

# **Studio Contest Manager 1.0 Component Specification**

## **1. Design**

This component provides operations on contests like add new contest, get contest, update contest, update contest status; CRUD (Create, Read, Update, Delete) operations on contest status; CRUD operations on competition documents; get client by contest and project; CRUD operations on the contest category; and CRUD operations for the configuration parameters. It also provides an ability to save files to the server's file system and retrieve them from server's file system. Component runs as a stateless EJB. It uses Hibernate JPA implementation to work with persistence. It is used by Studio Service and can be used for the other purposes.

The ContestManagerBean is the stateless session bean to provide required functionalities. Both local and remote EJB interfaces are supported by it.

The ContentDocumentManager interface is used by the bean to manage the document content, and an implementation is provided to manage the document content in a file system server using SocketChannel from Java NIO library. A simple file system server (SocketDocumentContentServer class) is also provided in this design.

### **1.1 Design Patterns**

**Strategy Pattern** - DocumentContentManager interface and its implementations are used as strategy in the ContestManagerBean.

### **1.2 Industry Standards**

JPA  
EJB 3.0

### **1.3 Required Algorithms**

#### *1.3.1 Logging*

Logging should be done only if it's enabled. (The logger instance variable is not null).

##### **1.3.1.1 Logging in ContestManagerBean**

Logging should be done in all public methods of ContestManagerBean.

Entrance and exit of methods should be logged at the INFO level, the method arguments should also be logged, and if the method argument is a data entity object, only its id needs to be logged.

Exception should be logged at the ERROR level, both exception message and exception stack trace should be logged.

##### **1.3.1.2 Logging in SocketDocumentContentServer**

Logging should also be done in the SocketDocumentContentServer class and its inner SocketDocumentContentWorker class.

Server start and stop actions should be logged at the INFO level in start/stop methods. The timestamp should also be logged.

Exception should be logged at the ERROR level, both exception message and exception stack trace should be logged.

### 1.3.2 Annotations for ContestManagerBean's public methods

All public methods should be marked with the following annotations:

@PermitAll - This annotation indicates all declared roles for the bean class are allowed to perform the operation. Right now only "Administrator" role is declared for the bean class.

@TransactionAttribute(TransactionAttributeType.REQUIRED) - This annotation indicates the transaction is required.

### 1.3.3 Rollback Transaction in ContestManagerBean

SessionContext.setRollbackOnly() should be called if exception occurs in methods of ContestManagerBean to ensure the transaction is rolled back correctly.

### 1.3.4 Request and Response format used in SocketDocumentContentManager and SocketDocumentContentServer

NOTE: All String values should be encoded into byte array using the "UTF-8" charset.

#### 1.3.4.1 Common Request Format

The first byte indicates the request type. Refer to the 1.3.3.3 - 1.3.3.5 for more details.

#### 1.3.4.2 Common Response Format

The first byte indicates the request is processed successfully or not.

If the value of first byte is 0x01, it means the operation succeeds, and it may have no more data in response if the operation doesn't require return value, otherwise, the following bytes represents the value to return. Refer to 1.3.3.3 - 1.3.3.5 for more details about the successful response format.

If the value of first byte is 0x00, it means the operation is failed, and the following data represents the error message string:

The next two bytes indicates the length of the error message value in bytes.

The following bytes are for the error message value. Its length should be as defined above.

#### 1.3.4.3 SocketDocumentContentManager.saveDocumentContent Method

Request Format:

The first byte indicates the request type, and its value should always be 0x01 indicating it's a saveDocumentContent request.

The next two bytes indicates the length of the path value in bytes.

The following bytes are for the path value. Its length should be as defined above.

The next four bytes indicates the length of documentContent byte array.

The following bytes are for the documentContent byte array. Its length should be as defined above.

Successful Response Format:

It doesn't require any return value, so the successful response only contains a single byte as mentioned in 1.3.3.2.

#### 1.3.4.4 SocketDocumentContentManager.getDocumentContent Method

Request Format:

The first byte indicates the request type, and its value should always be 0x02 indicating it's a getDocumentContent request.

The next two bytes indicates the length of the path value in bytes.

The following bytes are for the path value. Its length should be as defined above.

Successful Response Format:

(If the corresponding file exists on the file sever)

Other than the first byte, it will contain the following bytes:

The next four bytes indicates the length of documentContent byte array.

The following bytes are for the documentContent byte array. Its length should be as defined above. (There can be no documentContent data if the corresponding file is empty.)

(If the corresponding file doesn't exist on the file sever)

There is no more data except the first byte.

#### 1.3.4.5 SocketDocumentContentManager.existDocumentContent

Request Format:

The first byte indicates the request type, and its value should always be 0x03 indicating it's an existDocumentContent request.

The next two bytes indicates the length of the path value in bytes.

The following bytes are for the path value. Its length should be as defined above.

Successful Response Format:

Other than the first byte, it contains a second byte indicating the document content exists or not. If the value of the second byte is 0x01, it means the document content exists, and if its value is 0x00, it means the document content doesn't exist.

## 1.4 Component Class Overview

### 1.4.1 Package *com.topcoder.service.studio.contest*

- **ContestManager:** It provides operations on contest like add new contest, get contest, update contest, update contest status; CRUD operations on contest status; CRUD operations on competition document; get client by contest and project; CRUD operations on the contest category; CRUD operations for the configuration parameters; operations to save, retrieve or check existence of document content; operations to get all contest statuses, categories and studio file types.
- **ContestManagerLocal:** This EJB Local interface extends the ContestManager interface, and it is marked with @Local annotation to indicate it's an EJB Local interface.
- **ContestManagerRemote:** This EJB Remote interface extends the ContestManager interface, and it is marked with @Remote annotation to indicate it's an EJB Remote interface.
- **DocumentContentManager:** This interface will be used in ContestManager implementation to provide operations to save document content, get document content and check document content existence.

### 1.4.2 Package *com.topcoder.service.studio.contest.bean*

- **ContestManagerBean:** This bean class implements the ContestManagerLocal and ContestManagerRemote interfaces. It is a stateless session bean with @Stateless annotation.

### 1.4.3 Package *com.topcoder.service.studio.contest.documentcontentmanagers*

- **SocketDocumentContentManager:** This class implements the DocumentContentManager interface and it acts as a socket client to save document content to a file server or get document content information from a file server.

#### 1.4.4 *Package com.topcoder.service.studio.contest.documentcontentservers*

- **SocketDocumentContentServer**: This class acts as the file system server for the SocketDocumentContentManager class.
- **SocketDocumentContentWorker**: It implements the Runnable interface, and it is used to process clients' requests, and send back proper responses to client.

### 1.5 **Component Exception Definitions**

#### 1.5.1 *System Exceptions*

- **IllegalArgumentException**: It is thrown when the passed-in argument is illegal.  
NOTE: A string is empty if its length is 0 after being trimmed.
- **IOException**: It is thrown if any I/O error occurs.

#### 1.5.2 *Custom Exceptions*

- **ContestConfigurationException**: This runtime exception extends the BaseRuntimeException, and it is thrown if the configured value is invalid, it is also used to wrap the underlying exceptions when loading the configured values.
- **ContestManagementException**: This exception extends the BaseCriticalException, and it is used to cover almost all general errors (except generic ones) thrown from this component. It is also used as the super class for all custom exceptions in this component.
- **EntityNotFoundException**: This exception extends the ContestManagementException, and it is thrown when the entity cannot be found in the persistence, but it's not supposed to be.
- **EntityAlreadyExistsException**: This exception extends the ContestManagementException, and it is thrown when the entity already exists in the persistence, but it's not supposed to be.
- **DocumentContentManagementException**: This exception is thrown from DocumentContentManager interface and its implementations if any error (not including I/O error) occurs when managing the document content.
- **InvalidRequestException**: This exception extends the BaseCriticalException, and it is thrown by SocketDocumentContentWorker.run method if the received request data is invalid.

### 1.6 **Thread Safety**

The ContestManagerBean is a stateless session bean. It must be thread-safe as it will be accessed from multiple threads by multiple users when deployed in the EJB container.

The variables in ContestManagerBean are set only once in the initialize method immediately after it is created, and their values will never be changed afterwards, so they won't affect the thread-safety of this bean.

All dependent TCS component are either thread-safe or used thread-safely, so they won't affect the thread-safety of this design.

The implementations of DocumentContentManager interface are required to be thread-safe, and so does the provided SocketDocumentContentManager implementation.

The JPA operations in the bean are executed in container managed transaction, so ContestManagerBean can be safely accessed from multiple threads in EJB container.

The SocketDocumentContentServer class and its inner class are thread-safe as they are all properly synchronized (either use synchronized methods or synchronize the mutable variables properly).

## 2. Environment Requirements

### 2.1 Environment

- Java 5.0+
- JBoss 4.2
- Hibernate 3.2 and higher

### 2.2 TopCoder Software Components

- **Logging Wrapper 2.0:** Used to log invocation information and exceptions.
- **Base Exception 2.0:** Custom exceptions in this design extend the base exceptions from this component.
- **Contest and Submission Entities 1.0:** It provides the data entities used in this design.
- **Project Service 1.0:** The used Project data entity is from this component.

### 2.3 Third Party Components

None.

## 3. Installation and Configuration

### 3.1 Package Name

com.topcoder.service.studio.contest  
com.topcoder.service.studio.contest.bean  
com.topcoder.service.studio.contest.documentcontentmanagers  
com.topcoder.service.studio.contest.documentcontentservers

### 3.2 Configuration Parameters

Configuration for ContestManagerBean (in env-entry of the deployment descriptor)

Parameter Name	Parameter Description	Parameter Value
unitName	Represents unit name to obtain the EntityManager instance from SessionContext. Required.	Must be a non-empty string.
activeContestStatusId	Represents the contest status id of the active contest status. Required.	Must be a long value.
loggerName	Represents the logger to get a Log instance from LogManager. Optional.	Must be a non-empty string if present.
defaultDocumentPathId	Represents the id of FilePath used as the default document path. Required.	Must be a long value.
documentContentManagerClassName	Represents the fully qualified class name of DocumentContentManager implementation class. Required.  NOTE: The DocumentContentManager implementation must have a constructor taking a Map<String, Object> argument, in order to be created by this design using Reflection API.	Must be a non-empty string.

documentContentManagerAttributeKeys	Represents the attribute keys used as the keys of the Map<String, Object> argument in order to create the DocumentContentManager implementation instance. Required.	Must be non-empty string. If it contains multiple attribute keys, they should be delimited by comma. NOTE: The attribute key should NEVER contain comma(s).
[attributeKey]	Represents the corresponding value for specific attributeKey configured above. Required.	Can be any value.  DocumentContentManager implementation is supposed to validate this value.

### 3.3 Dependencies Configuration

The dependent data entities should be deployed properly.

## 4. Usage Notes

### 4.1 Required steps to test the component

- Extract the component distribution.
- Follow [Dependencies Configuration](#).
- Execute 'ant test' within the directory that the distribution was extracted to.

### 4.2 Required steps to use the component

Deploy this component in an EJB 3.0 container, and then follow the demo to call the beans. The beans should be configured in ejb-jar.xml file like this:

```
<enterprise-beans>
  <session>
    <ejb-name>contestManager</ejb-name>
    <remote>com.topcoder.service.studio.contest.ContestManagerRemote</remote>
    <local>com.topcoder.service.studio.contest.ContestManagerLocal</local>
  </session>

  <ejb-class>com.topcoder.service.studio.contest.bean.ContestManagerBean</ejb-class>

  <session-type>stateless</session-type>
  <transaction-type>Container</transaction-type>
  <env-entry>
    <env-entry-name>unitName</env-entry-name>
    <env-entry-value>contestManager</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
  <env-entry>
    <env-entry-name>activeContestStatusId</env-entry-name>
    <env-entry-value>1</env-entry-value>
    <env-entry-type>java.lang.Long</env-entry-type>
  </env-entry>
  <env-entry>
    <env-entry-name>loggerName</env-entry-name>
    <env-entry-value>contestManagerLogger</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
</env-entry>
```

```

        <env-entry-name>defaultDocumentPathId</env-entry-name>
        <env-entry-value>1</env-entry-value>
        <env-entry-type>java.lang.Long</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>documentContentManagerClassName</env-entry-name>
        <env-entry-
value>com.topcoder.service.studio.contest.documentcontentmanagers.SocketD
ocumentContentManager</env-entry-value>
        <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>documentContentManagerAttributeKeys</env-entry-
name>
        <env-entry-value>serverAddress,serverPort</env-entry-value>
        <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>serverAddress</env-entry-name>
        <env-entry-value>127.0.0.1</env-entry-value>
        <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>serverPort</env-entry-name>
        <env-entry-value>2100</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>

</session>

</enterprise-beans>

```

And we can configure it in the jboss.xml like this:

```

<jboss>
    <enterprise-beans>
        <session>
            <ejb-name>contestManager</ejb-name>
            <jndi-name>contestManager</jndi-name>
        </session>
    </enterprise-beans>
</jboss>

```

### 4.3 Demo

```

// create and start the file server to listen on port = 2100
// with backlog = 10 and loggerName = "test"
SocketDocumentContentServer server =
    new SocketDocumentContentServer(2100, 10, "test");
server.start();

// NOTE: Only user with "Administrator" role can access the APIs exposed
// by the stateless session bean.

// lookup the bean from JNDI
ContestManagerRemote bean = (ContestManagerRemote)
    new InitialContext().lookup("contestManager");

// get contest by id = 1, the corresponding Contest should be retrieved
Contest contest = bean.getContest(1);

// update contest name, the new name should be updated to persistence

```

```
contest.setName("new-name");
bean.updateContest(contest);

// get document by id = 2, the corresponding Document should be retrieved
Document doc = bean.getDocument(2);

// add the document (with id = 2) into contest (with id = 1),
// the document will be associated with the contest in persistence.
bean.addDocumentToContest(2, 1);

// save document content of the document (with id = 2) to file server,
// assume the document's path value is: c:\contests,
// and its systemFileName value is: test.txt
// the data will be saved into the "c:\contests\test.txt" file on
// the file server.
byte[] data = "test-data".getBytes();
bean.saveDocumentContent(2, data);

// get the saved document content of the document (with id = 2)
// the returned array should contain the same bytes as the data variable.
byte[] content = bean.getDocumentContent(2);

// remove the document (with id = 2) from contest (with id = 1),
// True should be returned to indicate the association between the
// document and contest is removed successfully.
boolean flag = bean.removeDocumentFromContest(2, 1);

// stop the file server
server.stop();

// get contest status with id = 1
ContestStatus contestStatus = bean.getContestStatus(1);

// remove contest category with id = 1
bean.removeContestCategory(1);

// get contest config with id = 1
ContestConfig contestConfig = bean.getConfig(1);

// get contest type config with id = 1
ContestTypeConfig contestTypeConfig = bean.getContestTypeConfig(1);

// get contest prizes in contest with id = 1
List<Prize> prizes = bean.getContestPrizes(1);

// get all contest statuses
List<ContestStatus> contestStatuses = bean.getAllContestStatuses();

// get client for contest with id = 1
long clientId = bean.getClientForContest(1);
```



## **5 Future Enhancements**

Add more DocumentContentManager implementations, and add more sophisticated file servers.