

Time Entry 3.2 Requirements Specification

1. Scope

1.1 Overview

The Time Entry custom component is part of the Time Tracker application. It provides an abstraction of a time entry that an employee enters into the system on a regular basis. 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 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 Time Entry

1.2.2.1 Overview

A time entry represents the date and amount of time an employee has worked for a particular project and client. It is normally used for payroll and billing purposes. This entity object TimeEntry extends Base Entry models the following time entry information:

- Hours – the amount of hours the employee worked
- Time Entry Id – the unique time entry id (TimeTrackerBean id field)
- Task Type – the task type of time entry
- Status – the status of the time entry
- Billable – a flag to indicate whether the entry is billable to client (0 for false, 1 for true)
- Invoice – this is the invoice that the entry is associated with. This may be null if it has not yet been invoiced.

1.2.2.2 Search Filters

This component will provide search functionalities based on a logical (AND, OR, NOT) combination of search filters. The following is a summary of the required filters:

- Return all entries with the given company ID
- Return all entries with the given Invoice ID
- Return all entries with description that contains a given string
- Return all entries with entry date within a given inclusive date range (may be open-ended)
- Return all entries with amount within a given inclusive amount range (may be open-ended)
- Return all entries with a given task type
- Return all entries with a given time status
- Return all entries with a given billable flag
- Return all entries with a given reject reason ID
- Return all entries created within a given inclusive date range (may be open-ended)
- Return all entries modified within a given inclusive date range (may be open-ended)
- Return all entries created by a given username

- Return all entries modified by a given username

1.2.2.3 Database Schema

The time entry information will be stored in the following tables (refer to TimeTrackerTime_ERD.jpg):

- time_entry
- time_reject_reason

1.2.2.4 Required Operations

- Create a new time entry
- Retrieve an existing time entry by ID
- Update an existing time entry information
- Delete an existing time entry
- Enumerate all existing time entries
- Batch versions of the CRUD operations
- Search time entries by filters
- Get/Set task type of an existing time entry (company ID must match)
- Get/Set time status of an existing time entry
- Link reject reason IDs to an existing time entry (company ID must match)
- Unlink reject reason IDs from an existing time entry
- Unlink all reject reason IDs (if any) from an existing time entry
- Get all linked reject reason IDs for an existing time entry

1.2.2.5 Audit Requirements

Each method, which is capable of modification of data, is required to allow the consumer to optionally require auditing. This allows the consumer to determine if the change of data will be audited or not. 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.

The Audit component required the consumer to identify the application area that the audit is for. The application area for the Time Entry will be TT_TIME.

1.2.3 Task Types

1.2.3.1 Overview

Each time entry has an associated type of task. For the new version, the type will be associated with a company ID. This entity object TaskType extends the TimeTrackerBean and models the following task type information:

- Company ID – the company ID associated with the task type
- Task Type ID – the unique task type ID number (TimeTrackerBean id field)
- Description – a brief description of the task type
- Active – a flag that indicates whether the task type is active (0 for false, 1 for true)

1.2.3.2 Search Filters

This component will provide search functionalities based on a logical (AND, OR, NOT)

combination of search filters. The following is a summary of the required filters:

- Return all types with a given company ID
- Return all types with description that contains a given string
- Return all types with a given active flag condition
- Return all types created within a given inclusive date range (may be open-ended)
- Return all types modified within a given inclusive date range (may be open-ended)
- Return all types created by a given username
- Return all types modified by a given username

1.2.3.3 Database Schema

The task type information will be stored in the following tables (refer to TimeTrackerTime_ERD.jpg):

- task_type
- comp_task_type

1.2.3.4 Required Operations

- Create a new task type
- Retrieve an existing task type by ID
- Update an existing task type information
- Delete an existing task type
- Enumerate all existing task types
- Search task types by filters

1.2.4 Time Status

1.2.4.1 Overview

Each time entry has an assigned status. The entry status will change over the course of the application lifetime. This entity object TimeStatus extends TimetrackerBean and models the following Time status information:

- Time Status ID – the unique time status ID number (TimeTrackerBean id field)
- Description – a brief description of the time status

1.2.4.2 Search Filters

This component will provide search functionalities based on a logical (AND, OR, NOT) combination of search filters. The following is a summary of the required filters:

- Return all statuses with description that contains a given string
- Return all statuses created within a given inclusive date range (may be open-ended)
- Return all statuses modified within a given inclusive date range (may be open-ended)
- Return all statuses created by a given username
- Return all statuses modified by a given username

1.2.4.3 Database Schema

The Time status information will be stored in the following tables (refer to TimeTrackerTime_ERD.jpg):

- time_status

1.2.4.4 Required Operations

- Create a new time status
- Retrieve an existing time status by ID
- Update an existing time status information
- Delete an existing time status
- Enumerate all existing time statuses
- Search time statuses by filters

1.2.5 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.

1.2.6 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.

1.2.7 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.

1.2.7.1 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.

1.2.7.2 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 within 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 within the EJB.
- Review the Sun J2EE specification for any other limitations.

All Business logic for the component 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.

1.2.7.3 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 it 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

- Time Tracker Common
 - TimeTrackerBean
- Time Tracker Audit
 - AuditManager
 - AuditHeader
 - AuditDetail
- Time Tracker Base Entry
 - BaseEntry
- Time Tracker Reject Reason
 - RejectReasonManager
 - RejectReason

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.time

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
- Time Tracker Audit
- Time Tracker Reject Reason
- Time Tracker Common
- Time Tracker Base Entry

**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.