Time Tracker Common 3.1 Component Specification

1. Design

The Time Tracker Common Custom component is part of the Time Tracker application. It provides for some basic common classes used by many of the components in TimeTracker. This component is simply a couple of classes. It encapsulates the persistence for payment types. It is packaged as a component in order to make it commonly available to many other components.

TimeTracker Bean This is the basis for every entity, which it persisted in to the database. It provides the common attributes, which are required when persisting. This is an abstract class that is to be used by many of the entities in Time Tracker.

Payment Terms are used on an invoice to determine the payment agreement for that invoice. Clients and projects have default payment terms associated with their records. The payment term extends TimeTrackerBean and has the following attributes: id, description, and term, active.

CommonManager provide an easy to use API that operates on PaymentTerms with many new methods added over the required ones.

The Time Tracker application will use this component to perform operations related to company and user authentication and authorization.

1.1 Industry Standards

JDBC, JavaBeans

1.2 Design Patterns

- **Strategy Pattern** – The PaymentTermDAO interface and their implementation implement this pattern, so that we can plug in to the other implementations easily.

- **Data Access Object (DAO) Pattern** – PaymentTermDAO implementations implement this pattern.

1.3 Required Algorithms

- None.

1.4 Component Class Overview

Package com.topcoder.timetracker.common

TimeTrackerBean

Bean base class used to represent all entities of the Time Tracker Application, which it persisted in to the database. It contains all the properties which are common to these entities (id, creation date, modification date, creation user, modification user and changed flag). It is also Serializable to support network transfer, and state persistence.

PaymentTerm

This bean represents the Payment Terms that are used on an invoice to determine the payment agreement for that invoice. Clients and projects have default payment terms associated with their records. The PaymentTerm extends TimeTrackerBean and has the following attributes: term, description, active.

CommonManager:

This interface defines the contract to manage the PaymentTerms in the persistence for Time Tracker Application. Contains operations like add, update, retrieve, delete.

SimpleCommonManager:

This class implements the CommonManager interface and it uses the PaymentTermDAO object to manage the PaymentTerms in the persistence, and it also has a configurable recentDays used in the retrieveRecentlyCreatedPaymentTerms and retrieveRecentlyModifiedPaymentTerms methods. Contains operations like add, update, retrieve, delete.

PaymentTermDAO

This interface defines the necessary methods that a PaymentTerm DAO should support. Add, retrieve, update and delete methods are provided.

Package com.topcoder.timetracker.common.persistence

DatabasePaymentTermDAO:

Database implementation of the PaymentTermDAO interface. It is capable of persisting and retrieving PaymentTerm information from the database. Add, Retrieve, Update, Delete methods are provided. DB Connection is realized using DBConnectionFactory. An IDGenerator instance is used to generate ids for PaymentTerms that should be added in the persistence.

1.5 Component Exception Definitions

1. Standard Exceptions

IllegalArgumentException
This exception is thrown when the parameters passed to a function invalid. (e.g. empty string, null argument etc.) Please refer to the method exception documentation for more details. Empty string means the length of string is zero after it is trimmed.

2. Custom Exceptions

CommonManagementException

This exception extends the BaseException and is thrown from the CommonManager interface. It is also the parent exception class for all the other custom exceptions in this design.

CommonManagerConfigurationException

This exception extends the CommonMangementException, and it is thrown if the configured value is invalid, or any required configuration value is missing. It is also used to wrap the exceptions from ConfigManager and IDGenerationException.

PaymentTermDAOException

This exception is thrown when a problem occurs when performing operations with the PaymentTerms in the persistence.

PaymentTermNotFoundException

This exception is thrown if the involved PaymentTerm was not found in the persistence. This is thrown during update/delete/retrieve methods.

DuplicatePaymentTermException

This exception is thrown if the involved PaymentTerm was not found in the persistence. This is thrown during update/delete/retrieve methods.

1.6 Thread Safety

All the classes (excerpt the beans) are immutable and therefore thread safe.

TimeTrackerBean is not used directly only by the classes that extend it.

PaymentTerm is mutable, but multiple threads are advised to work with their own instances to ensure thread safety of the application.

External synchronization can be used to achieve the thread safety of this component.

NOTE: All the TCS components used in this component are either thread-safe or used in thread-safe way, so they won't affect the thread-safety of this component.

2. Environment Requirements

2.1 Environment

- Development language: Java1.4
- Compile target: Java1.4, Java1.5

2.2 TopCoder Software Components

- **Configuration Manager 2.1.5** – Configuration Manager is used to configure data in this component.
- **DB Connection Factory 1.0** – It is used to get connection to database.
- **Object Factory 2.0.1** – It is used to create PaymentTermDAO and DBConnectionFactory.
- **Base Exception 1.0** – This is used as a base for all custom exceptions.
- **ID Generator 3.0** – is used to create ids for PaymentTerms when they are added to persistence.

2.3 Third Party Components

- None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.timetracker.common

com.topcoder.timetracker.common.persistence

3.2 Configuration Parameters

For **SimpleCommonManager** class

Parameter	Description	Values
of_namespace	The namespace used to create ObjectFactory object. **Required**.	Must be non-empty string.
payment_term_dao_key	The key used to create PaymentTermDAO object from ObjectFactory. **Required**.	Must be non-empty string.
recent_days	Represents the number of recent days used to get the recently unblocked users. **Required**.	Must be positive int value or -1.

For **DatabasePaymentTermDAO** class

Parameter	Description	Values
of_namespace	The namespace used to create ObjectFactory object. **Required**.	Must be non-empty string.
db_connection_factory_key	The key used to create DBConnectionFactory object from ObjectFactory. **Required**.	Must be non-empty string.
connection_name	The connection name used to obtain db connection from DBConnectionFactory. **Required**.	Must be non-empty string.
id_generator_name	This property contains the name that will be used by IDGeneratorFactory. **Required**.	Must be non-empty string.

3.3 Dependencies Configuration

- None

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration. See build.xml for details.
- Refer to /test_files/UnitTests.readme
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

- None.

4.3 Demo

```
    private PaymentTerm getPaymentTermToAdd() {

        //No need to set creation date, modification date, modification user
        //if you're planning to add payment term through SimpleCommonManager
        PaymentTerm paymentTerm = new PaymentTerm();
        paymentTerm.setCreationUser("creation_user");
        paymentTerm.setDescription("description");
        paymentTerm.setActive(true);
        paymentTerm.setTerm(2);
        return paymentTerm;
    }


        PaymentTerm first = this.getPaymentTermToAdd();
        PaymentTerm second = this.getPaymentTermToAdd();
        PaymentTerm third = this.getPaymentTermToAdd();

        PaymentTerm requiredPaymentTerm;
        PaymentTerm[] requiredPaymentTerms;

        SimpleCommonManager scManager = new SimpleCommonManager();
        // 1. Add a PaymentTerm to persistence
        scManager.deleteAllPaymentTerms();
        scManager.addPaymentTerm(first);
        scManager.addPaymentTerm(second);

        //If you're planning to use DAO to add payment term directly
        //ensure the creation date is not null and equals modification date,
and not exceeds current date
        //and ensure the modification user is set with a valid value
        Date currentDate = new Date();
        third.setCreationDate(currentDate);
        third.setModificationDate(currentDate);
        third.setModificationUser("modificationUser");
        dao.addPaymentTerm(third);

        long givenId = second.getId();
        long[] ids = {first.getId(), second.getId(), third.getId()};

        // 2. Modify a PaymentTerm
        second.setTerm(1);
        second.setActive(false);
        second.setDescription("Term changes to 1 day");
        second.setModificationUser("me");

        // 3. Update a PaymentTerm
        scManager.updatePaymentTerm(second);

        //If you're planning to use DAO to update payment term directly
        //ensure the creation date < modification date, and modification date
does not exceeds current date
        third.setTerm(1);
        third.setActive(false);
        third.setDescription("Term changes to 1 day");
        third.setModificationUser("me");
        currentDate = new Date();
        third.setCreationDate(new Date(currentDate.getTime() - 1));
        third.setModificationDate(currentDate);
        dao.updatePaymentTerm(third);

        // 4. Retrieve a PaymentTerm using its id
        // 4.1 Retrieve a updated PaymentTerm
        requiredPaymentTerm = scManager.retrievePaymentTerm(givenId);
        // 4.2 Retrieve a non-exist PaymentTerm
        requiredPaymentTerm = scManager.retrievePaymentTerm(10);

        // 5. Retrieve some PaymentTerms using their ids
        requiredPaymentTerms = scManager.retrievePaymentTerms(ids);

        // 6. Retrieve all active PaymentTerms
        requiredPaymentTerms = scManager.retrieveActivePaymentTerms();

        // 7. Retrieve recently created PaymentTerms (with configured recent
number of days)
        requiredPaymentTerms =
scManager.retrieveRecentlyCreatedPaymentTerms();

        // 8. Retrieve recently created PaymentTerms (with specified number
of days)
        requiredPaymentTerms =
scManager.retrieveRecentlyCreatedPaymentTerms(Integer.MAX_VALUE);

        // 9. Retrieve recently modified PaymentTerms (with configured recent
number of days)
        requiredPaymentTerms =
scManager.retrieveRecentlyModifiedPaymentTerms();

        // 10. Retrieve recently modified PaymentTerms (with specified number
of days)
        requiredPaymentTerms =
scManager.retrieveRecentlyModifiedPaymentTerms(Integer.MAX_VALUE);

        // 11. Retrieve all PaymentTerms
        requiredPaymentTerms = scManager.retrieveAllPaymentTerms();

        // 12. Delete a non-exist PaymentTerm
        try {
            scManager.deletePaymentTerm(10);
            fail("PaymentTermNotFoundException expected");
        } catch (PaymentTermNotFoundException e) {
            assertEquals(10, e.getProblemPaymentTermId());
        }

        // 13. Delete PaymentTerms corresponding to the given ids
        scManager.deletePaymentTerms(ids);

        // 14. Delete all PaymentTerms from persistence
        scManager.deleteAllPaymentTerms();
          requiredPaymentTerms = scManager.retrieveAllPaymentTerms();
```

5. Future Enhancements

Other database systems maybe plugged in for some client environments. Multiple user stores may be used for the same client environment.