DR Track Manager 1.0 Component Specification

1. Design

This component provides operations on digital run track related entities like create/update/remove/get/get active/search/add and remove project type for Track and create/update/remove/get/get all for TrackType, ProjectType, TrackStatus and PointsCalculator. Component runs as a stateless EJB. The ejb will make use of five daos to interact with the persistence. This component is used by Digital Run Service component.

1.1 Design Patterns

Strategy: Different implementations of DAO interfaces can be plugged into the DigitalRunTrackManagerBean. Application can also used different implementations of DigitalRunTrackManager interface.
DAO: DigitalRunTrackDAO, DigitalRunPointsCalculatorDAO, DigitalRunTrackTypeDAO, DigitalRunTrackStatusDAO and DigitalRunProjectTypeDAO are dao interfaces.
Factory: DigitalRunTrackFilterFactory implmenets this pattern.

1.2 Industry Standards

- EJB 3.0
- XML
- JPA
- Hibernate

1.3 Required Algorithms

1.3.1 How to perform logging

Logging will be performed in all public methods and in AbstractDAO.getEntityManager protected method. Following are the rules for logging:

- Entrance and exit of methods should be logged at the INFO level
- Exception should be logged at the ERROR level

1.4 Component Class Overview

DigitalRunTrackManager
This interface defines the contract for classes that will manage digital run track entities into persistence. The implementing classes will provide the following functionality:
-create a track/ track type /track status/points calculator into persistence
-update a track/ track type /track status/points calculator into persistence
-remove a track/ track type /track status/points calculator from persistence
-get a track/ track type /track status/points calculator/project type identified by its id from persistence
-search the track entities in persistence that match a given filter

-get all track type /track status/points calculator/project type from persistence
-get active tracks
-add/remove project type from track
Implementations are required to be thread safe.

DigitalRunTrackManagerLocal
This EJB Local interface extends the DigitalRunTrackManager, and it is marked with @Local annotation to indicate it's an EJB Local interface.
Implementation will run in an EJB container and thread safety depends on the container. Generally the implementation is thread safe when it's running inside the container.

DigitalRunTrackManagerRemote
This EJB Remote interface extends the DigitalRunTrackManager, and it is marked with @Remote annotation to indicate it's an EJB Remote interface.
Implementation will run in an EJB container and thread safety depends on the container. Generally the implementation is thread safe when it's running inside the container.

DigitalRunTrackDAO
This interface defines the contract for classes that will manage Track entities into persistence. The implementing classes will provide the following functionality:
-create a Track entity into persistence
-update a Track entity into persistence
-remove a Track entity identified by its id from persistence
-get a Track entity identified by its id from persistence
-search the Track entities in persistence that match a given filter
-get active Track entities
-add/remove ProjectType to/from Track
Implementations are not required to be thread safe.

DigitalRunTrackTypeDAO
This interface defines the contract for classes that will manage digital run track type entities into persistence. The implementing classes will provide the following functionality:
-create a digital run track type entity into persistence
-update a digital run track type entity into persistence
-remove a digital run track type entity identified by its id from persistence
-get a digital run track type entity identified by its id from persistence
-get all digital run track type entities from persistence
Implementations are not required to be thread safe.

DigitalRunTrackStatusDAO
This interface defines the contract for classes that will manage digital run TrackStatus entities into persistence. The implementing classes will provide the following functionality:

-create a digital run TrackStatus entity into persistence
-update a digital run TrackStatus entity into persistence
-remove a digital run TrackStatus entity identified by its id from persistence
-get a digital run TrackStatus entity identified by its id from persistence
-get all digital run TrackStatus entities from persistence
Implementations are not required to be thread safe.

DigitalRunProjectTypeDAO
This interface defines the contract for classes that will manage ProjectType entities into persistence. The implementing classes will provide the following functionality:
-get a digital run ProjectType entity identified by its id from persistence
-get all digital run ProjectType entities from persistence
Implementations are not required to be thread safe.

DigitalRunPointsCalculatorDAO
This interface defines the contract for classes that will manage digital run PointsCalculator entities into persistence. The implementing classes will provide the following functionality:
-create a digital run PointsCalculator entity into persistence
-update a digital run PointsCalculator entity into persistence
-remove a digital run PointsCalculator entity identified by its id from persistence
-get a digital run PointsCalculator entity identified by its id from persistence
-get all digital run PointsCalculator entities from persistence
Implementations are not required to be thread safe.

ConfigurationProvider
This class retrieves and stores the configuration for the current bean (and possibly for future beans and different implementations of DigitalRunTrackManager interface). It uses Configuration Persistence component to retrieve the configuration from a file. The user can also provide the configuration object by calling setConfiguration method. The configuration can be retrieved by calling getConfiguration method. This class was created because file access in ejb is forbidden by ejb specification.
This class is thread safe because all the methods are synchronized.

DigitalRunTrackFilterFactory
This class provides static methods that create filters that can be used by the DigitalRunTrackManager implementations to retrieve track related entities, that match the filter criteria, from the persistence. The methods provided return filters with:
-equals/in operation for track id/track status id/track type id/project type id
-equals/like operation track description
-equals/greater or equal than/lower or equal than for track start date and track end date

This class also provides and/or/not filters so that the user does not need to interact with Search Builder component to create such filters.
This class is thread since it has no state.

DigitalRunTrackManagerBean
This class is a stateless session bean that manages track related entities. It implements the DigitalRunTrackManagerLocal and DigitalRunTrackManagerRemote interfaces. All public methods from this class perform logging using LoggingWrapper component. This bean will not access the persistence directly; it will use five daos to access the persistence. These daos will be creates in initialize post construct method using Object Factory. The logger will also be initialized in initialize method.
This class will be annotated with:
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
Since the container will manage the transactions this class can be considered as thread safe. This bean is also immutable.

AbstractDAO
This abstract class holds the common field of all the five concrete dao implementations. It has a SessionContext and an unit name used to obtain an EntityManager instance and a Log instance used to perform logging if logging is desired; if logging is not desired this field will remain null.It provides a method that obtains an EntityManager instance.
It is not thread safe because it is mutable. Anyway, setters are meant to be called only once so there will be no thread safety issues.

JPADigitalRunTrackTypeDAO
This class implements DigitalRunTrackTypeDAO interface. It also extends the AbstractDAO class. This class manages TrackType entities in a JPA persistence (currently the JPA persistence will use Hibernate as a provider but any provider can be used). Each public method performs logging.
This class is not completely thread safe because it doesn't manage transactions and it is also mutable. Anyway, the intent is to use this implementation in a stateless session bean so there will be no thread safety issues generated by this class since the container will ensure thread safety.

JPADigitalRunTrackStatusDAO
This class implements DigitalRunTrackStatusDAO interface. It also extends the AbstractDAO class. This class manages TrackStatus entities in a JPA persistence (currently the JPA persistence will use Hibernate as a provider but any provider can be used). Each public method performs logging.
This class is not completely thread safe because it doesn't manage transactions and it is also mutable. Anyway, the intent is to use this implementation in a

stateless session bean so there will be no thread safety issues generated by this class since the container will ensure thread safety.

JPADigitalRunProjectTypeDAO
This class implements DigitalRunProjectTypeDAO interface. It also extends the AbstractDAO class. This class manages ProjectType entities in a JPA persistence (currently the JPA persistence will use Hibernate as a provider but any provider can be used). Each public method performs logging.
This class is not completely thread safe because it doesn't manage transactions and it is also mutable. Anyway, the intent is to use this implementation in a stateless session bean so there will be no thread safety issues generated by this class since the container will ensure thread safety.

JPADigitalRunPointsCalculatorDAO
This class implements DigitalRunProjectTypeDAO interface. It also extends the AbstractDAO class. This class manages ProjectType entities in a JPA persistence (currently the JPA persistence will use Hibernate as a provider but any provider can be used). Each public method performs logging.
This class is not completely thread safe because it doesn't manage transactions and it is also mutable. Anyway, the intent is to use this implementation in a stateless session bean so there will be no thread safety issues generated by this class since the container will ensure thread safety.

JPADigitalRunTrackDAO
This class implements DigitalRunTrackDAO interface. It also extends the AbstractDAO class. This class manages Track entities in a JPA persistence (currently the JPA persistence will use Hibernate as a provider but any provider can be used). It uses Search Builder component to search for DigitalRunPoints entities. Each public method performs logging.
This class is not completely thread safe because it doesn't manage transactions and it is also mutable. Anyway, the intent is to use this implementation in a stateless session bean so there will be no thread safety issues generated by this class since the container will ensure thread safety.

1.5 Component Exception Definitions

DigitalRunTrackManagerException

This is the base custom exception of all checked custom exceptions defined in this component. It is not thrown directly by any class.

EntityNotFoundException
This is a custom exception that will be thrown if an entity is not found in the persistence.

DigitalRunTrackManagerPersistenceException
This is a custom exception that will be thrown to indicate that errors occurred when accessing the persistence storage.

DigitalRunTrackManagerBeanConfigurationException
This is a custom exception that will be thrown to indicate that configuration related errors occurred (missing required property or errors that occur when getting the configuration from file).

1.6 Thread Safety

The component is not completely thread safe. The DigitalRunTrackManagerBean is thread safe because it is immutable and it will run in an ejb container which ensures thread safety. DigitalRunTrackFilterFactory is thread safe because it has no state. ConfigurationProvider is thread safe because its methods are synchronized. The dao implementations are not completely thread safe because they don't manage transactions. The dao implementations are intended to be plugged into the bean and in this case there are no thread safety issues; also, considering this usage, it is not necessary to make the dao implementations thread safe.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.5
- Compile Target: Java 1.5
- RedHat Linux 9
- JBoss 4.2 GA
- Informix 10.00.UC 5

2.2 TopCoder Software Components

- **Configuration API 1.0** - used for configuration purposes
- **Configuration Persistence 1.0** - used to get the configuration from file
- **Object Factory 2.0.1** - used to create objects
- **Object Factory Configuration API Plugin 1.0** - used with Object Factory component to create objects from ConfigurationObject.
- **Base Exception 2.0** - custom exceptions extend from BaseCriticalException and BaseRuntimeException
- **Search Builder 1.3.2** - used to perform searches with filters
- **Digital Run Entities 1.0** - entities are used from this component
- **Logging Wrapper 2.0** - used to perform logging

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.track
com.topcoder.service.digitalrun.track.manager.bean
com.topcoder.service.digitalrun.track.dao.implementations

3.2 Configuration Parameters

Configuration parameters for the bean:

Parameter Name	Parameter Description	Parameter Value
logName	Used to create a Log instance. If not present it means that logging is disabled. **Optional.**	non empty string
trackDAOKey	The key used with ObjectFactory to create a DigitalRunTrackDAO instance. **Required.**	non empty string
trackTypeDAOKey	The key used with ObjectFactory to create a DigitalRunTrackTypeDAO instance. **Required.**	non empty string
trackStatusDAOKey	The key used with ObjectFactory to create a DigitalRunTrackStatusDAO instance. **Required.**	non empty string
pointsCalculatorDAOKey	The key used with ObjectFactory to create a DigitalRunPointsCalculatorDAO instance. **Required.**	non empty string
projectTypeDAOKey	The key used with ObjectFactory to create a DigitalRunProjectTypeDAO instance. **Required.**	non empty string
unitName	The unit name used to retrieve an EntityManager instance. **Required.**	non empty string
activeStatusId	The active status id. **Required.**	long > 0

The configuration will also contain properties necessary to create a SearchBundle instance. It will have three properties :

Configuration parameters for the bean:

Parameter Name	Parameter Description	Parameter Value
name	The name of the SearchBundle instance.. **Required.**	non empty string
search_strategy_key	The key used with ObjectFactory to create a SearchStrategy instance. **Required.**	non empty string
context	The context. **Required.**	non empty string

It will also contain a child that contains the searchable fields of the SearchBundle. This child will be registered under “fields” name. This child contains attributes where the attribute name represents the field and the attribute value is a key used with ObjectFactory to create an ObjectValidator instance.

It will also contain a child that contains the column aliases. This child will be registered under “aliases” name. The attribute name is the column alias while the value is the column name.

3.3 Dependencies Configuration

None.

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 bean. The bean will be configured in ejb-jar.xml (see ejb-jar.xml from /docs directory)

4.3 Demo

1.3.1 Customer scenario that shows how to work with Track entities

```
//get the ejb
DigitalRunTrackManagerRemote remote = (DigitalRunTrackManagerRemote) new
     InitialContext().lookup(EAR_NAME +
```

```
            "/DigitalRunTrackManagerBean/remote");

        //create project type
        ProjectType projectType = this.createProjectType(10L);
        persist(projectType);

        //create pointsCalculator
        PointsCalculator pointsCalculator = createPointsCalculator(null);
        persist(pointsCalculator);

        //create trackStatus
        TrackStatus trackStatus = createTrackStatus();
        persist(trackStatus);

        //create trackType
        TrackType trackType = createTrackType();
        persist(trackType);

        //create track
        Track track = createTrack(pointsCalculator, trackStatus, trackType,
projectType);

        //create Track entity into persistence
        track = remote.createTrack(track);

        //the track entity has now an id; the start date is current date
        //modify start date to tomorrow date
        track.setStartDate(new Date(new Date().getTime() + (24 * 60 * 60 *
1000)));
        remote.updateTrack(track);

        //get a Track entity by specifying an id
        long id = track.getId();
        //the Track created above was returned
        remote.getTrack(id);

        //search for Track entities using a filter that returns entities //with
start date > current date
      DigitalRunTrackFilterFactory.createTrackStartDateGreaterOrEqualFilter(new
Date());

        //get active tracks
        List < Track > tracks2 = remote.getActiveTracks();

        //remove the created Track entity
        remote.removeTrack(id);
        delete(pointsCalculator);
        delete(trackStatus);
        delete(trackType);
        delete(projectType);
```

1.3.2 How to manage TrackType instances

```
        //create type
        TrackType type = createTrackType();

        //create track type
        type = remote.createTrackType(type);
```

```
        type.setCreationDate(new Date());
        type.setModificationDate(new Date());
        //update track type
        remote.updateTrackType(type);

        //get track type
        TrackType type1 = remote.getTrackType(type.getId());

        //get all track types
        List < TrackType > types = remote.getAllTrackTypes();

        //delete track type
             remote.removeTrackType(type.getId());
```

1.3.3 How to manage TrackStatus instances

```
        TrackStatus status = createTrackStatus();
        //create track status
        status = remote.createTrackStatus(status);
        status.setCreationDate(new Date());
        status.setModificationDate(new Date());
        //update status
        remote.updateTrackStatus(status);

        //get track status
        TrackStatus status1 =
remote.getTrackStatus(status.getId());

        //get all track statuses
        List < TrackStatus > statuses =
remote.getAllTrackStatuses();

        //delete track status
         remote.removeTrackStatus(status.getId());
```

1.3.4 How to manage PointsCalculator instances

```
        PointsCalculator pc = createPointsCalculator(null);
        //create points calculator
        pc = remote.createPointsCalculator(pc);

        pc.setCreationDate(new Date());
        pc.setModificationDate(new Date());
        //update points calculator
        remote.updatePointsCalculator(pc);

        //get points calculator
        PointsCalculator pc1 =
remote.getPointsCalculator(pc.getId());

        //get all points calculators
        List < PointsCalculator > pc2 =
remote.getAllPointsCalculators();
```

```
//delete points calculator
 remote.removePointsCalculator(pc.getId());
```

1.3.5 How to manage ProjectType instances

```
ProjectType pt = createProjectType(10L);
 persist(pt);
 //get project type
 pt = remote.getProjectType(10L);

 //get all project types
 List < ProjectType > pt1 = remote.getAllProjectTypes();

delete(getEntityManager().find(ProjectType.class,
pt.getId()));
```

5. Future Enhancements

Auditing can be added in the future for the create/update/remove methods.