Expense Entry 1.0 Component Specification

1. Design

The Expense Entry custom component is part of the Time Tracker application. It provides an abstraction of an expense entry that an employer enter into the system on a regular basis. This component handles the persistence and other business logic required by the application.
The manager classes handle the business logic. They provide a façade for the user to the functionality offered by this component: expense entry management, expense entry types management, expense entry statuses management.
The following classes implement the model part of this application: ExpenseEntry, ExpenseEntryType and ExpenseEntryStatus.
The persistence layer is abstracted using an interface. This allows easy swapping of the persistence storage without changes to the rest of the component. The default implementation uses an Informix database for storage.

1.1 Design Patterns

The **strategy** pattern is used for abstracting the persistence implementation in the ExpenseEntryPersistence interface.
The **data value object** pattern is used in the ExpenseEntry, ExpenseEntryType and ExpenseEntryStatus classes. These classes facilitate the data exchange between the business logic and the persistence layer.

1.2 Industry Standards

SQL, JDBC

1.3 Required Algorithms

SQL statements are very simple select, insert, update and delete queries so, it is not necessary to provide them as algorithms.
I will provide an algorithm for the constructor of ExpenseEntryManager(the algorithms for the constructors oh the other two managers are almost identical).
The algorithm is:

```
//read the configuration file
ConfigManager cm = ConfigManager.getInstance();
String className = cm.getStringArray(namespace,"entry_persistence_class");
String connectionProducerName = cm.getStringArray(namespace,"
                                                connection_producer_name");

//create the persistence instance through reflection

Class c = Class.forName(className);
if(connectionProducerName == null) {
      persistence = (ExpenseEntryPersistence)c.newInstance();
} else {
       Constructor constr = c.getConstructor(new Class[]{Class.forName("String")});
       persistence = (ExpenseEntryPersistence) constr.newInstance(new
                                              Object[]{connectionProducerName)});
}
```

1.4 Component Class Overview

ExpenseEntry
This class holds the information about an expense entry. When creating an instance of this class the user has two options:
1) Use the default constructor and allow the GUID Generator component to generate a unique id
2) Use the parameterized constructor and provide an id for the ExpenseEntry instance; if the id already is contained by another entry from the ExpenseEntries table, then the newly created entry will not be added to the ExpenseEntries table
Also the user should not populate the creationDate and modificationDate fields, because if he does, the entry will not be added to the database. This fields will be handled automatically by the component (the current date will be used). When loading from the persistence, all the fields will be properly populated.

ExpenseEntryType
This class holds the information about an expense entry type.When creating an instance of this class the user has two options:
1) Use the default constructor and allow the GUID Generator component to generate a unique id
2) Use the parameterized constructor and provide an id for the ExpenseEntryType instance; if the id already is contained by another type from the ExpenseTypes table, then the newly created type will not be added to the ExpenseTypes table.
Also the user should not populate the creationDate and modificationDate fields, because if he does, the type will not be added to the database. These fields will be handled automatically by the component (the current date will be used). When loading from the persistence, all the fields will be properly populated.

ExpenseEntryStatus
This class holds the information about an expense entry status.When creating an instance of this class the user has two options:
1) Use the default constructor and allow the GUID Generator component to generate a unique id
2) Use the parameterized constructor and provide an id for the ExpenseEntryStatus instance; if the id already is contained by another status from the ExpenseStatuses table, then the newly created status will not be added to the ExpenseStatuses table.
Also the user should not populate the creationDate and modificationDate fields, because if he does, the status will not be added to the database. This fields will be handled automatically by the component (the current date will be used). When loading from the persistence, all the fields will be properly populated.

CommonInfo
This abstract contains the common features of ExpenseEntry, ExpenseEntryType, ExpenseEntryStatus classes. This class is abstract because there is no need to instantiate it directly. It has no abstract methods; it is made abstract to group the common features of the data classes and to prevent direct instantiation.

ExpenseEntryTypeManager
The ExpenseEntryTypeManager class is a facade for the types management functionality.
It can do the following things:
-add a type(ExpenseEntryType instance) to the ExpenseTypes table; if the ExpenseEntryType instance has the id=-1 this manager will use the GUID Generator to generate an id for the ExpenseEntryType instance
-delete a type from the ExpenseTypes table
-delete all the types from the ExpenseTypes table

-retrieve a type(given its id) from the ExpenseTypes table
-retrieve all the types from the ExpenseTypes table
-update a type in the ExpenseTypes table
This manager is responsible for reading two properties from the configuration file. To accomplish this it will use the
Configuration Manager component. From the configuration file two properties will be read:
-a class name identifying an implementation of ExpenseEntryTypePersistence(this property is required)
-a connection producer name identifying a ConnectionProducer instance(this property is optional). This
instance of ConnectionProducer will provide the connection to the database.
Using these two properties this manager will create thourough reflection an ExpenseEntryTypePersistence implementation instance.

ExpenseEntryStatusManager
The ExpenseEntryStatusManager class is a facade for the statuses management functionality.
It can do the following things:
-add a status(ExpenseEntryStatus instance) to the ExpenseStatuses table; if the ExpenseEntryStatus instance has the id=-1 this manager will use the GUID Generator to generate an id for the ExpenseEntryStatus instance
-delete a status from the ExpenseStatuses table
-delete all the statuses from the ExpenseStatuses table
-retrieve a status(given its id) from the ExpenseStatuses table
-retrieve all the statuses from the ExpenseStatuses table
-update a status in the ExpenseStatuses table
This manager is responsible for reading two properties from the configuration file. To accomplish this it will use the
Configuration Manager component. From the configuration file two properties will be read:
-a class name identifying an implementation of ExpenseEntryStatusPersistence(this property is required)
-a connection producer name identifying a ConnectionProducer instance(this property is optional). This instance of ConnectionProducer will provide the connection to the database. Using these two properties this manager will create thourough reflection an ExpenseEntryStatusPersistence implementation instance.

ExpenseEntryManager
The ExpenseEntryManager class is a facade for the entries management functionality.
. It can do the following things:
-add an entry(ExpenseEntry instance) to the ExpenseEntries table; if the ExpenseEntry instance has the id=-1 this manager will use the GUID Generator to generate an id for the ExpenseEntry instance
-delete an entry from the ExpenseEntries table
-delete all the entries from the ExpenseEntries table
-retrieve an entry(given its id) from the ExpenseEntries table
-retrieve all the entries from the ExpenseEntries table
-update an entry in the ExpenseEntries table
This manager is responsible for reading two properties from the configuration file. To accomplish this it will use the
Configuration Manager component. From the configuration file two properties will be read:
-a class name identifying an implementation of ExpenseEntryPersistence(this property is required)

-a connection producer name identifying a ConnectionProducer instance(this property is optional). This
instance of ConnectionProducer will provide the connection to the database. Using these two properties this manager will create thourough reflection an ExpenseEntryPersistence implementation instance.

ExpenseEntryPersistence
ExpenseEntryPersistence represents the interface for expense entries access. Client can choose between alternative implementations to suit persistence migration. Interface defines all necessary methods to interact with the database. The methods exposes by this interface are very raw (it would be hard for a user to use them to obtain the functionality). They are aimed to an efficient database implementation (using INSERT, SELECT, UPDATE and DELETE statements) but other storage technologies can be used just as well (such as XML).

ExpenseEntryTypePersistence
ExpenseEntryTypePersistence represents the interface for expense entry types access. Client can choose between alternative implementations to suit persistence migration. Interface defines all necessary methods to interact with the database. The methods exposes by this interface are very raw (it would be hard for a user to use them to obtain the functionality). They are aimed to an efficient database implementation (using INSERT, SELECT, UPDATE and DELETE statements) but other storage technologies can be used just as well (such as XML).

ExpenseEntryStatusPersistence
ExpenseEntryTypePersistence represents the interface for expense entry types access. Client can choose between alternative implementations to suit persistence migration. Interface defines all necessary methods to interact with the database. The methods exposes by this interface are very raw (it would be hard for a user to use them to obtain the functionality). They are aimed to an efficient database implementation (using INSERT, SELECT, UPDATE and DELETE statements) but other storage technologies can be used just as well (such as XML).

ExpenseEntryDbPersistence
This class is a concrete implementation of the ExpenseEntryPersistence interface that uses an database as persistence. This implementation uses the DB Connection Factory component to obtain a connection to the database. Transaction should be employed to ensure atomicity. This class provides two constructors. The first is an empty constructor, and the second can be used to specify the connection producer name which will be used to obtain a connection. The connection will not be initialized in the constructors. It will
be initialized in one of the methods that will access the database; it will be initialized the first time one of this methods is called. It can be initialized using the setter or the initConnection method.

ExpenseEntryTypeDbPersistence
This class is a concrete implementation of the ExpenseEntryTypePersistence interface that uses an database as persistence.This implementation uses the DB Connection Factory component to obtain a connection to the database. Transaction should be employed to ensure atomicity. This class provides two constructors. The first is an empty constructor, and the second can be used to specify the connection producer name which will be used to obtain a connection. The connection will not be initialized in the constructors. It will be initialized in one of the methods that will access the database; it will be initialized the first time one of this methods is called. It can be initialized using the setter or the initConnection method.

ExpenseEntryStatusDbPersistence
This class is a concrete implementation of the ExpenseEntryStatusPersistence interface that uses an database as persistence. This implementation uses the DB Connection Factory component to obtain a connection to the database. Transaction should be employed to ensure atomicity. This class provides two constructors. The first is an empty constructor, and the second can be used to specify the connection producer name which will be used to obtain a connection. The connection will not be initialized in the constructors. It will be initialized in one of the methods that will access the database; it will be initialized the first time one of this methods is called. It can be initialized using the setter or the initConnection method.

1.5 Component Exception Definitions

Persistence Exception[custom]
The PersistenceException exception is used to wrap any persistence implementation specific exception. These exceptions are thrown by the persistence interfaces implementations. Since they are implementation specific, there needs to be a common way to report them to the user, and that is what this exception is used for. This exception is originally thrown in the persistence implementations. The business logic layer (the manager classes) will forward them to the user.

ConfigurationException[custom]
This exception is thrown by the managers if anything goes wrong in the process of loading the configuration file or if the information is missing or corrupt.

InsufficientDataException[custom]
This exception is thrown when some required fields (NOT NULL) are not set when creating or updating an entry, type or status in the persistence. This exception is thrown by the ExpenseEntryManager, ExpenseEntryTypeManager and ExpenseEntryStatusManager.

NullPointerException
This exception is thrown in various methods where null value is not acceptable.
Refer to the documentation in Poseidon for more details.

IllegalArgumentException
This exception is thrown in various methods if the given string argument is empty. Refer to the documentation in Poseidon for more details.

1.6 Thread Safety

This component is not thread safe. Thread safety was not a requirement. It is not thread safe because, for example, a user may request an entry while another may delete it at the same time. In order to achieve thread safety all the methods from the persistence layer have to be synchronized. I think that it is better not to add an overhead to this component by synchronizing all the methods from the persistence and let the application handle thread safety.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.4
- Compile target: Java 1.3, Java 1.4

2.2 TopCoder Software Components

- **Configuration Manager 2.1.3 –** used to retrieve the configured data. This component is used by getting its singleton instance with ConfigManager.getInstance(). Then the existsNamespace method should be used to determine whether the namespace is already loaded. If not, the add method is used to load the default configuration file. Finally, getString returns the values of the properties.
- **GUID Generator 1.0** is used to assign unique ids to records. This component has the advantage of not requiring persistent storage (such as ID Generator requires), making the component easier to use. A generator is obtained with UUIDUtility.getGenerator(UUIDType.TYPEINT32). Then using generator.getNextUUID().toString() ids are generated as needed.
- **Base Exception 1.0** is used as a base class for the all the custom exceptions defined in this component. The purpose of this component is to provide a consistent way to handle the cause exception for both JDK 1.3 and JDK 1.4.
- **DB Connection Factory 1.0** provides a simple but flexible framework allowing the clients to obtain the connections to a SQL database without providing any implementation details. This component is used to obtain a connection to a database.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.timetracker.entry.expense

3.2 Configuration Parameters

Parameter	Description	Values
connection_producer_name	Identifies a ConnectionProducer which will be used to obtain a connection to a database.**Optional**	Expense_Entry _Connection _Producer
entry_persistence_class	Fully qualified class name of the ExpenseEntryPersistence implementation. **Required**.	ExpenseEntryDbPersistence
entry_type_persistence_class	Fully qualified class name of the ExpenseEntryTypePersistence implementation. **Required**.	ExpenseEntryTypeDbPersistence
entry_status_persistence_class	Fully qualified class name of the ExpenseEntryStatusPersistence implementation. **Required**.	ExpnseEntryStatusDbPersistence

Here is an example of the configuration file:

```
<?xml version="1.0" ?>
  <CMConfig>
     <Config name=" com.topcoder.timetracker.entry.expense">
          <!--the  class name of the entry persistence-->
          <property name=" entry_persistence_class">
              <value>ExpenseEntryDbPersistence</value>
          </property>
```

```
        <!--the  class name of the entry type persistence-->
        <property name=" entry_type_persistence_class">
            <value>ExpenseEntryTypeDbPersistence</value>
        </property>
        <!--the  class name of the entry persistence-->
        <property name=" entry_status_persistence_class">
            <value>ExpenseEntryStatusDbPersistence</value>
        </property>
        <!--the  name identifying a ConnectionProducer instance-->
        <property name=" connection_producer_name">
            <value>Expense_Entry_Connection_Producer</value>
        </property>
</Config>
```

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

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

4.2 Required steps to use the component

Extract the component distribution.

4.3 Demo

For this demo I will use the configuration file shown above.

```
//create an ExpenseEntryType
ExpenseEntryType type1 = new ExpenseEntryType();
//set the fields of type1
type1.setDescription("Air Transportation");
type1.setCreationUser("John");
type1.setModificationUser("Tom");

//create another ExpenseEntryType
//this time supply the id
//if the id exist in the ExpenseTypes table then type2 will not be added to the table
ExpenseEntryType type2 = new ExpenseEntryType(2);
//set the fields of type2
type2.setDescription("Car Rental");
type2.setCreationUser("Jimmy");
type2.setModificationUser("Aristotel");

//create an ExpenseEntryTypeManager
ExpenseEntryTypeManager typeManager = new
ExpenseEntryTypeManager("com.topcoder.timetracker.entry.expense");


//add the types to the ExpenseTypes table
typeManager.addType(type1);
typeManager.addType(type2);
//assume that both types have been added to the ExpenseTypes table
```

```
//create an ExpenseEntryStatus
ExpenseEntryStatus status1 = new ExpenseEntryStatus();
//set the fields of the status1
status1.setDescription("Approved");
status1.setCreationUser("John");
status1.setModificationUser("Tom");

//create another ExpenseEntryStatus
//this time supply the id
//if the id exist in the ExpenseStatuses table then status2 will not be added to the
//table
ExpenseEntryStatus status2 = new ExpenseEntryStatus(3);
//set the fields of the status1
status2.setDescription("Not Approved");
status2.setCreationUser("Mike");
status2.setModificationUser("Tom");

//create an ExpenseEntryStatusManager
ExpenseEntryStatusManager statusManager = new
ExpenseEntryStatusManager("com.topcoder.timetracker.entry.expense");

//add the statuses to the ExpenseStatuses table
statusManager.addStatus(status1);
statusManager.addStatus(status2);
//assume that both statuses have been added to the ExpenseStatuses table

//create an ExpenseEntry
ExpenseEntry entry1 = new ExpenseEntry();
//set the fields of entry1
entry1.setDescription("project Ohio");
entry1.setCreationUser("George");
entry1.setModificationUser("George);
entry1.setDate(200000);
entry1.setExpenseType(type1);
entry1.setExpenseStatus(status1);
entry1.setBillable(true);
double amount =20000;
entry1.setAmount(amount);

//create another ExpenseEntry
//this time supply the id
//if the id exist in the ExpenseEntries table then entry2 will not be added to the
//table
ExpenseEntry entry2 = new ExpenseEntry(3);
//set the fields of entry2
entry2.setDescription("project New York");
entry2.setCreationUser("Alonso");
entry2.setModificationUser("Alonso);
entry2.setDate(200000);
entry2.setExpenseType(type2);
entry2.setExpenseStatus(status2);
entry2.setBillable(false);
amount =200000;
entry2.setAmount(amount);
```

```
//create an ExpenseEntryManager
ExpenseEntryManager entryManager = new
ExpenseEntryManager("com.topcoder.timetracker.entry.expense");

//add the entries to the ExpenseEntries table
entryManager.addStatus(entry1);
entryManager.addStatus(entry2);
//assume that both entries have been added to the ExpenseEntries table

//retrieve type(s), status(es) and entry(entries) from the database
ExpenseEntryType type3 = typeManager.retrieveType(2);
System.out.println("description = "+type3.getDescription());
ExpenseEntryStatus status3 = statusManager.retrieveStatus(2);
System.out.println("creation user = "+status2.getCreationUser());
ExpenseEntry entry3 = entryManager.retrieveEntry(3);
System.out.println("is billable = "+entry3.isBillable());

List entries = entryManager.retrieveAllEntries();
for(int i=0;i<entries.size();i++){
   ExpenseEntry e = (ExpenseEntry)entries.get(i);
   System.out.println("is billable+"e.isBillable());
}

List types = typeManager.retrieveAllTypes();
for(int i=0;i<types.size();i++){
   ExpenseEntryType t = (ExpenseEntryType)types.get(i);
   System.out.println("description+"t.getDescription());
}

List statuses = statusManager.retrieveAllStatuses();
for(int i=0;i<statuses.size();i++){
   ExpenseEntryStatus s = (ExpenseEntryStatus)statuses.get(i);
   System.out.println("description+"t.isBillable());
}

//update an entry
entry3.setBillabel(true);
entryManager.updateEntry(entry3);

//update a type
type3.setModificationUser("Pam);
typeManager.updateType(type3);

//update a status
status3.setDescription("Pending Approval");
statusManager.updateStatus(status3);

//delete an entry
if (entryManager.deleteEntry(3))
   System.out.println("entry was deleted");

//delete all entries
entryManager.deleteAllEntries();

//delete a type
if (typeManager.deleteType(2))
```

```
    System.out.println(“type was deleted”);

//delete all types
typeManager.deleteAllTypes();

//delete a status
if (statusManager.deleteStatus(3))
    System.out.println(“status was deleted”);

//delete all statuses
statusManager.deleteAllStatuses();
```

5. Future Enhancements

More ExpenseEntryPersistence implementations to this component.