Time_Tracker_Project 1.0 Component Specification

1. Design

The Time Tracker Project custom component is part of the Time Tracker application. It provides an abstraction of projects and the clients that the projects are assigned to. This component handles the persistence and other business logic required by the application.
The manager classes handle the business logic. They provide a sort of façade for the user to the functionality offered by this component: client management and project management. The following classes implement the model part of this application: Client, ProjectWorker, Project, ProjectManager..
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.
An observation: in this release a project can have only one client and only one manager(this was decided on the forum).

1.1 Design Patterns

The **strategy** pattern is used for abstracting the persistence implementation in the TimeTrackerProjectPersistence interface.
The **data value object** pattern is used in the Client, Project, ProjectWorker and ProjectManager 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

The sql statements are very simple select, insert, update and delete queries so, is not necessary to provide them as algorithms.

1) I will provide an algorithm for the constructor of ProjectPersistenceManager.
The algorithm is:

```
//read the configuration file
ConfigManager cm = ConfigManager.getInstance();
String className = cm.getStringArray(namespace,"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 = (TimeTrackerProjectPersistence)c.newInstance();
} else {
         Constructor constr = c.getConstructor(new Class[]{Class.forName("String"});
         persistence = (TimeTrackerProjectPersistence) constr.newInstance(new
                                                   Object[]{connectionProducerName)});
}
```

2) Id generation:
-obtain a Generator (Int32Generator instance):
  UUIDUtility.getGenerator(UUIDType.TYPEINT32)
-obtain an UUID (UUID32Implementation instance)
 UUID uuid = generator.getNextUUID().
 In this case an UUID32Implementation instance is obtained. UUID32Implementation represents the 32-bit UUID implementation of the UUID interface. Since 32 bits are used there is no problem in obtaining an int from the uuid.
-obtain a String from the UUID
 String stringId = uuid.toString()
-obtain an id:
 int id = (new Integer(stringId)).intValue();

1.4 Component Class Overview

Project
This class holds the information about project. 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 one of the parameterized constructors and provide an id for the Project instance; if the id already is contained by another project from the Projects table, then the newly created project will not be added to the Projects 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.

Client
This class holds the information about a client. 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 one of the parameterized constructors and provide an id for the Client instance; if the id already is contained by another client from the Clients table, then the newly created client will not be added to the Clients table.
Also the user should not populate the creationDate and modificationDate fields, because if he does, the client 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.

ProcesManager
This class holds the information about an project manager. It has the following fields: project, managerId, creationDate, creationUser, modificationDate, modificationUser. When creating an instance of this class the user has two options:
1) Use the default constructor which does nothing
2) Use the parameterized constructor and provide an id and a project argument. The user should not populate the creationDate and modificationDate fields, because if he does, the project manager 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.

ProcesWorker

This class holds the information about an project worker. It has the following fields: project, workerId, startDate, endDate, payRate, creationDate, creationUser, modificationDate, modificationUser. When creating an instance of this class the user has two options:
1) Use the default constructor which does nothing
2) Use the parameterized constructor and provide an id and a project argument. The user should not populate the creationDate and modificationDate fields, because if he does, the project worker 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.

ProjectPersistenceManager
This manager is responsible for reading the configuration file. To accomplish this it will use the Configuration Manager component. From the configuration file two properties can be read:
-a class name identifying an implementation of TimeTrackerProjectPersistence(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 thorough reflection an TimeTrackerProjectPersistence implementation instance which will be used by the other two managers. These managers will use the getPersistence method to access the TimeTrackerProjectPersistence.

ClientUtility
The ClientUtility is useful to manage the clients.
. It can do the following things:
-add a client(Client instance) to the Clients table; if the Client instance has the id=-1 this manager will use the GUID Generator to generate an id for the Client instance
-delete a client from the Clients table
-delete all the clients from the Clients table
-retrieve a client(given its id) from the Clients table
-retrieve all the clients from the Clients table
-update a client in the Clients table
-add a project to a client
-retrieve a certain project(specified by its id) of a client
This manager will receive in the constructor an ProjectPersistenceManager instance. It will use this instance to gain access to the database. It will use the getPersistence method from the ProjectPersistenceManager instance to obtain the persistence

ProjectUtility
The ProjectUtility is useful to manage the projects.
. It can do the following things:
-add a Project(Project instance) to the Projects table; if the Project instance has the id=-1 this manager will use the GUID Generator to generate an id for the Project instance
-delete a project from the Projects table
-delete all the clients from the Projects table
-retrieve a project(given its id) from the Projects table
-retrieve all the projects from the Projects table
-update a project in the Projects table
-assign a client to a project
-retrieve the client of a specified project
-add/remove/update/retrieve workers
-assign/remove/retrieve project manager

-add/remove/retrieve time entries
-add/remove/retrieve expense entries
This manager will receive in the constructor an ProjectPersistenceManager instance. It will use this instance to gain access to the database. It will use the getPersistence method from the ProjectPersistenceManager instance to obtain the persistence

TimeTrackerProjectPersistence
TimeTrackerProjectPersistence represents the interface for data access. Client can choose between alternative implementations to suit persistence migration. Interface defines all necessary methods to interact with the database. This release comes with an Informix implementation. 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).

InformixTimeTrackerProjectPersistence
This class is a concrete implementation of the TimeTrackerProjectPersistence interface that uses an Informix 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. If the default constructor is used to create an instance of this class, then the DEFAULT_CONNECTION_PRODUCER_NAME constant will be used to obtain a connection from the DB Connection Factory component. If the parameterized constructor is used then the user has the possibility to specify a name that will be passed to the DB Connection Factory component to obtain a connection to the Informix database.

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 TimeTrackerProjectPersistence interface 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 ProjectPersistenceManager 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 ProjectUtility and ClientUtility.

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.
Conflicts may occur in the persistence layer. In order to avoid them one would have to synchronize all the methods from the persistence implementations, to avoid concurrent database access. In addition, before making any change to the database, one would have to make sure the existing data is still valid.

In order to achieve this goal the database operations have to be atomic. The solution is to use JDBC transactions. The Connection class provides the setAutoCommit method to switch into manual commit mode. After that, the SQL statement that need to be executed in one transaction are executed in sequence. At the end the commit or rollback methods are used depending on whether everything was successful or an error occurred. If commit is successful then we have the guarantee that everything was executed atomically.

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 an Informix database(in the current implementation of Expense Entry component).

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.timetracker.project

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**	Informix_Connection _Producer
persistence_class	Fully qualified class name of the TimeTrackerProjectPersistence implementation. **Required**.	InformixTimeTrackerProjectPersistence

Here is an example of the configuration file:

```
<?xml version="1.0" ?>
  <CMConfig>
      <Config name=" com.topcoder.timetracker.project">
           <!--the  class name of the persistence-->
           <property name=" persistence_class">
               <value>InformixTimeTrackerProjectPersistence</value>
          </property>
           <!--the  name identifying a ConnectionProducer instance-->
           <property name=" connection_producer_name">
               <value> Informix_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 a ProjectPersistenceManager
ProjectPersistenceManager pm = new
        ProjectPersistenceManager(“com.topcoder.timetracker”);

//create a Project instance using the parameterized constructor
Project project = new Project(1);
//using the setters the fields should be initialized; it is not necessary to perform
//this task in this demo since it is a trivial one, and the demo will get very big

//create a ProjectUtility
```

```
ProjectUtility pu = new ProjectUtility(pm);

//add the project to the database
pu.addProject(project);

//create a ProjectManager instance
ProjectManager projectManager = new ProjectManager(project, 1);

//assign the projectManager to the project
pu.assignProjectManager(projectManager);

//change the description of the project
project.setDescription(“Renault project”);

//update the project
pu.updateProject(project);

//query the project manager of the project
ProjectManager pm1 = pu.getProjectManager(project.getId());

//create a ProjectWorker
//ProjectWorker worker1 = new ProjectWorker(project,2);

//add the worker to the project(in the database)
pu.addWorker(worker1);

//create a second ProjectWorker
//ProjectWorker worke2r = new ProjectWorker(project,3);

//add the worker to the project(in the database)
pu.addWorker(worker2);

//update a worker
double PayRate = 100;
worker2.setPayRate(payRate);
pu.updateWorker(worker2);

//get a worker from the project
ProjectWorker worker3 = pu.getWorker(2,1);

//get all the workers from the project
List l = pu.getWorkers(1);

//add an expense entry to the project
pu.addExpenseEntry(5,1,”John”);

//get all the expense entries from a project
List expenses = pu.getExpenseEntries(1);

//add a time entry to the project
pu.addTimeEntry(4,1,”John”);

//get all the time entries from a project
List times = pu.getTimeEntries(1);

//create a ClientUtility instance
```

```
ClientUtility cu = new ClientUtility(pm);

//create a client using the parameterized constructor
Client client = new Client(5);
//using setters assume that the fields are initialized

//add the client to the database
cu.addClient(client);

//update the client description
client.setDescription(“bussinesmen”);

//update the client (in the database)
cu.updateClient(client);

//add a project to the client
cu.addProjectToClient(client.getId(), project);

//retrieve the projects of a client
List proj = cu.getAllClientProjects(client.getId());

//create another project
Project project1 = new Project(2);
//assume that the fields are initialized using the setters

//add the project to the datatbase
pu.addProject(project1);

//assign a client to project1
pu.assignClient(2,5,”Tim”);

//query the client of a project
Client client1 = pu.getProjectClient(project1.getId());

//remove a worker
pu.removeWorker(2,1);

//remove all workers from a project
pu.removeWorkers(1);

//remove a time entry from a project
pu.removeTimeEntry(4,1);

//remove an expense entry from a project
pu.removeExpenseEntry(5,1);


//remove the project manager from a project
pu.removeProjectManager(2,1);

//remove a client
cu.removeClient(5);

//remove all the clients
cu.removeAllClients();
```

5. Future Enhancements

More TimeTrackerProjectPersistence implementations to this component.