Base Entry 3.2 Requirements Specification

1. Scope

1.1 Overview

The Base Entry custom component is part of the Time Tracker application. It provides an abstraction of the high level entry that an employee enters into the system. It is an abstract entity, which is extended by Time Entry, Expense Entry and Fixed Billing entry. This component handles the common entry business logic required by the application.

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

1.2 Logic Requirements

1.2.1 Company Account

A company owns everything in the context of the application, such as users, clients, projects, and time entries. The following sections will describe how the company entity relates to the existing Base Entry entities in details.

1.2.2 Time Entry

1.2.2.1 Overview

A base entry is associated with a company ID. This component will model the following entry information:

- Company ID – the company Id associated with the entry
- Description – a brief description of the entry
- Entry Date – the date for the entry
- Reject Reasons – the reasons why the entry was rejected.

Provided by parent class TimeTrackerBean

- Entry ID – the unique entry ID number
- Creation Date – the date the entry was created
- Creation User – the username that created the entry
- Modification Date – the date the entry was modified
- Modification User – the username that modified the entry

1.2.2.2 Entry Submission

In order to determine if an entry can be submitted it must be submitted before the cut off day of the week and time. Once this time is reached on the day of the week assigned as the cut off day, entries can no longer be entered for the previous week. The work week runs from Sunday to Saturday. Each company should be able to configure their own cut off day and time.

Here is the algorithm for determining the entry submission:

Compare the entry date to the Saturday 11:59pm before the cut off, as the work week default is Sunday to Saturday, and the **current day to the cut off**. This means, using an example of Monday 09:30 (9:30 am on a 24 hour clock) as the cut off, an entry prior to the Sunday Midnight would be allowed to be submitted. Valid entries would extend back 7 days prior to the Sunday Midnight. This example should help.

Using the date structure:
S1 M1 T1 W1 Th1 F1 Sa1 S2 M2 T2 W2 Th2 F2 Sa2 S3 M3 T3 W3 Th3 F3 Sa3 S4
Where the first letter represents the Day of the week and the number represents the week. We will use server time, in this case Eastern.

If an entry had a date of T2 and was submitted on S3 it would pass. If the same entry was submitted on Th3 it would fail. The entry on the same date, T2, should also be able to be submitted any day up to M3 at 09:30.

An entry with an entry date W1 submitted on M3 would fail. Another scenario which will fail would be an Entry on the date of T2 being submitted before T2.

At this point the work week is not configurable. I presume we will make it configurable in the next version.

1.2.2.3 Database Schema

The Cut Off Time information will be stored in the following tables (refer to TimeTrackerBaseEntry_ERD.jpg):

- Cutt_off_time

1.2.2.4 Required Operations

- Update the cut off time
- Retrieve the cut off time
- Can an entry be submitted

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 will use this component to perform operations related to time entries.

1.5 Future Component Direction

Other database systems maybe plugged in for some client environments.

2. Interface Requirements

2.1.1 Graphical User Interface Requirement

None.

2.1.2 External Interfaces

None.

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.entry.base

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

None.

3.2 Technical Constraints

3.2.1 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
- Reject Reason
- 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.x
- 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.

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.