Time Tracer Notification 3.2 Requirements Specification Scope

1.1 Overview

The Notification custom component is part of the Time Tracker application. It provides an abstraction of the notification features used to communication via email to users of the system. This component integrates the scheduling, sending and manages 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 notifications, 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 Notification

1.2.2.1 Overview

A Notification contains the elements of an email message, which is to be sent out according to a schedule. The Notification dictates what will be sent out and to whom. This component entity object Notification extends TimeTrackerBean and models the following expense entry information:

- Notification Id – the unique notification Id number (TimeTrackerBean id field)
- Company Id – the Company that this notification belongs to
- Subject – The text which will be placed in the subject of the email
- Message – the text of the main body of the email
- From Address – Whom the email will appear to be from. This is a free form text in application but it is required to be in the proper format.
- ToClients – a list of clients to send the email to. Only clients with email addresses at the time of the sending will actually receive the notification. The clients contact name and email address will be used.
- ToProjects – a list of projects to send the email to. Only projects with email addresses at the time of the sending will actually receive the notification. The projects contact name and email address will be used.
- ToResources - a list of resources to send the email to. Only resources with email addresses at the time of the sending will actually receive the notification. The users contact name and email will be used.
- Status – determines if the Notification is active or not. Only Active notifications are processed.
- LastTimeSent – the last time that the notification was sent
- NextTimeToSend – the approximate time that the notification will be sent again
- Schedule Id – the id of the schedule entry which represents the recurrence pattern provided by the user for this notification. Only one Notification can be associated with a given Schedule Id.

All email address will be in the format of "Name < email@domain.com >". This applies to the to and from elements.

1.2.2.2 Search Filters

This component will provide search functionalities based on a logical (AND, OR, NOT) combination of search filters. The return will be a collection of Notification objects. If no results are found then the result set will be empty. An exception will be thrown in the event of an error condition being raised. The following is a summary of the required filters:

- Return all entries with a given company Id
- Return all entries with a given project Id
- Return all entries with a given client Id
- Return all entries with a given resource Id
- Return all entries with a given status
- Return all entries with last sent within a given inclusive date range (may be open-ended)
- Return all entries with next send within a given inclusive date range (may be open-ended)
- Return all entries with Subject line containing some text
- Return all entries with Message line containing some text
- Return all entries with from line containing some text
- 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 Notification information will be stored in the following tables (refer to TimeTrackerNotification_ERD.jpg):

- notification
- notify_resources
- notify_clients
- notify_projects

1.2.2.4 Required Operations

- Create a new Notification
- Retrieve an existing Notification by ID
- Retrieve an existing Notification by schedule Id
- Update existing Notification
- Delete a notification by Id
- Enumerate all existing Notification
- Search Notification by filter.
- Send Notification Immediately – will look up the notification and use the SendNotificationUtilty to process the Notification

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 Notification or Recurrence Pattern will be TT_NOTIFICATION.

1.2.3 Recurrence Pattern

1.2.3.1 Overview

The Recurrence Pattern defines when and how often to send a notification. This pattern will be stored as part of a Schedule that defines when to send notifications out. The Recurrence Pattern will interact with the new java Job Scheduling component version 3.0 in order to manage scheduling.

The entities, which are used to represent the recurrence pattern, will be those of the Job Scheduling component.

1.2.3.2 Database Schema

Java Scheduling component has its own schema for the persistence of schedules in a DB. This schema will be used.

1.2.3.3 Arguments Passed to Job Scheduler

When creating a Job the following fields will contain certain data:

- Job.name = TT_NOTE_[Notification Id], so if the notification Id was 56, then the job name would be TT_NOTE_56.
- Rob.runCommand = com.topcoder.timetracker.notification.send.NotificationEvent
- The consumer will set the rest of the Job data, as they will be required to pass in a Job object as the Recurrence Pattern.

NOTE: because of the circular reference between Notification and Job, you will need to save the notification with a scheduleId = 0 (zero), then add the Job through Job Scheduler. Using the getjob(jobName) grab the job and get the Id, setting in the notification and updating the notification.

1.2.4 Send Notification utility

1.2.4.1 Overview

This utility class processes the Notification and forms the mail to be sent. It will process the notification and send it to the email addresses.

1.2.4.2 Logging

In the event of a failure to send a notification an entry in the log should indicate the notification Id, date and time and the reason for failure. If individual email addresses were involved list them as well.

1.2.5 Automatic Sending of Notifications

1.2.5.1 Overview

Based on the recurrence pattern, there needs to be an automated mechanism to process the schedule and send emails. This can be accomplished using the Job Processor, which uses the Job Scheduler to read an existing schedule.

1.2.5.2 Notification Event

The NotificationEvent implements the ScheduledEnable interface of Job Scheduling. This is a generic event which is registered with each scheduled entry. This event will take the supplied JobId and call the retrieveNotificationWithScheduleId(jobId) on NotificationManager to obtain the relevant Notification. Then the SendNotificationUtility will be called in order to process the Notification.

1.2.6 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 will be pluggable into the framework.

1.2.7 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.

1.2.8 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.

1.2.8.1 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.

1.2.8.2 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.

1.2.8.3 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 Notifications. A user will be able to define a notification, which will be assigned a recurrence Pattern. This notification will be scheduled to go out to the email address of clients, projects or system users.

1.5 Future Component Direction

Other database systems may be plugged in for some other client environments.

2. Interface Requirements

2.1.1 Graphical User Interface Requirement

None.

2.1.2 External Interfaces

- Time Tracker Common
  - TimetrackerBean
- Job Scheduling
  - Scheduler
  - Job
  - ScheduledEnable
- Time Tracker Audit
  - AuditManager
  - AuditHeader
  - AuditDetail

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.notification

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
- Email Address Validator
- Email Engine
- Search Builder
- Job Scheduling V 3.0
- Job Processor
- 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. 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.