# Time Tracker Rates 3.2 Requirements Specification

## 1. Scope

### 1.1 Overview

The Time Tracker Rates custom component is part of the Time Tracker application. It provides an abstraction of rates, which are configurable in the Time Tracker System. This component handles the persistence and other business logic required by the application.

The design for this specification exists, but requires modification. The text in RED is new requirements. You are to make the additions to the existing design.

### 1.2 Logic Requirements

#### *1.2.1 Company Account*

A company has a relation to everything in the context of the application, such as rates, users, clients, projects, and entries. This relation provides a separation of data by company such that many separate companies can use the same Time Tracker application with a logical separation of data.

#### *1.2.2 Rate*

##### 1.2.2.1 Overview

A Rate is a configurable entry in the Time Tracker System. For example the amount per mile that an auto mileage expense is reimbursed is a Rate. The Rate itself is the configurable item, but each rate has an independent value per company. This entity object Rate extends TimeTrackerBean and models the following expense entry information:

- Rate ID – the unique Rate ID number (TimeTrackerBean id field)
- Rate – the value of the rate. Currently rates are stored up to 3 decimal points of precission
- Description – a unique name which identifies the Rate

##### 1.2.2.2 Database Schema

The Rate information will be stored in the following tables (refer to TimeTrackerRate_ERD.jpg):

- rate
- comp_rate

##### 1.2.2.3 Required Operations

- Create a new Rate
- Retrieve an existing Rate by rateId and companyId
- Retrieve an existing Rate by rate description and companyId
- Update an existing Rate
- Delete an existing Rate
- Enumerate all existing Rates
- Batch versions of the CRUD operations

##### 1.2.2.4 Audit Requirements

The Time Tracker Audit component will encapsulate the actual auditing of the data. Note that the audit information should not exist for a transaction, which rolled back. If you have a transaction that fails and you have already submitted audit information make sure to remove the audit

information. All Create, Update and Deletes actions will be audited

The Audit component required the consumer to identify the application area that the audit is for. The application area for the Rate will be TT_CONFIGURATION.

### *1.2.3 Pluggable Persistence*

All entities defined in previous sections will be backed by a database. The design will follow the DAO pattern to store, retrieve, and search data from the database. All ID numbers will be generated automatically using the ID Generator component when a new entity is created. All creation and modification dates will be taken as the current datetime.

For this version, the Informix database system will be used as persistence storage for this component and the Time Tracker application. Other database systems should be pluggable into the framework.

### *1.2.4 JavaBeans Conventions*

For all the entities described in previous sections, the JavaBeans conventions will be followed (http://java.sun.com/products/javabeans/docs/spec.html):

- The class is serializable
- The class has a no-argument constructor
- The class properties are accessed through `get`, `set`, `is` methods. i.e. All properties will have get<PropertyName>() and set<PropertyName>(). Boolean properties will have the additional is<PropertyName>().

Note: Event-handling methods are not required.

### *1.2.5 Transaction Management*

As a result of the fine grain components used in this design there needs to be a transaction management strategy, which allows a single transaction to exist that encompasses all components called for a single use case. Since this component will be deployed into an Enterprise Java Bean container, JBoss 4.0.x, a Stateless Session Bean will be used to manage the transaction. The container will start a transaction when a method is invoked if one is not already running. The method will then join the new or existing transaction. Transaction Management will be Container Managed.

#### 1.2.5.1 User API for component

The user API for this component will exist in a Delegate object. This delegate will provide the contract for the component and interface with the EJB. The Delegate is not an EJB rather it will be a POJO. It will look up the EJB and call the related method, retrieve the results and return the results to the consumer. There will be no additional logic in the delegate.

#### 1.2.5.2 Stateless Session Bean

The methods on the Stateless Session bean will have a transaction level of REQUIRED in the deployment descriptor. This will allow for either a new transaction to be created or for the method to join the existing transaction. For this release we will use a Local Bean and not a Remote Bean. There are a few obstacles, which will need to be addressed:

- No File IO from with in the EJB so ConfigurationManager cannot use a file. Values can however be stored in the Deployment Descriptor.

- All parameters passed to/from the Session bean must be Serializable, however the Filter Object in Search Builder 1.3.1 is not Serializable. This is being addressed and will be fixed.
- The Session Bean should not have any class level variables to store things like the DAO. If it does have a class level variable it must be transient, therefore after activation it will have a value of null. Any of the approaches outlined below are acceptable:
  - Have a class level dao attribute and only access it via a getDAO() method which checks for null and sets the dao attribute if it is null.
  - Like the first method have a class level attribute but on creation or activation load the DAO to the class dao attribute. You must then ensure that under all scenarios that the attribute will be not null.
  - Use a singleton to act as a DAO cache
  - There may be others, and you are not limited to one of these.
- No threads can be created with in the EJB.
- Review the Sun J2EE specification for any other limitations.

All Business logic for the componet will reside in the Stateless Session Bean. There will be no logic in the delegate or in the DAO. There is one exception to this, in that the Audit functionality will exist in the DAO.

#### 1.2.5.3 DAO

The DAO's must retrieve the connection that it uses from the configured TXDatasource in JBoss. The configuration of the DataSource should be externalized so that is can be configured at deployment time.

All audit functionality will exist in the DAO.

### 1.3 Required Algorithms

None.

### 1.4 Example of the Software Usage

The Time tracker Application has many configurable parameters. One of which are the Rates. Currently the only configurable rate is the auto mileage expense rate. This is the rate per mile that can be expensed.

### 1.5 Future Component Direction

Additional Rates will be added.

## 2. Interface Requirements

#### *2.1.1 Graphical User Interface Requirement*

None.

#### *2.1.2 External Interfaces*

- Time Tracker Common
  - TimeTrackerBean
- Time Tracker Audit
  - AuditManager

- AuditHeader
- AuditDetail

*2.1.3 Environment Requirements*

- Development language: Java 1.4
- Compile target: Java 1.4, Java 1.5

*2.1.4 Package Structure*

com.topcoder.timetracker.rate

## 3. Software Requirements

### 3.1 Administration Requirements

*3.1.1 What elements of the application need to be configurable?*

None.

### 3.2 Technical Constraints

*3.1.1 Are there particular frameworks or standards that are required?*

- JavaBeans (http://java.sun.com/products/javabeans/docs/spec.html)

*3.2.2 TopCoder Software Component Dependencies:*

- Configuration Manager
- DB Connection Factory
- ID Generator
- Search Builder
- Time Tracker Audit
- Time Tracker Common

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

*3.2.3 Third Party Component, Library, or Product Dependencies:*

Informix Database.

*3.2.4 QA Environment:*

- JBoss 4.0
- Windows 2000
- Windows Server 2003
- Informix

### 3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. No use of Store procedures or triggers should exist.

### 3.4 Required Documentation

#### *3.4.1 Design Documentation*

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

#### *3.4.2 Help / User Documentation*

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of Poseidon.