[TopCoder]

# DR Contest Manager Component v1.0 Specification

## 1. Design

This component provides operations on digital run contests like add new contest, get contest, update contest, search contests; CRUD (Create, Read, Update, and Delete) operations on contest types; CRUD operations on contest calculators; Component runs as a stateless EJB. It uses Hibernate JPA implementation to work with persistence. It is used by *Digital Run Service 1.0* component.
DDL, entities and mappings are provided by Digital Run Entities 1.0 component.

### *1.0.1 Quick Overview of definitions and concepts*

This is a very straight forward implementation of a stateless EJB with JPA support and with search filter support for TrackContest based searches. Please note that we also added externally configurable auditing interceptors.

### *1.0.2 Anatomy of the proposed design/architecture*

Here we will discuss the aspects of the configurable interceptors:
Although the @Interceptors annotation allows you to apply the profiling interceptor easily, it does force you to modify and recompile your class every time you want to remove profiling from or add it to a particular method or EJB. This is NOT what we want here. We want the interceptors to be fully configurable and pluggable. Using XML bindings instead might be a better approach. Because the EJB 3.0 specification supports partial XML deployment descriptors, it is quite painless and easy to apply an interceptor through XML. Here we show the example of the setting up the create auditor interceptor for the `createTrackContest` method:

```
<ejb-jar
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
            http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd" version="3.0">
  <assembly-descriptor>
    <interceptor-binding>
      <ejb-name>DigitalRunContestManagerBean</ejb-name>
        <interceptor-class>
            com.topcoder.service.digitalrun.contest.audit.interceptors
            .AuditCreateContestInterceptor
         </interceptor-class>
         <method-name>createTrackContest</method-name>
           <method-params>
             <method-param>
                 com.topcoder.service.digitalrun.contest. TrackContest
             </method-param>
           </method-params>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

### 1.1 Design Patterns

The ***factory method pattern*** is used to allow for each creation of Filter instances to search against in the manager.
The ***strategy design pattern*** is utilized in the way the logger and auditor si plugged in.
The ***DAO design pattern*** (and the ***DTO design pattern***) is utilized in this design to abstract the CRUD implementation for specific entities.

### 1.2 Industry Standards

EJB 3.0, JPA, XML (for configuration of the EJB and interceptors)

### 1.3 Required Algorithms

No special algorithms were employed. Please consult the IML documentation for implementation details.

### 1.4 Component Class Overview

**DigitalRunContestManager**
This is a contract for operations on digital run contests like add new contest, get contest, update contest, search contests; CRUD (Create, Read, Update, and Delete) operations on contest types; CRUD operations on contest calculators. It also supports a search operation (based on Filters) for the TrackContest based searches.

**DigitalRunContestManagerLocal**
This EJB local interface defines contract to manage the digital run contest data in persistence. It inherits all the method of the DigitalRunContestManager interface. This interface should be marked with @Local annotation representing it's a local interface. Implementations of it needs to be stateless session bean, and should be thread-safe when used in EJB container.

**DigitalRunContestManagerRemote**
This EJB remote interface defines contract to manage the digital run contest data in persistence. It inherits all the method of the DigitalRunContestManager interface. This interface should be marked with @Remote annotation representing it's a remote interface. Implementations of it needs to be stateless session bean, and should be thread-safe when used in EJB container.

**ContestFilterFactory**
This is a factory for the creation of Filters which based on some specific criteria will return instances of TrackContest (i.e. the Filter would return such instances)
We currently support the following filters:

| Search By | Supported Operation |
|---|---|
| Contest ID | Equals/In |
| Contest Type ID | Equals/In |
| Contest Desc | Equals/In/Like |
| Track Id | Equals/In |

**BaseAuditContestInterceptor**
This is a base class for all the audit interceptors to extend.

**AuditRemoveContestInterceptor**
This is a specific interceptor to deal with Contest Remove action audit for any of the remove* methods in the DigitalRunContestManagerBean class.

**AuditUpdateContestInterceptor**
This is a specific interceptor to deal with Contest Update action audit for any of the update* methods in the DigitalRunContestManagerBean class.

**AuditCreateContestInterceptor**
This is a specific interceptor to deal with Contest Create action audit for any of the create* methods in the DigitalRunContestManagerBean class.

## 1.5 Component Exception Definitions

### *1.5.1 Custom Exceptions*

**DigitalRunContestManager**
This is the top level general Digital Run Contest Manager exception which signals issues with the processing request; it should be extended by other more specific exceptions for processing.
NOTE: It is marked with @ApplicationException annotation.

**EntityNotFoundException**
This exception extends the DigitalRunContestManager, and it is thrown from the DigitalRunContestManager interface and its implementations if the entity cannot be found.
NOTE: It is marked with @ApplicationException annotation.

**EntityExistsException**
This exception extends the DigitalRunTrackManagerException, and it is thrown from the DigitalRunTrackManager interface and its implementations if the entity already exists in such situations as creation. Note that this is also thrown by the persistence layer.
NOTE: It is marked with @ApplicationException annotation.

**ContestManagerConfigurationException**
This runtime exception extends the BaseRuntimeException, and it is thrown from the DigitalRunContestManagerBean class' initialize method if the configured value(s) is/are invalid, it is also used to wrap the underlying exceptions when loading the configured values.

**PersistenceException**
This is a general Persistence Exception which will specify that there were issues with actual persistence such as connection issues or SQL issues (if applicable) or JPA issues (if applicable)

### *1.5.2 System and general java exceptions*

**IllegalArgumentException**

This is thrown when illegal argument is passed. This will include empty strings (i.e. strings that when trimmed have length 0) as well as when null argument is passed.

**Exception**
This is used in interceptors to signal issues with processing.

### 1.6 Thread Safety

The DigitalRunContestManagerBean is a stateless session bean, and it must be thread-safe as it will be accessed from multiple threads. The variables in this bean class are set once in the initialize method immediately after the bean is constructed, and their values will never be changed afterwards, so they won't affect the thread-safety of this bean. All the entities used in this design are mutable, but as they are never used in multiple threads simultaneously, so they won't affect the bean class' thread-safety as well. And finally the JPA operations executed in the bean are thread-safe, so the bean is thread-safe.

## 2. Environment Requirements

### 2.1 Environment

- Development language: Java1.6
- Compile target: Java1.6

### 2.2 TopCoder Software Components

- **Base Exception 2.0:** This is used as the base for all custom exceptions defined here.
- **Digital Run Entities 1.0**: This provides the DDL, entities and mappings used by this component.
- **Search Builder 1.3.2:** This provides the Filter and SearchBundle classes that enable querying of data with filters.
- **Logging Wrapper 2.0**: This is used to log actions and exceptions in the bean's public methods.
- **Configuration API 1.0:** This is used as the main abstraction for configuration.
- **Configuration Persistence 1.0.1:** This will be used as the actual specific configuration reader for this component.
- **Object Factory 2.1:** This is used to instantiate configured objects as needed by this component.
- **Object Factory Configuration API plugin 1.0:** This is used to instantiate configured objects as needed by this component.
- **Object Factory ConfigManager plugin 1.0:** This is used to instantiate configured objects as needed by this component.
- **Auditor 2.0:** This will be used to specify and create the specific auditor for this component. It is provided/plugged-in through the interceptors.

*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

None.

## 3. Installation and Configuration

### 3.1 Package Name

*com.topcoder.service.digitalrun.contest*
*com.topcoder.service.digitalrun.contest.audit.interceptors*

### 3.2 Configuration Parameters

#### 3.2.1 Bean configuration

##### 3.2.1.1 ENC entries

| Parameter Name | Description | Value(s) |
|---|---|---|
| unit_name | The unit name for getting the Entity Manager (***Required***) | String. Should be a valid, non-empty name. |
| config_file_name | The configuration file name for the bean ***(Required***) | String. Should be a valid, non-empty name. |
| config_namespace | This the configuration namespace for the configuration (***Required***) | Configuration sub-tree which holds the necessary information for logger name and DAO implementation. |

##### 3.2.1.2 configuration properties inside the configuration object

| Parameter Name | Description | Value(s) |
|---|---|---|
| log_name | The logger name (***Optional***) | String. Should be a valid, non-empty name. |
| track_contest_dao_token | This is the token used to get the TrackContestDAO instance via object factory ***(Required)*** | String. Should be a valid, non-empty name |
| track_contest_type_dao_token | This is the token used to get the TrackContestTypeDAO instance via object factory ***(Required)*** | String. Should be a valid, non-empty name |
| result_calculator_dao_token | This is the token used to get the TrackContestResultCalculatorDAO instance via object factory ***(Required)*** | String. Should be a valid, non-empty name |
| object_factory_config | This a subchild property root which will hold the Object Factory configuration. (***Required***) | A whole sub-tree of other properties. This will specifically hold the information used by all the *dao_token" properties |

##### 3.2.1.3 configuration properties for TrackContestDAO

| Parameter Name | Description | Value(s) |
|---|---|---|
| search_bundle_manager_name space | The namespace for the Search Builder (R***equired***) | String. Should be a valid, non-empty name. |
| bundle | Bundle name (R***equired***) | String. Should be a valid, non-empty name. |

#### 3.2.2 Interceptor configuration

##### 3.2.2.1 ENC entries

| Parameter Name | Description | Value(s) |
|---|---|---|
| unit_name | The unit name for getting the Entity Manager (***Required***) | String. Should be a valid, non-empty name. |
| config_file_name | The configuration file name for the auditor configuration ***(Required***) | String. Should be a valid, non-empty name. |
| config_namespace | This the configuration namespace for the configuration (***Required***) | Configuration sub-tree which holds the necessary information for auditor. |

#### 3.2.3 Auditor Configuration

| Parameter Name | Description | Value(s) |
|---|---|---|
| auditor_config | The auditor configuration sub-child object. (***Required***) | Configuration sub-tree which holds the necessary information for Auditor implementation. |
| auditor_user | The user who is auditing (***Required***) | String. Should be a valid, non-empty name. |

### 3.3 Dependencies Configuration

None at this point.

## 4. Usage Notes

### 4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- See README for details
- 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 make call to the exposed DigitalRunContestManagerRemote from client. The DigitalRunContestManagerBean (remote) bean should be configured in an ejb-jar.xml file like this:

```
<enterprise-beans>
  <session>
          <ejb-name>DigitalRunContestManagerBean</ejb-name>

<remote>com.topcoder.service.digitalrun.contest.DigitalRunContestManagerRemote</re
mote>

<local>com.topcoder.service.digitalrun.contest.DigitalRunContestManagerLocal</loca
l>
      <ejb-
class>com.topcoder.service.digitalrun.contest.DigitalRunContestManagerBean</ejb-
class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <env-entry>
        <env-entry-name>unit_name</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>HibernateDRContestPersistence</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>config_file_name</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>config.properties</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>config_namespace</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>DigitalRunContestManagerBean</env-entry-value>
      </env-entry>
      <persistence-context-ref>
       <persistence-context-ref-name>
              HibernateDRContestPersistence
       </persistence-context-ref-name>
       <persistence-unit-name>HibernateDRContestPersistence</persistence-unit-
name>
       <persistence-context-type>Transaction</persistence-context-type>
      </persistence-context-ref>
  </session>
```

```
</enterprise-beans>
```

Here we show the example of the setting up the create auditor interceptor for the createTrackContest method:

```
<ejb-jar
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
            http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd" version="3.0">
  <assembly-descriptor>
    <interceptor-binding>
      <ejb-name>DigitalRunContestManagerBean</ejb-name>
        <interceptor-class>
            com.topcoder.service.digitalrun.contest.audit.interceptors
            .AuditCreateContestInterceptor
         </interceptor-class>
         <method-name>createTrackContest</method-name>
           <method-params>
             <method-param>
                 com.topcoder.service.digitalrun.contest. TrackContest
             </method-param>
           </method-params>
      </interceptor-binding>
   </assembly-descriptor>
</ejb-jar>
```

## 4.3 Demo

### *1.4.1 A simple user customer demo.*

```
// lookup the bean from JNDI
DigitalRunContestManagerRemote manager
       = new InitialContext().lookup(“DigitalRunContestManager”);


// create a new contest
TrackContest contest = manager.createTrackContest(contest);

// update contest
manager.updateTrackContest(contest);

// get contest
contest = manager.getTrackContest(contest.getId());

// delete contest
manager.removeTrackContest(contest.getId());

// search contests, with simple IDs search
long[] contestIds = {2,3,4};
Filter filter = ContestFilterFactory.createContestIdsInFilter(contestIds);
List<TrackContest> contests = manager.searchTrackContests(filter);

// create a new contest type
TrackContestType contestType = manager.createTrackContestType(contest);

// update contest type
manager.updateTrackContestType(contestType);

// get contest type
contestType = manager.getTrackContestType(contestType.getId());

// delete contest type
manager.removeTrackContestType(contestType.getId());

// get all contest types
List<TrackContestType> contestTypes = manager.getAllTrackContestTypes();

// create a new calculator
```

```
TrackContestResultCalculator calculator =
manager.createTrackContestResultCalculator(contest);

// update calculator
manager.updateTrackContestResultCalculator(calculator);

// get calculator
calculator = manager.getTrackContestResultCalculator(calculator.getId());

// delete calculator
manager.removeTrackContestResultCalculator(calculator.getId());

// get all calculators
List<TrackContestResultCalculator> calculators =
manager.getAllTrackContestResultCalculators();
```

## 5. Future Enhancements

None at this point.