Time Tracker Contact 3.2 Requirements Specification

1. Scope

1.1 Overview

The Time Tracker Contact custom component is part of the Time Tracker application. It provides an abstraction of contacts and addresses. Contacts and Addresses exist in Time Tracker as a many to any relationship to many other entities. 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. The sections are 1.2.2.4 and 1.2.5.

1.2 Logic Requirements

1.2.1 Contact

1.2.1.1 Overview

The Contact in formation is stored as a one to any relationship. This means that one contact can have a relation to any entity, which is configured to build relationships to it. For example in Time Tracker Company, Client and Project all can have relationship with contact. One contact can be assigned to a Client and that same contact can be assigned to a Project. Any changes made to the project contact will affect the Client contact automatically. This component extends TimeTrackerBean and models the following contact information:

- Contact ID – the unique contact ID number
- Contact Name
- Phone
- Email
- Contact type – the type of contact this is. This will determine what type of entity it is associated with. For example Company, Project, Client or User

The persistence model will record the id of the entity with which this contact is to be related in contact_relationships in the entity_id column. So for example if the Client has a contact then the Clients Id will be stored I the entity_id column and ContactType will be CLIENT.

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 contacts with name that contains a given string
- Return all contacts created within a given inclusive date range (may be open-ended)
- Return all contacts modified within a given inclusive date range (may be open-ended)
- Return all contacts created by a given username
- Return all contacts modified by a given username
- Return all contacts with a given email address
- Return all contacts with a given phone
- Return all contacts with a given contact type
- Return all contacts with a given entity id, the id of the parent record. If this provided then contact type is also required

1.2.1.3 Database Schema

The client information will be stored in the following tables (refer to TimeTrackerContact_ERD.jpg):

- contact
- contact_type
- contact_relationships

1.2.1.4 Required Operations

- Create a new contact
- Retrieve an existing contact by ID
- Update an existing contact information
- Delete an existing contact
- Enumerate all existing contact
- Batch versions of the CRUD operations
- Search contact by filters

1.2.1.5 Audit Requirements

The component is required to allow for a boolean on any method, which allows for the modification of the data, to determine is the action is to be audited. The Time Tracker Audit component will encapsulate the actual auditing of the data. Note that the audit should be in the transaction and rolled back if the transaction fails.

The application area for this will be TT_PROJECT, TT_CLIENT, TT_COMPANY or TT_USER depending on the Contact Type.

1.2.2 Address

1.2.2.1 Overview

The Address in formation is stored as a one to any relationship. This means that one address can have a relation to any entity, which is configured to build relationships to it. For example in Time Tracker Company, Client and Project all can have relationship with address. One address can be assigned to a Client and that same address can be assigned to a Project. Any changes made to the project address will affect the Client address automatically. This component extends TimeTrackerBean and models the following address information:

- Address Id – the unique id for this address
- Address Line 1
- Address Line 2
- City
- State
- Zip code
- Country
- Address type – the type of address this is. This will determine what type of entity it is associated with. For example Company, Project, Client or User

The persistence model will record the id of the entity with which this address is to be related in address _relationships in the entity_id column. So for example if the Client has a address then the Clients Id will be stored I the entity_id column and Address Type will be CLIENT.

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 addresses created within a given inclusive date range (may be open-ended)
- Return all addresses modified within a given inclusive date range (may be open-ended)
- Return all addresses created by a given username
- Return all addresses modified by a given username
- Return all addresses with a given City
- Return all addresses with a given State
- Return all addresses with a given Zip Code
- Return all addresses with a given Country
- Return all addresses with a given address type
- Return all addresses with a given entity id, the id of the parent record. If this provided then contact type is also required

1.2.2.3 Database Schema

The client information will be stored in the following tables (refer to TimeTrackerContact_ERD.jpg):

- address
- address_type
- address_relationships

1.2.2.4 Required Operations

- Create a new address
- Retrieve an existing address by ID
- Update an existing address information
- Delete an existing address
- Enumerate all existing address
- Batch versions of the CRUD operations
- Search address by filters
- Enumerate States
- Enumerate Countries

1.2.2.5 Audit Requirements

The component is required to allow for a boolean on any method, which allows for the modification of the data, to determine is the action is to be audited. The Time Tracker Audit component will encapsulate the actual auditing of the data. Note that the audit should be in the transaction and rolled back if the transaction fails.

The application area for this will be TT_PROJECT, TT_CLIENT, TT_COMPANY or TT_USER depending on the Address Type.

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 client and project management.

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.contact

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 Common
- Time Tracker Audit

**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.