Time Tracker Time Entry version 3.2 Component Specification

1.Design

The Time Tracker Time Entry custom component is part of the Time Tracker application. It provides an abstraction of Time Entries. This component handles the persistence and other business logic required by the application.

The design is separated into 2 layers of management: The topmost layer is the Manager classes, which provide the functionality needed to update, retrieve and search from the data store. Batch operations are also available in the Manager classes. The next layer is composed of the DAOs, which interact directly with the data store. There is also a sublayer, called the FilterFactory layer, which is responsible for building search filters which can be used to search the filter.

An additional layer is built on top of the first two layers. This layer is the J2EE layer, which consists of a Local, LocalHome, Delegate, and SessionBean classes for each of the different entities. The Local and LocalHome interfaces define the methods needed by the J2EE container to create the implementing classes. The Delegate, Local and SessionBean classes all implement/extend from their respective Manager interface, and therefore have the same set of business methods. The Delegate looks up the Local interface and routes all method calls to the Local interface. The Local interface then routes those calls to the SessionBean. The SessionBean finally routes those calls to the Manager implementation, where the actual business logic resides.

The final version of 3.1 included most of the functionality for the J2EE layer. The design makes some slight modifications and improves upon the documentation provided by this final version:

- A ConfigurationException class was added to handle problems with configuration. This is relevant for the Delegate classes, as well as another newly introduced ManagerFactory class.
- No more state is maintained within the SessionBeans. The SessionBean 'looks up' the relevant Manager using a ManagerFactory. The ManagerFactory acts like a singleton, and functions as a global access point for the Managers.
- A sample Deployment Descriptor is provided in the docs directory.
- A new Sequence Diagram was added to depict the J2EE functionality.
- A note was added to the Use Cases, saying that the application may now run under a J2EE container. Since the requirements for the new changes are not actual business functionality, but rather more on the inner functionality of the component, it did not seem appropriate to introduce a new use case for the new requirements.
- There was no need to refactor the DAOs, since no business logic was found in the DAOs. The JBoss Transaction DataSource may be configured using ConnectionFactory.
- The design attempts to adhere to the J2EE specification of not allowing File access by delegating all file access to an external ManagerFactory class. This is a pragmatic approach which has been proven to work in previous designs (see the Orpheus Application components, like Game Persistence, Administration Persistence components). File access does not occur within the SessionBean itself, but rather within external classes when the application is called. It is also possible for File Access to occur before any SessionBean calls occur, since the ConfigManager may be initialized beforehand.
- Note that the only other alternative to not using ConfigManager and configuration

files will be to restrict usage of ANY TC components that utilize configuration files - these include the Connection Factory, Search Builder and ID Generator components, which are a core part of the given component.

1.1Design Patterns

The Strategy Pattern. *The DAO classes such as TimeEntryDAO, TimeStatusDAO, and TaskTypeDAO are strategies for persisting and retrieving information from a data store.*

The Facade Pattern. *The manager classes [TimeEntryManager, TimeStatusManager, TaskTypeManager] encapsulate subsystem DAOs and other components to provide a unified API that makes it easier to manage the Time Tracker Entries.*

The Factory Pattern. *The Factory pattern is used in the FilterFactory layer to create search filters. The TimeEntryFilterFactory, TimeStatusFilterFactory and TaskTypeFilterFactory employ this pattern.*

The Business Delegate Pattern. *The Business Delegate pattern is utilized in the TimeEntryManagerDelegate, TimeStatusManagerDelegate, and TaskTypeManagerDelegate classes.*

The Singleton Pattern. *While not technically a singleton, the ManagerFactory behaves very similarly, and may be considered a close relative, if not a member of, the singleton pattern.*

1.2Industry Standards

EJB

JDBC

1.3 Required Algorithms

There were no algorithms required and the component is straightforward enough. We will discuss the database schema, and the method of searching in this section.

1.3.1 Data Mapping

This section will deal with mapping the different values in the beans to their respective columns. The developer is responsible for generating the necessary SQL to insert/retrieve the data in the beans from the appropriate columns. The SQL will require some simple table joins at the most. Please refer to the ERD diagrams for more information [see TimeTrackerTimeEntry_ERD.jpg].

1.3.1.1 TimeEntry Class and time_entry Table

- Column time_entry_id maps to the id property in the TimeTrackerBean base class.
  Note: The id generator is used to create a new id when a new record is inserted.
- Column company_id maps to the companyId property in the BaseEntry superclass.
- Column time_status_id maps to the timeStatus property in the TimeEntry class.
- Column task_type_id maps to the taskType property in the TimeEntry class.
- Column description maps to the description property in the TimeEntry class.

- Column invoice_id maps to the invoice id property in the TimeEntry class.
- Column creation_date maps to the creationDate property in the TimeTrackerBean base class.
- Column creation_user maps to the creationUser property in the TimeTrackerBean base class.
- Column modification_date maps to the modificationDate property in the TimeTrackerBean base class.
- Column modification_user maps to the modificationUser property in the TimeTrackerBean base class.

1.3.1.2 TaskType Class and task_type Table

- Column task_type_id maps to the id property in the TaskType class.
  Note: The id generator is used to create a new id when a new record is inserted.
- Column description maps to the description property in the TaskType class.
- Column active maps to the active property in the TaskType class.
- Column creation_date maps to the creationDate property in the TimeTrackerBean base class.
- Column creation_user maps to the creationUser property in the TimeTrackerBean base class.
- Column modification_date maps to the modificationDate property in the TimeTrackerBean base class.
- Column modification_user maps to the modificationUser property in the TimeTrackerBean base class.

The companyId property is filled by retrieving the companyId from company_id column in comp_task_type table with task_type_id equal to the desired task_type.

1.3.1.3 TimeStatus Class and time_status Table

- Column time_status_id maps to the id property in the TaskType class.
  Note: The id generator is used to create a new id when a new record is inserted.
- Column description maps to the description property in the TaskType class.
- Column creation_date maps to the creationDate property in the TimeTrackerBean base class.
- Column creation_user maps to the creationUser property in the TimeTrackerBean base class.
- Column modification_date maps to the modificationDate property in the TimeTrackerBean base class.
- Column modification_user maps to the modificationUser property in the TimeTrackerBean base class.

1.3.2 Searching

Searching is executed in this component by using the Search Builder component and the FilterFactory layer. First off, the developer needs to develop a search query off which to base the search on. The search query will retrieve all the necessary attributes, and join with the tables of any possible criterion. Once a query has been defined, in the constructor, the appropriate FilterFactory will be initialized with the column names used in the query. This FilterFactory can then be returned by the appropriate getFilterFactory method.

The user may then use the Factory to build the search filters. Once the filters are

provided back to the DAO implementation, it may use the DatabaseSearchStrategy to build the search. The DatabaseSearchStrategy will then add the necessary WHERE clauses to constrain the search to the search criterion specified in the Filters.

Example:

The following query may be used as the context for searching the Time Entries (we reduce the retrieved data to only the id, but developer may modify it to retrieve all relevant fields):

```
SELECT
      time_entry_id
FROM
      time_entry
INNER JOIN
      time_reject_reason
      ON
      time_reject_reason.time_entry_id =
time_entry.time_entry_id
WHERE
```

From this context, the Search Builder can then add the different WHERE clauses, depending on the filter that was provided. For example, if a filter for the Time Entry was created using createCompanyIdFilter, then the Search Builder would add:

```
WHERE (continued from above)
      time_entry.company_id = [value of filter]
```

To accomplish this, in the constructor of the DAO, the FilterFactory must be configured according to the context query that was used. In this example, the mapped value for the COMPANY_ID_COLUMN_NAME should be "time_entry.company_id".

1.3.3 Auditing

The method implementation notes contain the audit header information that should be used when performing the audit. The following clarifies how to create AuditDetail objects to add to the AuditHeader. When creating a new entry (audit header action type is CREATE), then the oldValue for each detail is null. When deleting an entry (audit header action type is DELETE), then the newValue for each detail is null. When updating an entry, the old value is retrieved from the datastore to populate the AuditDetail's old value.

1.3.4 Transaction Rollback

The business methods of the SessionBean should signal a rollback to the EJB container in the event that an exception was encountered. The SessionBean should catch any exceptions and call setRollbackOnly() method in the SessionContext before rethrowing the exception.

1.4Component Class Overview

Package com.topcoder.time.tracker.entry.time

TimeEntryManager (interface)

This interface represents the API that may be used in order to manipulate the various details involving a Time Tracker Time Entry. CRUDE and search methods are

provided to manage the Time Entries inside a persistent store. There are also methods for the manipulation of RejectReasons associated with the Time Entry.

TimeEntryDAO (interface)

This is an interface definition for the DAO that is responsible for handling the retrieval, storage, and searching of Time Tracker TimeEntry data from a persistent store. It is also responsible for generating ids for any entities within its scope, whenever an id is required.

TimeEntryFilterFactory (interface)

This interface defines a factory that is capable of creating search filters used for searching through Time Tracker Time Entries. It offers a convenient way of specifying search criteria to use. The factory is capable of producing filters that conform to a specific schema, and is associated with the TimeEntryManager that supports the given schema.

BaseFilterFactory (interface)

This is a base FilterFactory interface that provides filter creation methods that may be used for filters of any Time Tracker Bean. It encapsulates filters for the common functionality - namely, the creation and modification date and user.

TimeEntryRejectReasonDAO (interface)

This is an interface definition for the DAO that is responsible for handling the association between Time Tracker Entries and the Reject Reasons. Simple CRUDE methods are specified.

TimeStatusManager (interface)

This interface represents the API that may be used in order to manipulate the various details involving a TimeStatus. CRUDE and search methods are provided to manage the TimeStatuses inside a persistent store.

TimeStatusDAO (interface)

This is an interface definition for the DAO that is responsible for handling the retrieval, storage, and searching of TimeStatus data from a persistent store. It is also responsible for generating ids for any entities within it's scope, whenever an id is required.

TimeStatusFilterFactory (interface)

This interface defines a factory that is capable of creating search filters used for searching through TimeStatuses. It offers a convenient way of specifying search criteria to use. The factory is capable of producing filters that conform to a specific schema, and is associated with the TimeStatusManager that supports the given schema.

TaskTypeManager (interface)

This interface represents the API that may be used in order to manipulate the various details involving a TaskType. CRUDE and search methods are provided to manage the TaskTypes inside a persistent store.

TaskTypeDAO (interface)

This is an interface definition for the DAO that is responsible for handling the retrieval, storage, and searching of TaskType data from a persistent store. It is also responsible for generating ids for any entities within it's scope, whenever an id is required.

TaskTypeFilterFactory (interface)

This interface defines a factory that is capable of creating search filters used for

searching through TaskTypes. It offers a convenient way of specifying search criteria to use. The factory is capable of producing filters that conform to a specific schema, and is associated with the TaskTypeManager that supports the given schema.

TimeEntry

This is the main data class of the component, and includes getters and setters to access the various properties of a Time Entry, A Time Entry is an amount of time spent by a user on a specific task.

TimeStatus

This is a bean that represents a TimeStatus, which represents a possible state that a TimeEntry can have in the context of the Time Tracker system.

TaskType

This is a bean class that indicates a type of task for which a Time Entry may be submitted.

TimeEntryManagerImpl

This is a default implementation of the TimeEntryManager interface. it utilizes instances of the TimeEntryDAO in order to fulfill the necessary CRUDE and search operations defined for the Time Tracker User. It also uses TimeEntryRejectReasonDAO to associate any RejectReasons with the TimeEntries.

TaskTypeManagerImpl

This is a default implementation of the TaskTypeManager interface. it utilizes instances of the TaskTypeDAO in order to fulfill the necessary CRUDE and search operations defined for the Time Tracker TaskType.

TimeStatusManagerImpl

This is a default implementation of the TimeStatusManager interface. it utilizes instances of the TimeStatusDAO in order to fulfill the necessary CRUDE and search operations defined for the TimeStatus.

ManagerFactory

This class is used to create Manager implementations to use by the delegate classes.

Package com.topcoder.time.tracker.entry.time.db

BaseDAO

This is a base DAO class that encapsulates the common elements that may be found within a DAO such as the connection details, id generator, search strategy and audit manager.

DbTimeEntryDAO

This is an implementation of the TimeEntryDAO interface that utilizes a database with the schema provided in the Requirements Section of Time Tracker Time Entry 3.1.

DbTimeStatusDAO

This is an implementation of the TimeStatusDAO interface that utilizes a database with the schema provided in the Requirements Section of Time Tracker Time Entry 3.1.

DbTaskTypeDAO

This is an implementation of the TaskTypeDAO interface that utilizes a database

with the schema provided in the Requirements Section of Time Tracker Time Entry 3.1.

DbTimeEntryRejectReasonDAO
This is an implementation of the TimeEntryRejectReasonDAO interface that utilizes a database with the schema provided in the Requirements Section of Time Tracker Time Entry 3.1.

DbBaseFilterFactory
This is an implementation of the BaseFilterFactory that may be used for building searches in the database.

DbTimeEntryFilterFactory
This is an implementation of the TimeEntryFilterFactory that may be used for building searches in the database.

DbTimeStatusFilterFactory
This is an implementation of the TimeStatusFactory that may be used for building searches in the database.

DbTaskTypeFilterFactory
This is an implementation of the TaskTypeFactory that may be used for building searches in the database.

Package com.topcoder.time.tracker.entry.time.ejb

TimeEntryManagerLocal
Local interface for TimeEntryManager. It contains exactly the same methods as TimeEntryManager interface.

TimeStatusManagerLocal
Local interface for TimeStatusManager. It contains exactly the same methods as TimeStatusManager interface.

TaskTypeManagerLocal
Local interface for TaskTypeManager. It contains exactly the same methods as TaskTypeManager interface.

TimeEntryManagerLocalHome
LocalHome interface for the TimeEntryManager. It contains only a single no-param create method that produces an instance of the
local interface. it is used to obtain a handle to the Stateless SessionBean.

TimeStatusManagerLocalHome
LocalHome interface for the TimeStatusManager. It contains only a single no-param create method that produces an instance of the
local interface. it is used to obtain a handle to the Stateless SessionBean.

TaskTypeManagerLocalHome
LocalHome interface for the TaskTypeManager. It contains only a single no-param create method that produces an instance of the
local interface. it is used to obtain a handle to the Stateless SessionBean.

TimeEntryManagerDelegate
This is a Business Delegate/Service Locator that may be used within a J2EE application. It is responsible for looking up the local interface
of the SessionBean for TimeEntryManager, and delegating any calls to the bean.

TaskTypeManagerDelegate

This is a Business Delegate/Service Locator that may be used within a J2EE application. It is responsible for looking up the local interface of the SessionBean for TaskTypeManager, and delegating any calls to the bean.

TimeStatusManagerDelegate

This is a Business Delegate/Service Locator that may be used within a J2EE application. It is responsible for looking up the local interface of the SessionBean for TimeStatusManager, and delegating any calls to the bean.

TimeEntryManagerSessionBean

This is a Stateless SessionBean that is used to provide business services to manage TimeEntries within the Time Tracker Application.
It contains the same methods as TimeEntryManager, and delegates to an instance of TimeEntryManager.

TimeStatusManagerSessionBean

This is a Stateless SessionBean that is used to provide business services to manage TimeStatuses within the Time Tracker Application.
It contains the same methods as TimeStatusManager, and delegates to an instance of TimeStatusManager.

TaskTypeManagerSessionBean

This is a Stateless SessionBean that is used to provide business services to manage TaskTypes within the Time Tracker Application.
It contains the same methods as TaskTypeManager, and delegates to an instance of TaskTypeManager.

1.5Component Exception Definitions

DataAccessException

This exception is thrown when a problem occurs while this component is interacting with the persistent store. It is thrown by all the DAO and Manager interfaces (and their respective implementations).

UnrecognizedEntityException

This exception is thrown when interacting with the data store and an entity cannot be recognized. It may be thrown when an entity with a specified identifier cannot be found. It is thrown by all the Manager and DAO interfaces (and their implementations).

InvalidCompanyException

This exception is thrown when there is an attempt to relate a Time Tracker entity to a TimeEntry whose companies do not match. For example, adding a TaskType to a TimeEntry when their companies are different.

ConfigurationException

This exception is thrown when there is a problem that occurs when configuring this component.

1.6Thread Safety

This component is not completely thread-safe, The Bean instances are not thread safe,

and it is expected to be used concurrently only for read-only access. Otherwise, each thread is expected to work with its own instance.

The Manager classes are required to be thread-safe, and this is achieved via the thread safety of the implementations of the DAO Layer, and the FilterFactory Layer.

The FilterFactory layer is thread-safe by virtue of being immutable.

Thread-safety of the J2EE layer is dependent on the statelessness and thread safety of the manager classes, and also on the application container controlling them. The only thing to pay attention to is that the inner state of the classes are not modified after creation.

2.Environment Requirements

2.1Environment

Java 1.4

2.2TopCoder Software Components

DB Connection Factory 1.0 – for creating the DB connections

Base Exception 1.0 – base class for custom exception is taken from it

Object Factory 2.0 – is used to configure the component.

ID Generator 3.0 – for generating IDs in the persistence implementation.

JNDI Context Utility 1.0 - for retrieving the LocalHome interface implementations from the container.

Search Builder 1.3.1 – is used to perform the searches

Time Tracker Audit 3.2 – is used to perform the optional audit.

Time Tracker Base Entry 3.2 – is used to provide the base entry class.

Time Tracker Reject Reason 3.2 – is used to retrieve the RejectReason information.

Time Tracker Common 3.2 – is used to provide the TimeTrackerBean base class.

ConfigManager 2.1.5 – was used to configure the Business Delegate classes.

2.3Third Party Components

None

3.Installation and Configuration

3.1Package Names

com.topcoder.time.tracker.entry.time
com.topcoder.time.tracker.entry.time.db
com.topcoder.time.tracker.entry.time.ejb

3.2Configuration Parameters

This component relies on Object Factory 2.0 for configuration that is based from a file.

For the ejb portion, the following config parameters are used by all business delegates:

For TimeEntryManagerDelegate and TimeStatusManagerDelegate and TaskTypeManagerDelegate:

Property Name: *contextName*

Property Description: This is the context name used to retrieve the home object of the respective session bean. If not specified, then the default context provided by JNDIUtils is used.
Is Required: *Optional*

Property Name: *localHomeName*
Property Description: This is the name used to provide the LocalHome object from the configured context.
Is Required: *Required*

3.3Dependencies Configuration

All the dependencies are to be configured according to their component specifications.

Do note that the default configuration for Search Builder is desired to be able to build the search string correctly.

4.Usage Notes

4.1Required steps to test the component

Extract the component distribution.

Follow Dependencies Configuration.

Execute 'ant test' within the directory that the distribution was extracted to.

4.2Required steps to use the component

Configure the dependency components.

4.3 Demo

Please see the ejb-jar.xml file in docs directory for the deployment descriptor for the J2EE layer of this component.

We will assume here that everything is configured properly. Try/catch clauses have been removed to enhance clarity.

```
// Create a TimeEntryManager using the DB DAOs, and rejectReasonManager
TimeEntryManager entryManager = new
TimeEntryManagerDelegate("applicationNamespace.time.entry");
TaskTypeManager taskManager = new
TaskTypeManagerDelegate("applicationNamespace.time.tasktype");
TimeStatusManager statusManager = new
TimeStatusManagerDelegate("applicationNamespace.time.status");

//  4.3.1 Creating a new Time Entry with a Task Type of "Topcoder Design" and a Status of "Pending"
TimeEntry newEntry = new TimeEntry();
newEntry.setCompanyId(tcCompanyId);
newEntry.setDescription("Pluggable Persistence 3.0");
newEntry.entryDate(new Date());
newEntry.setBillable(false);
newEntry.setHours(10);
```

```
// Retrieve a task type with a description of "Topcoder Design"
TaskTypeFilterFactory taskFilterFactory = taskManager.getTaskTypeFilterFactory();

List taskCriterion = new ArrayList();
taskCriterion.add(taskFilterFactory.createCompanyIdFilter(TOPCODER_ID));
taskCriterion.add(taskFilterFactory.createDescriptionFilter("Design"));

// Perform the search
TaskType TcDesignType = taskManager.search(taskCriterion)[0];

// Assign the type to the entry
newEntry.setTaskType(TcDesignType);

// Retrieve a status with a description of "Pending"
TimeStatus pendingStatus = statusManager.getTimeStatus(PENDING_ID));

// Assign the status to the entry
newEntry.setStatus(pendingStatus);

// Register the entry with the manager, with auditing.
entryManager.createTimeEntry(newEntry, true);
```

// 4.3.2 Changing the Entry Details

```
// Retrieve an entry from the manager
TimeEntry changingEntry = entryManager.getEntry(entryIdToChange);

// Update the entry details.
changingEntry.setDescription("Pluggable Persistence 3.1");
changingEntry.setStatus(approvedStatus);

// Update the entry in the manager
entryManager.updateEntry(changingEntry);

// Add a RejectReason to the Entry
entryManager.addRejectReasonToEntry(changingEntry, rejectReasonId);
```

// 4.3.3 Search for a group of Entries with "design" in the description and status of "pending" and delete them. The delete operation should be atomic.

```
// Define the search criteria.
TimeEntryFilterFactory filterFactory = entryManager.getTimeEntryFilterFactory();
List criteria = new ArrayList();
criteria.add(filterFactory.createDescriptionFilter("SS:design"));
criteria.add(filterFactory.createStatusFilter(pendingStatus));

// Create a search filter that aggregates the criteria.
Filter searchFilter = new AndFilter(criteria);

// Perform the actual search.
TimeEntry[] matchingEntries = entryManager.searchEntries(searchFilter);

// Delete the users using atomic mode; auditing is performed.
entryManager.deleteEntries(matchingEntries, true, true);
```

5.Future Enhancements

Provide implementations for different RDBMS.