Time Tracker Base Entry 3.2 Component Specification

Changes for version 3.2 are in red.
The Base Entry custom component is part of the Time Tracker application. It provides an abstraction of the high level entry that an employee enters into the system. It is an abstract entity, which is extended by Time Entry, Expense Entry and Fixed Billing entry. This component handles the common entry business logic required by the application.

1. Design

The main idea in this design is to provide an abstraction, which can then be used by other components as an extension point. Mainly, the `BaseEntry` bean definition is meant as a base definition for other components to use. This is coupled with a persistence layer, which effectively provides the ability to customize cut off times for submission of entries.

Thus we can summarize that the BaseEntry models an abstraction of an entry, which can be submitted to the system, provided that the cut off time has not been reached for the submitting company. This aspect can be customized (per company rather than per entry type) so that different companies can have different cut off times associated with them.

1.1.1 Base implementations decisions

In general this is a very simple design but it is very important that it is extensible. Thus persistence will be abstracted out through an abstract DAO class, which will 'know' how to deal with connection creation (from configuration) etc… This way we can easily just extend this class with the proper SQL commands (or whatever) and be able to plug in a new implementation (of a different database source) without the need to change any client code.

1.1.2 Additional functionality

Apart from some functional enhancements such as high configurability this component will also offer some functional enhancements in the area of logging.

Logging is removed from version 3.2.

~~1.1.2.1 Logging~~

~~The main reason why logging is provided here is that we are dealing with two tiers: the management tier and the data persistence tier. As calls are being delegated to the DAO it would be nice to see what was done (and who is doing it – user info) this is especially true for any failures, which would be logged with user name and a time stamp (as well as a unique message id). We will be basically logging the following aspects:~~

1. ~~This component must log all entries and exits into and out of methods and loops. It also logs all error conditions. This section shows where to place the logging code. A sample method with a loop would look like this.~~

```
method_start {
  //declarations
  //loop start
  during_loop_condition
  {
    //some code
    //possible inner loops
    //possible method exits
  }
}
```

2. After logging code is inserted it must look like this. Note that the method documentation in the UML file does not have any reference to logging. However *every method*, except for those of the exception classes, must be logged.

```
method_start {
  //log method_entry

  //declarations
  //loop start
  //log loop_entry
  during_loop_condition
  {
    //some code
    //possible inner loops
    //log similarly for inner loop

    //possible method exits
    //log method_exit
  }
  //log loop_exit
  //log method_exit
}
```

The logging code itself is trivial. We log method entries and exits at the DEBUG level. Any exceptional conditions are logged at the ERROR level.

Version 3.2:
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.

In order to accommodate the new requirements, the following changes are necessary:

1. A POJO Delegate and a Stateless Session Bean are added. These provide the functionality provided by EntryManager in the current design, except that the Session Bean relies on container managed transactions.
2. The existing EntryManager class is removed from the design, replaced by an EntryManager interface that provides the same set of CRUD methods. This interface is implemented by EntryDelegate, EntrySessionBean, and EntryLocalObject.
3. The existing logging functionality is removed, since it cannot be used from with a Stateless Session Bean.
4. Auditing functionality that relies on the Time Tracker Audit component is added.
5. The deployment descriptor will need to set the transaction level for EntrySessionBean to REQUIRED.
6. Since the transaction is now handled by the EJB container, transaction management is removed from InformixCutoffTimeDAO.

The new EntrySessionBean will rely on an internal CutoffTimeDAO to handle all of its operations. It will configure the DAO using configuration properties from the Configuration Manager component. The new EntryDelegate class, which provides an interface to the EntrySessionBean and to the component as a whole, will also be configured through ConfigurationManager.

1.2 Design Patterns

The **strategy design pattern** will be used to create pluggable aspect of the DAO. This way the user will be able to plugin a new `CutoffTimeDao` implementation through the manager class and use the existing client code without any changes. Note that the plugin capabilities will be both programmatic as well as through a configurable file.

The **DAO pattern** is used to abstract the persistence layer from the rest of the application (i.e. from the manager)
Business Delegate: The EntryDelegate class functions as a business delegate that delegates all operations to a SessionBean.

1.3 Industry Standards

Java 1.4.x and 1.5.x

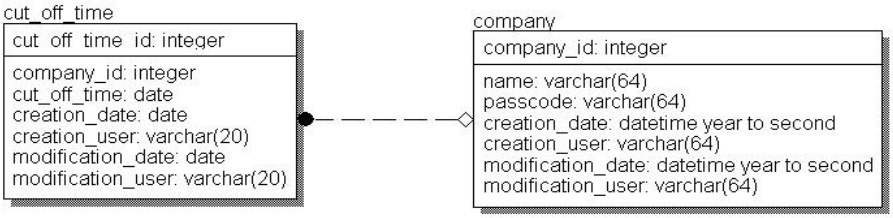
1.4 Required Algorithms

There are no special algorithms that were required by the RS but we will discuss here some common aspects that might be useful to the developers.

1.4.1 Database considerations

The DAO that we will be modeling will be accessing this particular table [cut_off_time].

Please note that the *date* in the below image should in fact be *datetime*.



Note that the DAO, which will wrap around this table, will simply provide a CRUD set of operations.

Creating a record

Step 1. Using the IDGenerator we create a new id for the new record
Step 2. We then create a prepared statement to create the record based on the data in the passed in information.
  (a) Note that we assume that the `creation_date` is the current date/time
  (b) We use the user passed in through the manager
  (c) We use the cutoff time without any validation. It is up to the caller to ensure proper semantics of the data being created.

Deleting a record

Step 1. We simply create a DELETE sql and execute it against the database. The deletion will be made based on `cut_off_time_id`. We will also make an assumption that we can delete based on the `company_id` as well (we assume a 1-1)

Reading a record

Step 1. We will be looking up the record based on the company_id (which we get from

the `BaseEntry.getCompanyId()` method) We then create a prepared statement to fetch the record with this id.

Updating a record

Step 1. We update the record with the specific `cut_off_time_id` , but we will also make an assumption that we can update based on the `company_id` as well (we assume a 1-1)

1.4.2 Checking if an entry can be submitted

This is quite simple and would be done against the DAO as follows:

Step 1. Using the `companyId` (`entry.getCompanyId()`) we fetch the record using the cut off date dao.
Step 2. Assuming the record exists (if not we throw an exception) we simply read the `"cut_off_date"` field value and compare it to current date/time. Here is how we do the comparison:
   (a) The assumption is that we only use the week day and the time from the date to establish the cutoff date/time. Thus for example if the week day is Wednesday and the time is "17:00:00" this means that the cut off time is any Wednesday at 5 pm. This in turn means that any entry that is entered before this day and time would qualify to be submitted for the week ending the previous Sunday
   (b) Once we have the cutoff time and week day we extract the weekday of the day date and the time for the input Base Entry (i.e. the date that the entry was entered). Then we do the comparison as follows: we check if the entry date falls on the cutoff day or before the cutoff day in which case we return true (i.e. the entry can be submitted for the previous week) otherwise we return false.

The general idea, to reiterate, is as follows:
Compare the entry date to the Sunday before the cut off, as the workweek default is Monday to Sunday, and the **current day to the cut off**. This means, using the above example of Wed 17:00 as the cut off, that an entry prior to the Monday Midnight would be allowed to be submitted. Note, the interval extends back 7 days prior to the Monday Midnight.

Consider this example:

Let us assume the following date structure:
M1 T1 W1 Th1 F1 Sa1 S1 M2 T2 W2 Th2 F2 Sa2 S2 M3 T3 W3 Th3 F3 Sa3 S3, where the first letter represents the day of the week and the number represents which relative week we are dealing with. Let us also assume that we are currently in week 2, and that we are using the above cutoff at 17:00, (we will use the Time zone of the server)

If an entry had a date of T2 and was submitted on M3 it would pass. If the same entry were submitted on Th3 it would fail. The entry on the same date, T2, should also be able to be submitted on a T2, or any day up to W3 at 17:00.

An entry with an entry date W1 submitted on M3 would fail. Another scenario, which should fail, would be an Entry on the date of T2 being submitted before T2.

Let us also assume that we are currently in the beginning of Sunday, note that any date from S2 on, should be allowed, up to W3 @ 17:00 as well. Again this is because the workweek is from Monday to Sunday.

1.4.3 Container managed transactions

Transactions in this version are container managed. This component doesn't need to commit any transactions. However, if an operation fails, the EntrySessionBean should call sessionContext.setRollbackOnly(). This will let the container know that the current transaction should be rolled back rather than committed. In addition, it will be necessary to set the transaction level of the various Session Beans to REQUIRED in the deployment descriptor.

1.4.4 Component Configuration

File I/O is not possible from a Stateless Session Bean. However, Configuration Manager loads the files specified in its configuration file at startup, and maintains configuration in memory thereafter. Because of this, it is possible to use Configuration Manager to configure the EntrySessionBean and the classes it depends on. In this design, configuring the EntrySessionBean is a simple matter of reading an Object Factory namespace and a key for the CutoffTimeDao to use from configuration and using ObjectFactory to instantiate it.
Configuring the EntryDelegate is slightly more complex. It will read the JNDI context_name property from configuration manager and use this to retrieve the needed Context using the Topcoder JNDIUtils component. It will then read the jndi_name from configuration and use this to retrieve a EntryLocalHome object from the Context, and use this LocalHome object to look up and retrieve the right EntryLocalObject that provides the local handle for EntrySessionBean. The EntryLocalObject object will be used by the EntryDelegate to complete all needed operations.

1.4.5 Auditing

Operations that modify database data will be audited. These include createCutoffDate, updateCutoffDate, and deleteCutoffDate. Auditing will be handled in the DAO.

The createAuditRecord() method of AuditManager requires that an AuditHeader be passed in. The constructed AuditHeader should be created in this way:

The short descriptions of each field in AuditHeader:

entityId: CutoffTimeBean.getCutoffId()
tableName: cut_off_time
companyId: user.getCompanyId()
actionType: INSERT, DELETE or UPDATE depending of the action type
applicationArea: TT_ENTRY
resourceId: not set
creationUser : username

For the create operation, the old value of the audit detail is null.
For the delete operation, the new value of the audit detail is null.

1.5 Component Class Overview

1.5.1 com.topcoder.timetracker.entry.base

BaseEntry

This is an abstraction of an entry. It is extended from the `TimeTrackerBean` and thus inherits the following:

- Entry ID – the unique entry ID number
- Creation Date – the date the entry was created
- Creation User – the username that created the entry
- Modification Date – the date the entry was modified
- Modification User – the username that modified the entry

What this class adds are the following:

- Company ID – the company Id associated with the entry
- Description – a brief description of the entry
- Entry Date – the date for the entry

- Reject Reasons – the reasons why the entry was rejected.

Please note that this class follows the JavaBeans standard as far as the following is concerned:

- The class is serializable (it is assumed that this is already taken care of in the base class – `TimeTrackerBean`)
- 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>()`.

This class is not thread-safe as it is mutable.

~~**EntryManager**~~

~~This class acts as something of a façade. It hides the fact that persistence operations are involved. What this class currently provides to the user is the functionality that sets such context information as logging, user name either through API (i.e. through a constructor) or through configuration (i.e. namespace based constructor)~~

~~In general this manager provides the following operations:~~

- ~~Update the cut off time – will update the cut off time for a company~~
- ~~Retrieve the cut off time – will get the cut off time for a company~~
- ~~Delete the cut off time – will remove the cut off time for the input company~~
- ~~Create a cut off time – will create a cut off time for a company~~
- ~~Can an entry be submitted – checks if an entry can be submitted based on some cut off time~~

~~This class is thread-safe as it is immutable.~~

EntryManager (interface)

This interface defines the common contract that EntryDelegate, EntrySessionBean, and EntryLocalObject all implement. These common methods are the main public functionality of the component.

The methods defined by this interface are basic CRUD methods for CutoffTimes, and a method to determine whether a BaseEntry can be submitted.

Implementations of this interface are required to be thread safe.

BaseDao

This is an abstraction of a DAO, which implements such common operations as getting and setting a connection as well as id generation. This way other database implementations can be quickly implemented.

In version 3.2, file based logging functionality is removed, and auditing through the Timetracker Audit Component is added.

It is assumed that implementation will be thread-safe.

CutoffTimeDao

This is a general CRUD interface for cut off date/time persistence.

In version 3.2, methods that modify data take a Boolean parameter to determine if they should be audited or not. In addition, it is added to the contract that transactions should not be managed by the Dao.

Implementations should be thread-safe.

CutOffTimeBean

This is a basic java bean, which encapsulates all the data for a “cut_off_time” table entity.

Please note that this class follows the JavaBeans standard as far as the following is concerned:

- 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>()`.

This class is not thread-safe as it is mutable.

~~**LoggingContext**~~
~~This is a simple logging context class, which acts as a configurable logger. Basically this is a convenience class that can be used to log activity, or error messages.~~
~~It exposes a single method for logging which accepts the level (error, Info, etc…) and the message. Internally it generates a unique id for each message so that messages can be tracked or correlated.~~
~~The id generation is configuration based.~~
~~This class is thread-safe as it simply acts as a logging utility and is immutable once initialized.~~

1.5.2 com.topcoder.timetracker.entry.base.persistence

InformixCutoffTimeDao
This is a specific implementation of the CutoffDateDao interface and it implements the CRUD functionality for a specific `cut_off_time` table presented for the Informix database.
In version 3.2, optional auditing is added to those methods that modify data. In addition, all transaction management is removed. The constructor is altered to remove logger and add AuditManager.
It is thread-safe as it is immutable.

1.5.3 com.topcoder.timetracker.entry.base.ejb

EntryLocalObject
Local interface for EntitySessionBean. It contains exactly the same methods as EntityManager interface.
Implementation will be generated by EJB container and thread-safety is dependent on the container.

EntryLocalHome
LocalHome interface for the EntitySessionBean. 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.
Implementation will be generated by EJB container and thread-safety is dependent on the container.

EntryDelegate
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 EntrySessionBean, and delegating any calls to the bean.
This class is thread safe, since all state is modified at construction.

EntrySessionBean
This is a Stateless SessionBean that is used to provided business services to manage BaseEntry and CutoffTimeBean objects within the Time Tracker Application.
It implements the EntryManager interface and delegates to an instance of CutoffTimeDao. Transactions for this bean are handled by the EJB Container.
It is expected that the transaction level declared in the deployment descriptor for this class will be REQUIRED.
All method calls on methods in EntryManager interface except for canSubmitEntry() are delegated to an instance of CutoffTimeDao through the getDao() method. The getDao() method will return the value of the dao field.

1.6 Component Exception Definitions

1.6.1 Custom Exceptions

- **ConfigurationException**: This is a general-purpose exception created for the component to signal that something went wrong during configuration processing.
- **PersistenceException:** This is an exception that is thrown when an issue with persistence (connection issue, missing table issue, etc…) This is a broad exception.
- **EntryNotFoundException:** This is a specific persistence exception (it is extended from it) which signals that an entry that was expected to be found was not (in persistence)
- **DuplicateEntryException:** This is a specific persistence exception (it is extended from it.) This would be thrown in the very rare situation where an entry record is being persisted but the id (i.e. entry) already exists.
- **EntryManagerException:** This is a general manager exception which will be used to wrap all other exceptions when internal processing fails for some reason (like persistence for example)
- **IdGenerationException**: This is a specific exception for situations when id generation might fail. It could happen that all ids have been exhausted (or any other failure

1.6.2 System exceptions

- **IllegalArgumentException**: Exception thrown in various methods where value is invalid or null. In our case this will happen if required data (such as date in BaseEntry) is null or empty (as in an empty description for example) Please consult the UML method documentation for further details.

1.7 Thread Safety

This component is thread-safe on the level of EntryDelegate. This is accomplished by making the EntryDelegate's state immutable after initialization. EntryLocalObject and EntryLocalHome are generated by the EJB container, which will ensure that they are used in a thread safe manner. EntrySessionBean is not inherently thread safe, but it is managed by the EJB container which will ensure that it is used by no more than one thread simultaneously.
Please note that the BaseEntry, being modeled after the JavaBeans convention, is not thread-safe as it is mutable. It could be made thread safe by synchronizing each method but this is not necessary in this component as it is assumed that the user will use the beans in appropriate manner, and will not have multiple threads changing the bean's state.
In addition, the DAO classes presented in this design must be thread-safe since they will be possible used by a number of threads at the same time.

As far as ACID and isolation control is concerned this is something that the database administrator would have to look into since transaction control doesn't guarantee complete data integrity unless setup properly for the strictest of isolation levels.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.4
- Compile target: Java 1.4, Java 1.5

2.2 TopCoder Software Components

- **BaseException 1.0:** Used for custom exception definitions. It allows TopCoder to control future aspects of error handling if necessary.
- **Configuration Manager 2.1.5:** is used to configure EntryDelegate

- **Object Factory 2.0.1:** This is used in configuring (and creating) instances of specific CutoffTimeDao implementations. In version 3.2, usage of this component is changed to avoid use of Configuration Manager.
- **DB Connection Factory 1.0:** which is used for configuration based connection creation.
- **ID Generator 3.0;** which is used for configuration based ID generator (and id generation) creation.
- **Reject Reason 1.0:** which is used by the BaseEntry class internally as one of the constituent data members. It tracks the reject reasons for the particular entry.
- **Time Tracker Common 1.0:** which is used for some common extension classes such as `TimeTrackerBean` from which the BaseEntry class extends.
- ~~**Logging Wrapper 1.2:** which is used in this component to log activity.~~
- **Time Tracker Audit 3.2:** used to audit methods that mutate data
- **JNDI Utility 2.0** - used to look up EntryLocalHome

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.timetracker.entry.base
com.topcoder.timetracker.entry.base.persistence
com.topcoder.timetracker.entry.base.ejb

3.2 Configuration Parameters

The following properties may be specified to the EntryDelegate class through Configuration Manager.

context_name	This is the context name used to retrieve the home object of the respective session bean.	Optional; If not specified, then the default context provided by JNDIUtils is used	Any valid JNDI context.
jndi_home	This is the name used to retrieve the EJBLocalHome object for EntrySessionBean	Required	Valid name for EJBLocalHome object in context

The following properties should be specified in the default namespace (com.topcoder.timetracker.entry.base.ejb.EntrySessionBean) for EntrySessionBean.

of_namespace	This is the namespace used for the Object Factory	Required	A valid object factory namespace
dao_key	This is the key used to retrieve the CutoffTimeDao	Required	A key for a CutoffTimeDao

3.3 Dependencies Configuration

The company table is assumed to have all the necessary entries present when this component creates/updates records that link to records in the database.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

It is assume that all necessary tables have been created and if needed populated. It is also assumed that configuration has been properly set up (if used).

4.3 Demo

The following demo is new with version 3.2, since the usage of the component has changed so that EntryDelegate, rather than the old EntryManager class, is the central point of access for the component.

In version 3.1, each operation occurred in its own transaction. Transactions are now managed by the EJB container, and the scope of the transactions will be determined on the application level and may include operations from other components.

Create an EntryDelegate

Assume proper configuration for EntryDelegate and EntrySessionBean:

```
// create a new EntryDelegate
EntryDelegate delegate = new EntryDelegate("com.topcoder.timetracker.entry.base");
```

Use the delegate

```
// create a new CutoffDateBean to store the minimal data
CutoffDateBean cotBean = new  CutoffDateBean();
cotBean.setCompanyId(10123);
cotBean.setCutoffTime(new Date());
// Create an entry of cutoff time for a specific company with id “10123"
// Audit this operation using Timetracker Audit Component
delegate.createCutoffTime(cotBean, true);
// get the cutoff time for a specific company
CutoffDateBean cutofftime = delegate.fetchCutoffTimeByCompanyId(10123);
// we can make some change to the retrieved cutoff time, and update it
long cutoffId = cutofftime.getCutoffTimeId();
cutofftime.setCompanyId(12123);
delegate.updateCutoffTime(cutofftime, false);
// we can retrieve the cutoff time by id
CutoffDateBean cutofftime = delegate.fetchCutoffTimeById(cutoffId);
// delete operations may be optionally audited
delegate.deleteCutoffDate(cutofftime, true);
// assuming that we have some BaseEntry based entry we check if the entry can be
// submitted
boolean canSubmitFlag = manager.canSubmitEntry(someEntry);
```

5. Future Enhancements

- Provide other implementation of database persistence for the cut off time data.