Service Detail 1.0 Requirements Specification

1. Scope

1.1 Overview

The Service Detail custom component is part of the Time Tracker application and provides additional detail to the invoice for Time Entries. This component handles the persistence and other business logic required by the application.

1.2 Logic Requirements

1.2.1 Service Detail

1.2.1.1 Overview

The Service Detail encapsulates the data for both Time and rate of the resource. Once a Time entry is associated with an Invoice the Rate is stored with it. We use a Service Detail to create a composite view of the two pieces of data. This allows the service detail rate to be edited apart from original project_workers rate entry. ProjectWorker exists in the Project Component and defines system users as workers on a project. The entity object ServiceDetail extends TimeTrackerBean and models the following information:

- Service Detail ID – the unique Service Detail Id number (TimeTrackerBean id field)
- Invoice Id – the parent invoice that this record is assigned to
- Time Entry Id – the id of the associated Time Entry
- Rate – The resources Rate if this is a time entry
- Amount - the amount of the hours times the rate.
-

1.2.1.2 Database Schema

The service detail information will be stored in the following tables (refer to TimeTrackerInvoice_ERD.jpg):

- service_detail

1.2.1.3 Required Operations

- Create a new Service Detail
- Retrieve an existing Service Detail by ID
- Update an existing Service Detail information
- Delete an existing Service Detail
- Batch versions of the CRUD operations
- Enumerate all existing Service Detail for a given Invoice

1.2.1.4 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 Service detail will be TT_INVOICE. Modifications to the Time Entry should be done using the Time Tracker Time, with the audit function turned on.

1.2.2 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.

1.2.3 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.

1.2.3.1 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.

1.2.3.2 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.

1.2.3.3 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 Invoice processing. Invoices will be created after the Projects Manager approves all Time, Expense and Fixed Billing entries. Once the Invoice has been created the Invoice will be managed through a short workflow, which allows account the ability to modify the invoice and purchase order number as well as indicate when the invoice has been paid.

1.5 Future Component Direction

Invoice reporting, aging of invoices and additional steps I the workflow process will likely evolve.

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 Time Entry
  - TimeManager
  - TimeEntry
- Time Tracker Project
  - Project Manager
  - Project
  - ProejctWorker

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.invoice.servicedetail

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 Time Entry
- Time Tracker Expense Entry
- Time Tracker Fixed Billing Entry
- Time Tracker Common
- Time Tracker Audit
- Time Tracker Project

**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.