

Expense Entry 2.0 Requirements Specification

1. Scope

1.1 Overview

The Expense Entry custom component is part of the Time Tracker application. It provides an abstraction of an expense 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.

Version 2.0 will leverage the version 1.1 design by modifying existing and adding new classes and methods to meet the new requirements.

Note: existing table names have changed, but their meanings have not.

1.2 Logic Requirements

1.2.1 Company Account

The new version of the Time Tracker application introduces a new entity called company. A company owns everything in the context of the application, such as users, clients, projects, and expense entries. The new design will retrofit the existing classes and methods to accommodate this new entity. The following sections will describe how the company entity relates to the existing Expense Entry entities in details.

1.2.2 Expense Entry

1.2.2.1 Overview

An expense entry represents the date and amount of money an employee has spent for a particular project and client. It is normally used for payroll and billing purposes. Expense entries will not be associated with a company ID. This component will model the following expense entry information:

- Company ID – the company ID associated with the expense entry
- Entry ID – the unique expense entry ID number
- Description – a brief description of the expense entry
- Entry Date – the date for the expense entry
- Amount – the amount of money the employee spent
- Expense Type – the type of expense
- Status – the status of the expense entry
- Billable – a flag to indicate whether the entry is billable to client (0 for false, 1 for true)
- Creation Date – the date the expense entry was created
- Creation User – the username that created the expense entry
- Modification Date – the date the expense entry was modified
- Modification User – the username that modified the expense entry

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 a given company 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 expense type
- Return all entries with a given expense 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 expense entry information will be stored in the following tables (refer to Expense_Entry.sql):

- expense_entry
- exp_reject_reason

1.2.2.4 Required Operations

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

1.2.3 Expense Types

1.2.3.1 Overview

Each expense entry has an associated type of expense. For the new version, the type will be associated with a company ID. This component will model the following expense type information:

- Company ID – the company ID associated with the expense type
- Expense Type ID – the unique expense type ID number
- Description – a brief description of the expense type
- Active – a flag to indicate whether the expense type is active (0 for false, 1 for true)
- Creation Date – the date the expense type was created
- Creation User – the username that created the expense type
- Modification Date – the date the expense type was modified
- Modification User – the username that modified the expense type

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 expense type information will be stored in the following tables (refer to Expense_Entry.sql):

- expense_type
- comp_exp_type

1.2.3.4 Required Operations

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

1.2.4 Expense Status

1.2.4.1 Overview

Each expense entry has an assigned status. The entry status will change over the course of the application lifetime. This component will model the following expense status information:

- Expense Status ID – the unique expense status ID number
- Description – a brief description of the expense type
- Creation Date – the date the expense status was created
- Creation User – the username that created the expense status
- Modification Date – the date the expense status was modified
- Modification User – the username that modified the expense 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 expense status information will be stored in the following tables (refer to Expense_Entry.sql):

- expense_status

1.2.4.4 Required Operations

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

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

The Time Tracker application will use this component to perform operations related to expense 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.cronos.timetracker.entry.expense

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

****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:

- Solaris 7
- RedHat Linux 7.1
- 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. Modifications to these guidelines for this component should be detailed below.

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.