will intercept that EJB3 deployment (the bean will also be there) and create an HTTP endpoint at deploy-time: "

4.3.2 Usage

```
    //Create the competition and set its values
    ContestData contestData = new ContestData();
    contestData.setContestId(20);
    contestData.setContestDescriptionAndRequirements("The requirements
are...");
    contestData.setContestCategoryId(20);
    contestData.setCreatorUserId(30);

    //Create the upload documentation
    List<UploadedDocument> documentationUpload = new
ArrayList<UploadedDocument>();
    UploadedDocument document = new UploadedDocument();
    document.setContestId(20);
    document.setDescription("Design Studio Distribution for Kink DAO");
    document.setDocumentId(40);
    document.setFile("designDistribution".getBytes());
    documentationUpload.add(document);

    //add the design distribution documentation to competition
    contestData.setDocumentationUploads(documentationUpload);

    //launch the competition now
```

```
    // create the timestamp with the current date
    GregorianCalendar cal = new GregorianCalendar();
    // create the XMLGregorianCalendar
    XMLGregorianCalendar xmlCal =
DatatypeFactory.newInstance().newXMLGregorianCalendar(cal);
    contestData.setLaunchDateAndTime(xmlCal);
    contestData.setName("Kink DAO studio design");
    contestData.setOtherRequirementsOrRestrictions("nothing");

    //create the competition associated with a tc project
    studioService.createContest(contestData, 40);
    //now the competition is created and persisted

    //Get this competition back
    contestData = studioService.getContest(20);
    //the competition is retrieved
    //You can get the previous values through the getters, the getters are the
same as the setters

    //get the previous competition which is related with the project with id=40
    List<ContestData> contestDatas =
studioService.getContestsForProject(40);
    //the competition is retrieved (only 1 item)

    //get the file types
    String fileTypes = studioService.getSubmissionFileTypes();
    //it now contains "pdf","rtf","jpg","png",etc... separated by commas

    //remove the previous created document from the related competition
    studioService.removeDocumentFromContest(document);
    //the design distribution is removed now

    // New features of the version 1.3

    // Get all contests for which test_user is a resource
    List<ContestData> testUserContests =
        studioService.getUserContests("test_user");

    // Get milestone submissions for contest with ID=1
    List<SubmissionData> milestoneSubmissions =
        studioService.getMilestoneSubmissionsForContest(1);

    // Get final submissions for contest with ID=1
    List<SubmissionData> finalSubmissions =
        studioService.getFinalSubmissionsForContest(1);

    // Set milestone prize with ID=1 to submission with ID=2
    studioService.setSubmissionMilestonePrize(2, 1);
```

5. Future Enhancements

Other methods can be added to the service.