Time Tracker Reject Reason 3.2 Requirements Specification

1. Scope

1.1 Overview

The Time Tracker Reject Reason custom component is part of the Time Tracker application. It provides an abstraction of reject reasons for many of the other components in the 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 Reject Reason

1.2.1.1 Overview

A project manager can reject time, expense and fixed billing entries. The manager will select one or more reasons from a drop down list of enumerations. Each company will maintain its own set reject reasons stored in the database lookup table. The component will model the following information for reject reason:

- Company ID – the company ID associated with the reject reason
- Reason ID – the unique reject reason ID number
- Description – free-form text description of the reject reason
- Creation Date – the date the reject reason was created
- Creation User – the username that created the reject reason
- Modification Date – the date the reject reason was modified
- Modification User – the username that modified the reject reason

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

1.2.1.3 Database Schema

The reject reason information will be stored in the following tables (refer to TimeTrackerRejectReason_ERD.jpg):

- reject_reason
- comp_rej_reason

1.2.1.4 Required Operations

- Create a new reject reason
- Retrieve and existing reject reason by ID
- Update an existing reject reason information
- Delete an existing reject reason
- Enumerate all existing reject reasons
- Search reject reasons by filters
-

1.2.1.5 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. 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 Reject Reason will be TT_CONFIGURATION.

1.2.2 Reject Email

1.2.2.1 Overview

When a time or expense entry is rejected by the project manager, a notification email will be sent to the contractor. The email will be generated from a template tailored for the company. The email template will be stored in the database. The component will model the following information for reject email:

- Company ID – the company ID associated with the reject email
- Email ID – the unique reject email ID number
- Body – the email template body
- Creation Date – the date the reject email was created
- Creation User – the username that created the reject email
- Modification Date – the date the reject email was modified
- Modification User – the username that modified the reject email

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

1.2.2.3 Database Schema

The reject email information will be stored in the following tables (refer to TimeTrackerRejectReason_ERD.jpg):

- reject_email
- comp_reject_email

1.2.2.4 Required Operations

- Create a new reject email
- Retrieve an existing reject email by ID
- Update an existing reject email information
- Delete an existing reject email
- Enumerate all existing reject emails
- Search reject emails by filters
-

1.2.2.5 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. 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 Reject Email 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 will use this component to perform operations related to company and user authentication and authorization.

1.5 Future Component Direction

Other database systems maybe plugged in for some client environments. Multiple user stores

may be used for the same client environment.

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.rejectreason

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
- Encryption
- Search Builder
- Authentication Factory
- Authorization
- 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. 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.